‹packt›

1ST EDITION

ChatGPT for Cybersecurity Cookbook

Learn practical generative AI recipes to supercharge your cybersecurity skills

CLINT BODUNGEN

Foreword by Aaron Crow, OT Cybersecurity Professional & Thought Leader
Host of PrOTect IT All Podcast

ChatGPT for Cybersecurity Cookbook

Learn practical generative AI recipes to supercharge your cybersecurity skills

Clint Bodungen

‹packt›

ChatGPT for Cybersecurity Cookbook

Group Product Manager: Niranjan Naikwadi
Publishing Product Manager: Nitin Nainani
Book Project Manager: Aishwarya Mohan
Senior Editors: Aamir Ahmed and Nathanya Dias
Technical Editor: Simran Haresh Udasi
Copy Editor: Safis Editing
Indexer: Manju Arasan
Production Designer: Shankar Kalbhor
DevRel Marketing Coordinator: Vinishka Kalra

First published: March 2024
Production reference: 1130324

Published by Packt Publishing Ltd.
Grosvenor House
11 St Paul's Square
Birmingham
B3 1RB, UK

ISBN 978-1-80512-404-7

`www.packtpub.com`

To my wife, Ashley, for her unwavering support throughout the many weekends and hours burning the midnight oil for this project. To my sons, Caleb and Connor, the future is what you make of it. You can accomplish anything if you just believe.

– Clint Bodungen

Foreword

In the relentless cyber battleground, where threats morph with each tick of the clock, generative **artificial intelligence** (**AI**) emerges as our digital sentinel. ChatGPT and its kin are not mere tools; they are force multipliers in our cyber arsenals. We're talking about a paradigm shift here – generative AI doesn't just uplift; it transforms the cybersecurity landscape. It lets us run rings around potential threats, streamline security measures, and forecast nefarious plots with an astuteness that's simply otherworldly.

This isn't just tech talk; it's about real muscle in the fight against digital adversaries. Imagine crafting a cyber training regimen so robust that it catapults neophytes into seasoned defenders within the data trenches. Generative AI is that game-changer, shattering the barriers to entry, democratizing the field, and nurturing a new generation of cyber mavens.

But there's more. With generative AI, we dive into data oceans and surface with those elusive security insights – the kind that traditional tools would miss. This is about harnessing AI to not just respond to threats but also to anticipate them, to be steps ahead of the adversary. We're entering an era where our collaboration with AI amplifies our strategic nous, sharpens our foresight, and fortifies our resilience.

As we join forces with AI, we're not just bolstering defenses; we're fostering a culture of cybersecurity innovation. We're empowering minds to push beyond the conventional, to envision a digital realm where safety is the norm, not the exception. This book is a testament to that vision, a guide on wielding AI's might to safeguard our cyber frontiers. Welcome to the future – a future where we stand united with AI in the vanguard of cybersecurity.

Aaron Crow

OT Cybersecurity Professional & Thought Leader

Host of PrOTect IT All Podcast

Contributors

About the author

Clint Bodungen is a globally recognized cybersecurity professional and thought leader with 25+ years of experience, and author of *Hacking Exposed: Industrial Control Systems*. He is a U.S. Air Force veteran, has worked for notable cybersecurity firms Symantec, Booz Allen Hamilton, and Kaspersky Lab, and is a co-founder of ThreatGEN, a cybersecurity gamification and training firm. Clint has been at the forefront of integrating gamification and AI into cybersecurity with his flagship product, *ThreatGEN® Red vs. Blue*, the world's first online multiplayer computer game designed to teach real-world cybersecurity. Clint continues his pursuit to help revolutionize the cybersecurity industry using gamification and generative AI.

I would first like to thank my amazing team at Packt Publishing for their patience and their trust in me to write this book. And special thanks to the cybersecurity community and the pioneers of the AI industry.

About the reviewers

Aaron Shbeeb is a lifelong programmer, cybersecurity enthusiast, and game developer. He has programmed in over a dozen programming languages both personally and professionally. He has also worked as a penetration tester and vulnerability researcher. Lately, his passion has been for developing *ThreatGEN® Red vs. Blue*, a cybersecurity training video game that he co-founded/co-developed with Clint Bodungen. Developing that game allows him to practice some of his favorite parts of software development such as system design, machine learning, and AI.

Pascal Ackerman, a principal security consultant, began his career in IT in 1999. He is a seasoned industrial security professional with a degree in electrical engineering and experience in industrial network design and support, information and network security, risk assessments, penetration testing, threat hunting, and forensics. His passion lies in analyzing new and existing threats to **Industrial Control System** (**ICS**) environments and he fights cyber adversaries both from his home base and while traveling the world with his family as a digital nomad.

Bradley Jackson navigates the intricate world of cybersecurity with a quiet dedication to Python and emerging technologies. His journey, though marked by meaningful professional accomplishments, finds its truest joy in life's simpler facets. At heart, Bradley is a family man, deeply devoted to his wife Kayla and their four children. This grounding influence of family life in Arkansas beautifully complements his thoughtful contributions to the *ChatGPT for Cybersecurity Cookbook*, reflecting a blend of practical wisdom with a down-to-earth approach to technology.

Table of Contents

Preface

In the ever-evolving domain of cybersecurity, the advent of generative AI and **large language models** (**LLMs**), epitomized by the introduction of ChatGPT by OpenAI, marks a significant leap forward. This book, dedicated to the exploration of ChatGPT's applications within cybersecurity, embarks on a journey from the tool's nascent stages as a basic chat interface to its current stature as an advanced platform reshaping cybersecurity methodologies.

Initially conceptualized to aid AI research through the analysis of user interactions, ChatGPT's journey from its initial release in late 2022 to its current form illustrates a remarkable evolution in a span of just over a year. The integration of sophisticated features such as web browsing, document analysis, and image creation through DALL-E, combined with advancements in speech recognition and text-to-image understanding, has transformed ChatGPT into a multi-faceted tool. This transformation is not merely technical but extends into functional realms, potentially significantly impacting cybersecurity practices.

A key facet in ChatGPT's evolution was the incorporation of code completion and debugging functionalities, which expanded its utility across technical domains, particularly in software development and secure coding. These advancements have significantly enhanced coding speed and efficiency and have effectively democratized programming skills and accessibility.

The Advanced Data Analysis feature (formerly known as Code Interpreter) has further opened new avenues in cybersecurity. It enables professionals to rapidly analyze and debug security-related code, automate the creation of secure coding guidelines, and develop custom security scripts. The capability to process and visualize data from diverse sources, including documents and images, and to generate detailed charts and graphs, transforms raw data into actionable cybersecurity insights.

ChatGPT's web-browsing capabilities have greatly enhanced its role in cybersecurity intelligence gathering. By enabling professionals to extract real-time threat information from a broad spectrum of online sources, ChatGPT facilitates a rapid response to emerging threats and supports informed strategic decision-making. This synthesis of data into concise, actionable intelligence underscores ChatGPT's value as a dynamic tool for cybersecurity experts navigating the rapidly evolving landscape of cyber threats.

Finally, this book extends beyond the confines of the ChatGPT web interface, venturing into the OpenAI API to unlock a world of possibilities, empowering you to not only utilize but also innovate with the OpenAI API. By delving into the creation of custom tools and expanding upon the inherent capabilities of the ChatGPT interface, you are equipped to tailor AI-powered solutions to their unique cybersecurity challenges.

This book serves as a quintessential guide for cybersecurity professionals looking to harness the power of ChatGPT in their projects and tasks by providing practical, step-by-step examples of how to employ ChatGPT in real-world scenarios.

Each chapter focuses on a unique facet of cybersecurity, from vulnerability assessment and code analysis to threat intelligence and incident response. Through these chapters, you are introduced to the innovative application of ChatGPT in creating vulnerability and threat assessment plans, analyzing and debugging security-related code, and even generating detailed threat reports. The book delves into using ChatGPT in conjunction with frameworks such as MITRE ATT&CK, automating the creation of secure coding guidelines, and crafting custom security scripts, thereby offering a comprehensive toolkit for enhancing cybersecurity infrastructure.

By integrating the advanced capabilities of ChatGPT, this book not only educates but also inspires professionals to explore new horizons in cybersecurity, making it an indispensable resource in the age of AI-driven security solutions.

Who this book is for

ChatGPT for Cybersecurity Cookbook is written for a diverse audience with a shared interest in the intersection of artificial intelligence and cybersecurity. Whether you are a seasoned cybersecurity professional aiming to incorporate the innovative capabilities of ChatGPT and the OpenAI API into your security practices, an IT professional eager to broaden your cybersecurity acumen with AI-powered tools, a student or emerging cybersecurity enthusiast keen on understanding and applying AI in security contexts, or a security researcher fascinated by the transformative potential of AI in cybersecurity, this book is tailored for you.

The content is structured to accommodate a spectrum of knowledge levels, initiating you with fundamental concepts before advancing to sophisticated applications. This inclusive approach ensures the book's relevance and accessibility to individuals across various stages of their cybersecurity journey.

What this book covers

Chapter 1, *Getting Started: ChatGPT, the OpenAI API, and Prompt Engineering*, introduces ChatGPT and the OpenAI API, laying the foundation for leveraging generative AI in cybersecurity. It covers the basics of setting up an account, mastering prompt engineering, and utilizing ChatGPT for tasks including code writing and role simulation, setting the stage for more advanced applications in subsequent chapters.

Chapter 2, *Vulnerability Assessment*, focuses on enhancing vulnerability assessment tasks, guiding you through using ChatGPT to create assessment plans, automate processes with the OpenAI API, and integrate with frameworks including MITRE ATT&CK for comprehensive threat reporting and analysis.

Chapter 3, *Code Analysis and Secure Development*, delves into the **secure software development lifecycle** (**SSDLC**), showing how ChatGPT can streamline the process from planning to maintenance. It highlights the use of AI in crafting security requirements, identifying vulnerabilities, and generating documentation to improve software security and maintainability.

Chapter 4, *Governance, Risk, and Compliance (GRC)*, offers insights into using ChatGPT for enhancing cybersecurity governance, risk management, and compliance efforts. It covers generating cybersecurity policies, deciphering complex standards, conducting cyber risk assessments, and creating risk reports to strengthen cybersecurity frameworks.

Chapter 5, *Security Awareness and Training*, focuses on leveraging ChatGPT in cybersecurity education and training. It explores creating engaging training materials, interactive assessments, phishing training tools, exam preparation aids, and employing gamification to enhance learning experiences in cybersecurity.

Chapter 6, *Red Teaming and Penetration Testing*, explores AI-enhanced techniques for red teaming and penetration testing. It includes generating realistic scenarios using the MITRE ATT&CK framework, conducting OSINT reconnaissance, automating asset discovery, and integrating AI with penetration testing tools for comprehensive security assessments.

Chapter 7, *Threat Monitoring and Detection*, addresses the use of ChatGPT in threat intelligence analysis, real-time log analysis, detecting **advanced persistent threats** (**APTs**), customizing threat detection rules, and using network traffic analysis to improve threat detection and response capabilities.

Chapter 8, *Incident Response*, focuses on utilizing ChatGPT to enhance incident response processes, including incident analysis, playbook generation, root cause analysis, and automating report creation to ensure efficient and effective responses to cybersecurity incidents.

Chapter 9, *Using Local Models and Other Frameworks*, investigates the use of local AI models and frameworks in cybersecurity, highlighting tools such as LMStudio and Hugging Face AutoTrain for privacy-enhanced threat hunting, penetration testing, and sensitive document review.

Chapter 10, *The Latest OpenAI Features*, provides an overview of the most recent OpenAI features and their applications in cybersecurity. It emphasizes leveraging ChatGPT's advanced capabilities for cyber threat intelligence, security data analysis, and employing visualization techniques for a deeper understanding of vulnerabilities.

To get the most out of this book

To maximize the benefits derived from this book, you are encouraged to possess the following:

- A foundational grasp of cybersecurity principles, including prevalent terminology and best practices, to contextualize the applications of ChatGPT within the security landscape. (*This book is not intended to be an introduction to cybersecurity.*)

- An understanding of programming fundamentals, particularly in Python, as the book employs Python scripts extensively to demonstrate interactions with the OpenAI API.
- Proficiency with command-line interfaces and a rudimentary knowledge of networking concepts, essential for executing the practical exercises and understanding the cybersecurity applications discussed.
- A basic familiarity with web technologies such as HTML and JavaScript, which underpin several web application security and penetration testing examples presented in the book.

Software/hardware covered in the book	**OS requirements**
Python 3.10 or higher	Windows, macOS, and Linux (any)
A code editor (such as VS Code)	Windows, macOS, and Linux (any)
A command-line/terminal application	Windows, macOS, and Linux (any)

If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Important note

Generative AI and LLM technology is evolving extremely fast, so much so that in some cases you will discover that some examples in this book might already be outdated and not function as intended due to recent API and/or AI model updates, and even the ChatGPT web interface itself. As such, it is imperative to reference the most recent code and notes for this book from the official GitHub repository. Every effort will be made to keep the code up to date in order to reflect the latest changes and updates by OpenAI and other technology providers used throughout this book.

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`. If there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Code in Action

Code in Action videos for this book can be viewed at (`https://bit.ly/3uNma17`).

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "If you are using a different shell configuration file, replace `~/.bashrc` with the appropriate file (for example, `., ~/.zshrc` or `~/.profile`)."

A block of code is set as follows:

```
import requests
url = "http://localhost:8001/v1/chat/completions"
headers = {"Content-Type": "application/json"}
data = { "messages": [{"content": "Analyze the Incident Response Plan
for key strategies"}], "use_context": True, "context_filter": None,
"include_sources": False, "stream": False }
response = requests.post(url, headers=headers, json=data)
result = response.json() print(result)
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "In the **System Properties** window, click the **Environment Variables** button."

Tips or important notes

Appear like this.

Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it...*, *How it works...*, *There's more...*, and *See also*).

To give clear instructions on how to complete a recipe, use these sections as follows:

Getting ready

This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

How to do it...

This section contains the steps required to follow the recipe.

How it works...

This section usually consists of a detailed explanation of what happened in the previous section.

There's more...

This section consists of additional information about the recipe in order to make you more knowledgeable about the recipe.

See also

This section provides helpful links to other useful information for the recipe.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read *ChatGPT for Cybersecurity Cookbook*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

1
Getting Started: ChatGPT, the OpenAI API, and Prompt Engineering

ChatGPT is a **large language model** (**LLM**) developed by **OpenAI**, which is specifically designed to generate context-aware responses and content based on the prompts provided by users. It leverages the power of **generative AI** to understand and respond intelligently to a wide range of queries, making it a valuable tool for numerous applications, including cybersecurity.

Important note

Generative AI is a branch of **artificial intelligence** (**AI**) that uses **machine learning** (**ML**) algorithms and **natural language processing** (**NLP**) to analyze patterns and structures within a dataset and generate new data that resembles the original dataset. You likely use this technology every day if you use autocorrect in word processing applications, mobile chat apps, and more. That said, the advent of LLMs goes far beyond simple autocomplete.

LLMs are a type of generative AI that are trained on massive amounts of text data, enabling them to understand context, generate human-like responses, and create content based on user input. You may have already used LLMs if you have ever communicated with a helpdesk chatbot.

GPT stands for **Generative Pre-Trained Transformer** and, as the name suggests, is an LLM that has been pre-trained to improve accuracy and/or provide specific knowledge-based data generation.

ChatGPT has raised concerns about plagiarism in some academic and content-creation communities. It has also been implicated in misinformation and social engineering campaigns due to its ability to generate realistic and human-like text. However, its potential to revolutionize various industries cannot be ignored. In particular, LLMs have shown great promise in more technical fields, such as programming and cybersecurity, due to their deep knowledge base and ability to perform complex tasks such as instantly analyzing data and even writing fully functional code.

In this chapter, we will guide you through the process of setting up an account with OpenAI, familiarizing yourself with ChatGPT, and mastering the art of prompt engineering (the key to leveraging the real power of this technology). We will also introduce you to the OpenAI API, equipping you with the necessary tools and techniques to harness ChatGPT's full potential.

You'll begin by learning how to create a ChatGPT account and generate an API key, which serves as your unique access point to the OpenAI platform. We'll then explore basic ChatGPT prompting techniques using various cybersecurity applications, such as instructing ChatGPT to write Python code that finds your IP address and simulating an AI CISO role by applying ChatGPT roles.

We'll dive deeper into enhancing your ChatGPT outputs with templates to generate comprehensive threat reports, as well as formatting output as tables for improved presentation, such as creating a security controls table. As you progress through this chapter, you'll learn how to set the OpenAI API key as an environment variable to streamline your development process, send requests and handle responses with Python, efficiently use files for prompts and API key access, and effectively employ prompt variables to create versatile applications, such as generating manual pages based on user inputs. By the end of this chapter, you'll have a solid understanding of the various aspects of ChatGPT and how to utilize its capabilities in the cybersecurity domain.

Tip

Even if you are already familiar with the basic ChatGPT and OpenAI API setup and mechanics, it will still be advantageous for you to review the recipes in *Chapter 1* as they are almost all set within the context of cybersecurity, which is reflected through some of the prompting examples.

In this chapter, we will cover the following recipes:

- Setting up a ChatGPT Account
- Creating an API Key and interacting with OpenAI
- Basic prompting (Application: Finding Your IP Address)
- Applying ChatGPT Roles (Application: AI CISO)
- Enhancing Output with Templates (Application: Threat Report)
- Formatting Output as a Table (Application: Security Controls Table)
- Setting the OpenAI API Key as an Environment Variable

- Sending API Requests and Handling Responses with Python
- Using Files for Prompts and API Key Access
- Using Prompt Variables (Application: Manual Page Generator)

Technical requirements

For this chapter, you will need a **web browser** and a stable **internet connection** to access the ChatGPT platform and set up your account. Basic familiarity with the Python programming language and working with the command line is necessary as you'll be using **Python 3.x**, which needs to be installed on your system so that you can work with the OpenAI GPT API and create Python scripts. A **code editor** will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter.

The code files for this chapter can be found here: `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

Setting up a ChatGPT Account

In this recipe, we will learn about generative AI, LLMs, and ChatGPT. Then, we will guide you through the process of setting up an account with OpenAI and exploring the features it offers.

Getting ready

To set up a ChatGPT account, you will need an active email address and a modern web browser.

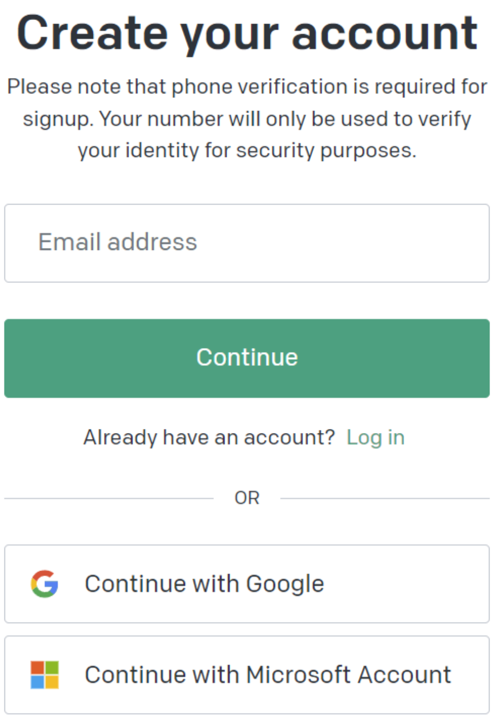
Important note

Every effort has been made to ensure that every illustration and instruction is correct at the time of writing. However, this is such a fast-moving technology and many of the tools used in this book are currently being updated at a rapid pace. Therefore, you might find slight differences.

How to do it…

By setting up a ChatGPT account, you'll gain access to a powerful AI tool that can greatly enhance your cybersecurity workflow. In this section, we'll walk you through the steps of creating an account, allowing you to leverage ChatGPT's capabilities for a range of applications, from threat analysis to generating security reports:

1. Visit the OpenAI website at `https://platform.openai.com/` and click **Sign up**.
2. Enter your email address and click **Continue**. Alternatively, you can register with your existing Google or Microsoft account:


Create your account

Please note that phone verification is required for signup. Your number will only be used to verify your identity for security purposes.

Email address
Continue
Already have an account? Log in
OR
Continue with Google
Continue with Microsoft Account

Figure 1.1 – OpenAI signup form

3. Enter a strong password and click **Continue**.
4. Check your email for a verification message from OpenAI. Click the link provided in the email to verify your account.
5. Once your account has been verified, enter the required information (first name, last name, optional organization name, and birthday) and click **Continue**.
6. Enter your phone number to verify by phone and click **Send code**.
7. When you receive the text message with the code, enter the code and click **Continue**.
8. Visit and bookmark `https://platform.openai.com/docs/` to start becoming familiar with OpenAI's documentation and features.

How it works…

By setting up an account with OpenAI, you gain access to the ChatGPT API and other features offered by the platform, such as **Playground** and all available models. This enables you to utilize ChatGPT's capabilities in your cybersecurity operations, enhancing your efficiency and decision-making process.

There's more...

When you sign up for a free OpenAI account, you get $18 in free credits. While you most likely won't use up all of your free credits throughout the recipes in this book, you will eventually with continued use. Consider upgrading to a paid OpenAI plan to access additional features, such as increased API usage limits and priority access to new features and improvements:

- **Upgrading to ChatGPT Plus**:

  ChatGPT Plus is a subscription plan that offers additional benefits beyond free access to ChatGPT. With a ChatGPT Plus subscription, you can expect faster response times, general access to ChatGPT even during peak times, and priority access to new features and improvements (this includes access to GPT-4 at the time of writing). This subscription is designed to provide an enhanced user experience and ensure that you can make the most out of ChatGPT for your cybersecurity needs.

- **Benefits of having an API key**:

  Having an API key is essential for utilizing ChatGPT's capabilities programmatically through the OpenAI API. With an API key, you can access ChatGPT directly from your applications, scripts, or tools, enabling more customized and automated interactions. This allows you to build a wide range of applications, integrating ChatGPT's intelligence to enhance your cybersecurity practices. By setting up an API key, you'll be able to harness the full power of ChatGPT and tailor its features to your specific requirements, making it an indispensable tool for your cybersecurity tasks.

Tip

I highly recommend upgrading to ChatGPT Plus so that you have access to GPT-4. While GPT-3.5 is still very powerful, GPT-4's coding efficiency and accuracy make it more suited to the types of use cases we will be covering in this book and with cybersecurity in general. At the time of writing, there are also other additional features in ChatGPT Plus, such as the availability of plugins and the code interpreter, which will be covered in later chapters.

Creating an API Key and interacting with OpenAI

In this recipe, we will guide you through the process of obtaining an OpenAI API key and introduce you to the OpenAI Playground, where you can experiment with different models and learn more about their capabilities.

Getting ready

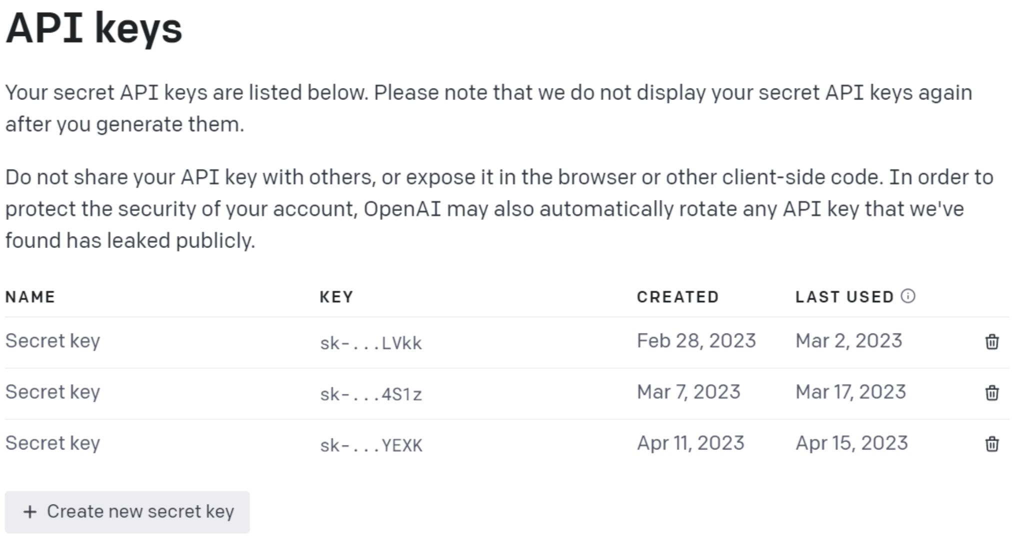
To get an OpenAI API key, you will need to have an active OpenAI account. If you haven't already, complete the *Setting up a ChatGPT account* recipe to set up your ChatGPT account.

How to do it…

Creating an API key and interacting with OpenAI allows you to harness the power of ChatGPT and other OpenAI models for your applications. This means you'll be able to leverage these AI technologies to build powerful tools, automate tasks, and customize your interactions with the models. By the end of this recipe, you will have successfully created an API key for programmatic access to OpenAI models and learned how to experiment with them using the OpenAI Playground.

Now, let's proceed with the steps to create an API key and explore the OpenAI Playground:

1. Log in to your OpenAI account at `https://platform.openai.com`.
2. After logging in, click on your **profile picture/name** in the top-right corner of the screen and select **View API keys** from the drop-down menu:



API keys

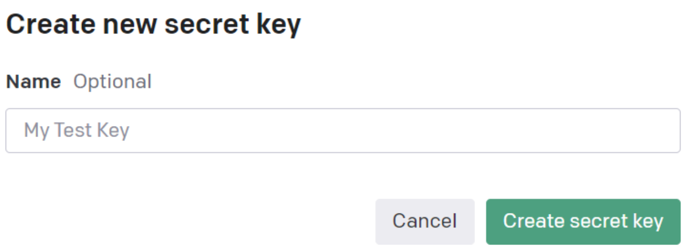
Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically rotate any API key that we've found has leaked publicly.

NAME	KEY	CREATED	LAST USED
Secret key	sk-...LVkk	Feb 28, 2023	Mar 2, 2023
Secret key	sk-...4S1z	Mar 7, 2023	Mar 17, 2023
Secret key	sk-...YEXK	Apr 11, 2023	Apr 15, 2023

Figure 1.2 – The API keys screen

3. Click the **+ Create new secret key** button to generate a new API key.
4. Give your API key a name (optional) and click **Create secret key**:



Create new secret key

Name Optional

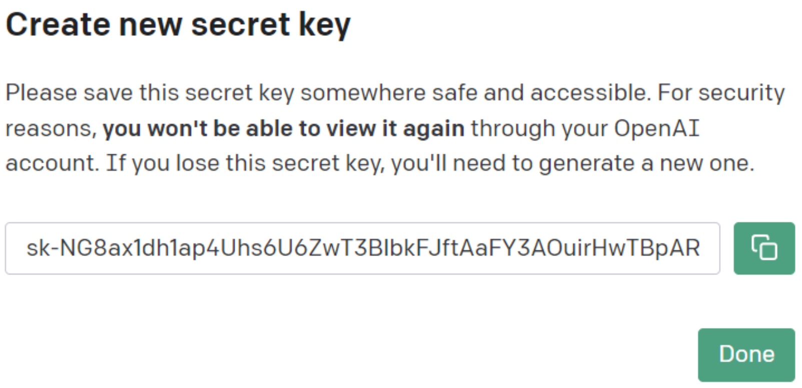
My Test Key

Figure 1.3 – Naming your API key

5. Your new API key will be displayed on the screen. Click the **copy icon**, ⧉, to copy the key to your clipboard:

Tip

Save your API key in a secure location immediately as you will need it later when working with the OpenAI API; you cannot view the key again in its entirety once it has been saved.



Create new secret key

Please save this secret key somewhere safe and accessible. For security reasons, **you won't be able to view it again** through your OpenAI account. If you lose this secret key, you'll need to generate a new one.

sk-NG8ax1dh1ap4Uhs6U6ZwT3BlbkFJftAaFY3AOuirHwTBpAR

Figure 1.4 – Copying your API key

How it works…

By creating an API key, you enable programmatic access to ChatGPT and other OpenAI models through the OpenAI API. This allows you to integrate ChatGPT's capabilities into your applications, scripts, or tools, enabling more customized and automated interactions.

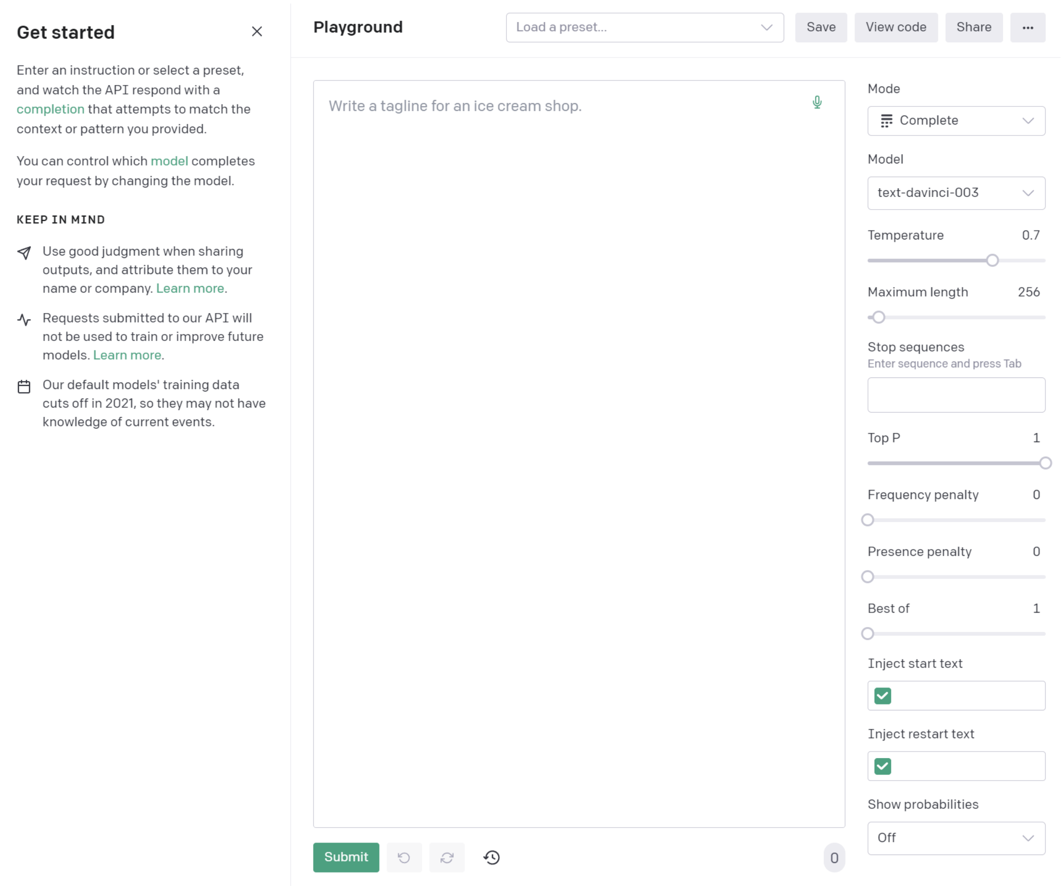
There's more…

The **OpenAI Playground** is an interactive tool that allows you to experiment with different OpenAI models, including ChatGPT, and their various parameters, but without requiring you to write any code. To access and use the Playground, follow these steps:

Important note

Using the Playground requires token credits; you are billed each month for the credits used. For the most part, this cost can be considered very affordable, depending on your perspective. However, excessive use can add up to significant costs if not monitored.

1. Log in to your OpenAI account.
2. Click **Playground** in the top navigation bar:



Get started

Enter an instruction or select a preset, and watch the API respond with a completion that attempts to match the context or pattern you provided.

You can control which model completes your request by changing the model.

KEEP IN MIND

Use good judgment when sharing outputs, and attribute them to your name or company. Learn more.

Requests submitted to our API will not be used to train or improve future models. Learn more.

Our default models' training data cuts off in 2021, so they may not have knowledge of current events.

Playground

Write a tagline for an ice cream shop.

Mode: Complete

Model: text-davinci-003

Temperature 0.7

Maximum length 256

Stop sequences
Enter sequence and press Tab

Top P 1

Frequency penalty 0

Presence penalty 0

Best of 1

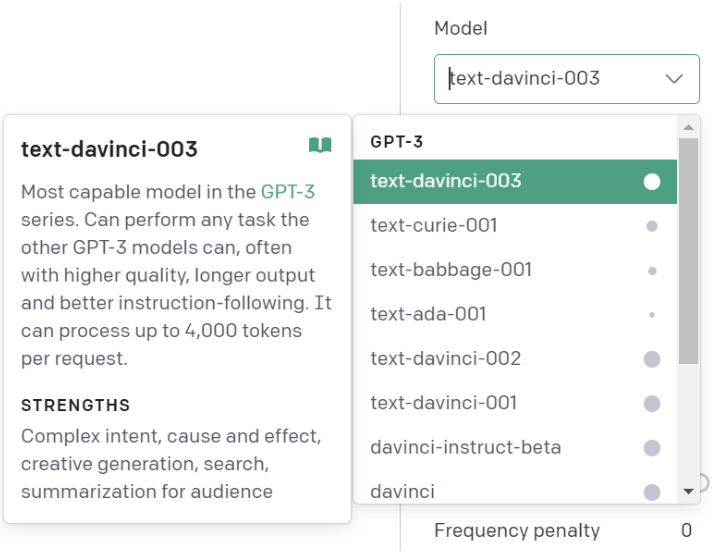
Inject start text ✓

Inject restart text ✓

Show probabilities: Off

Figure 1.5 – The OpenAI Playground

3. In the Playground, you can choose from various models by selecting the model you want to use from the **Model** drop-down menu:



text-davinci-003

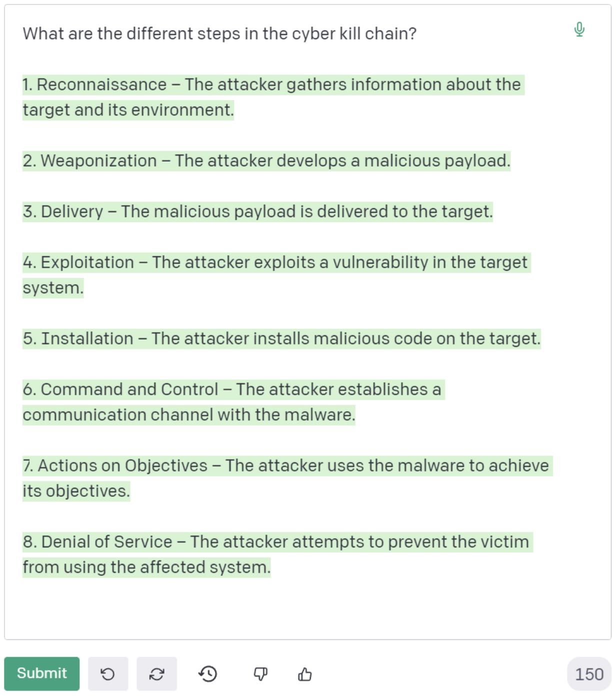
Most capable model in the GPT-3 series. Can perform any task the other GPT-3 models can, often with higher quality, longer output and better instruction-following. It can process up to 4,000 tokens per request.

STRENGTHS

Complex intent, cause and effect, creative generation, search, summarization for audience

Figure 1.6 – Selecting a model

4. Enter your prompt in the textbox provided and click **Submit** to see the model's response:



What are the different steps in the cyber kill chain?

1. Reconnaissance – The attacker gathers information about the target and its environment.

2. Weaponization – The attacker develops a malicious payload.

3. Delivery – The malicious payload is delivered to the target.

4. Exploitation – The attacker exploits a vulnerability in the target system.

5. Installation – The attacker installs malicious code on the target.

6. Command and Control – The attacker establishes a communication channel with the malware.

7. Actions on Objectives – The attacker uses the malware to achieve its objectives.

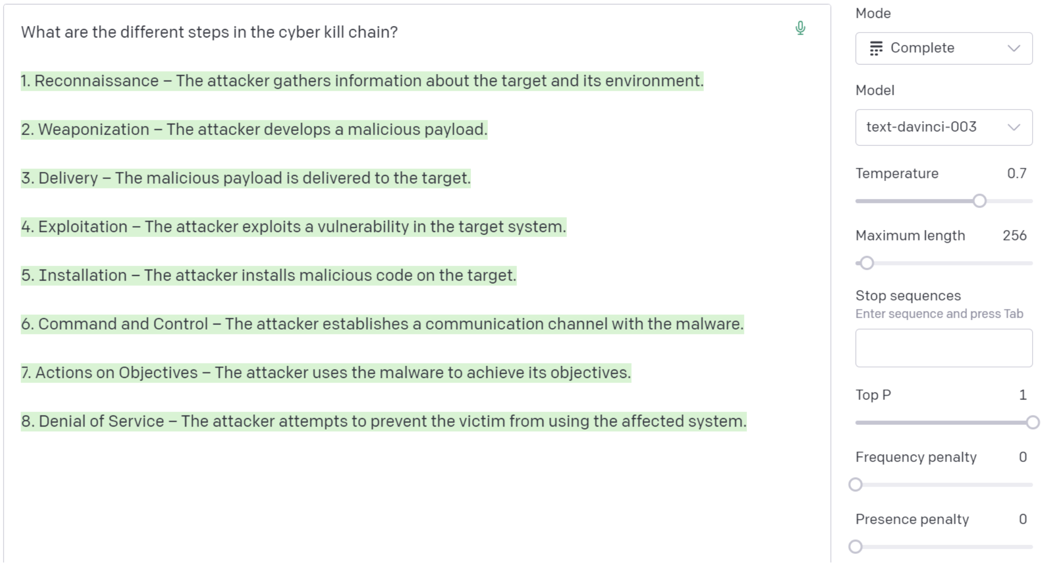
8. Denial of Service – The attacker attempts to prevent the victim from using the affected system.

Figure 1.7 – Entering a prompt and generating a response

Tip

Even though you are not required to enter an API key to interact with the Playground, usage still counts toward your account's token/credit usage.

5. You can also adjust various settings, such as the maximum length, number of generated responses, and more, from the settings panel to the right of the message box:



What are the different steps in the cyber kill chain?

1. Reconnaissance – The attacker gathers information about the target and its environment.

2. Weaponization – The attacker develops a malicious payload.

3. Delivery – The malicious payload is delivered to the target.

4. Exploitation – The attacker exploits a vulnerability in the target system.

5. Installation – The attacker installs malicious code on the target.

6. Command and Control – The attacker establishes a communication channel with the malware.

7. Actions on Objectives – The attacker uses the malware to achieve its objectives.

8. Denial of Service – The attacker attempts to prevent the victim from using the affected system.

Mode: Complete
Model: text-davinci-003
Temperature 0.7
Maximum length 256
Stop sequences
Enter sequence and press Tab
Top P 1
Frequency penalty 0
Presence penalty 0

Figure 1.8 – Adjusting settings in the Playground

Two of the most important parameters are **Temperature** and **Maximum length**:

- The **Temperature** parameter affects the randomness and creativity of the model's responses. A higher temperature (for example, 0.8) will produce more diverse and creative outputs, while a lower temperature (for example, 0.2) will generate more focused and deterministic responses. By adjusting the temperature, you can control the balance between the model's creativity and adherence to the provided context or prompt.
- The **Maximum length** parameter controls the number of tokens (words or word pieces) the model will generate in its response. By setting a higher maximum length, you can obtain longer responses, while a lower maximum length will produce more concise outputs. Adjusting the maximum length can help you tailor the response length to your specific needs or requirements.

Feel free to experiment with these parameters in the OpenAI Playground or when using the API to find the optimal settings for your specific use case or desired output.

The Playground allows you to experiment with different prompt styles, presets, and model settings, helping you better understand how to tailor your prompts and API requests for optimal results:

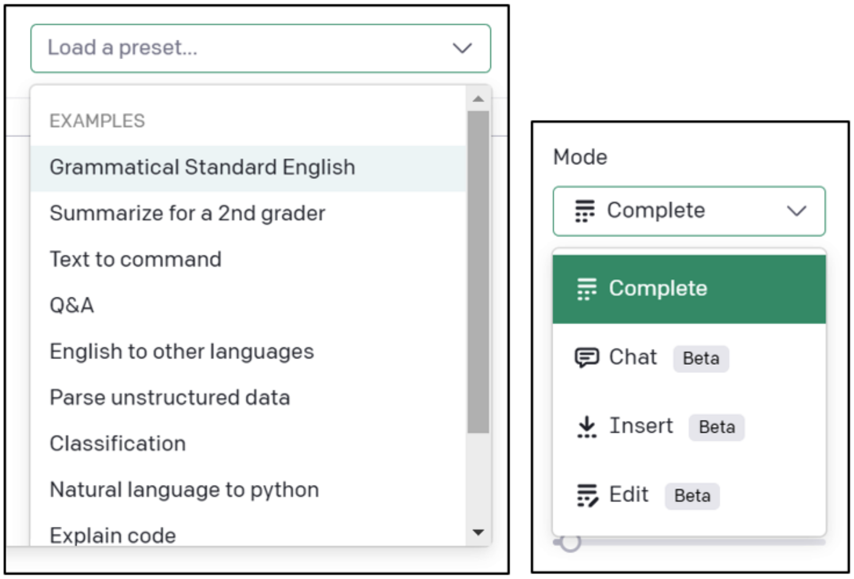


Figure 1.9 – Prompt presets and model modes

Tip
While we will be covering several of the different prompt settings using the API throughout this book, we won't cover them all. You are encouraged to review the *OpenAPI documentation* for more details.

Basic Prompting (Application: Finding Your IP Address)

In this recipe, we will explore the basics of ChatGPT prompting using the ChatGPT interface, which is different from the OpenAI Playground we used in the previous recipe. The advantage of using the ChatGPT interface is that it does not consume account credits and is better suited for generating formatted output, such as writing code or creating tables.

Getting ready

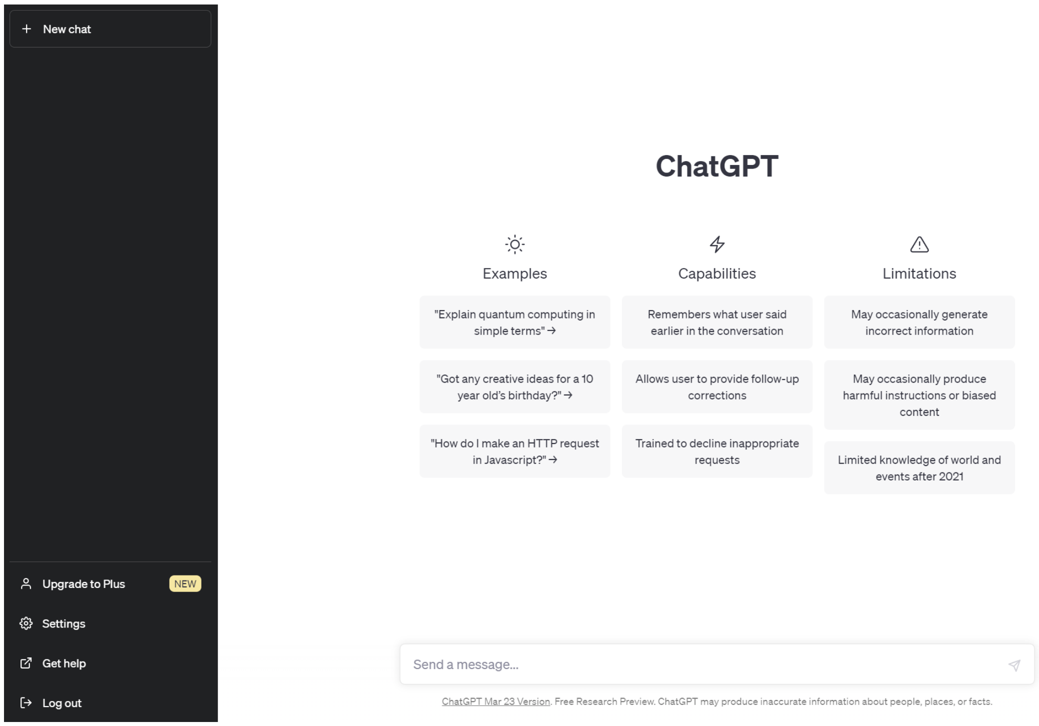
To use the ChatGPT interface, you will need to have an active OpenAI account. If you haven't already, complete the *Setting up a ChatGPT account* recipe to set up your ChatGPT account.

How to do it…

In this recipe, we'll guide you through using the ChatGPT interface to generate a Python script that retrieves a user's public IP address. By following these steps, you'll learn how to interact with ChatGPT in a conversation-like manner and receive context-aware responses, including code snippets.

Now, let's proceed with the steps in this recipe:

1. In your browser, go to `https://chat.openai.com` and click **Log in**.
2. Log in using your OpenAI credentials.
3. Once you are logged in, you will be taken to the ChatGPT interface. The interface is similar to a chat application, with a text box at the bottom where you can enter your prompts:

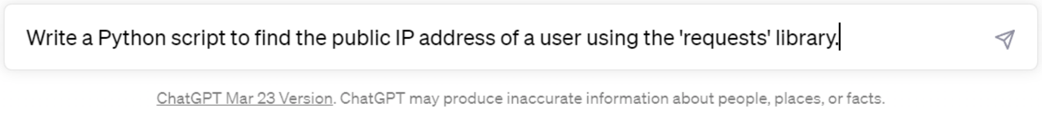


Examples	Capabilities	Limitations
"Explain quantum computing in simple terms" →	Remembers what user said earlier in the conversation	May occasionally generate incorrect information
"Got any creative ideas for a 10 year old's birthday?" →	Allows user to provide follow-up corrections	May occasionally produce harmful instructions or biased content
"How do I make an HTTP request in Javascript?" →	Trained to decline inappropriate requests	Limited knowledge of world and events after 2021

ChatGPT Mar 23 Version. Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts.

Figure 1.10 – The ChatGPT interface

4. ChatGPT uses a conversation-based approach, so you can simply type your prompt as a message and press *Enter* or click the ➢ button to receive a response from the model. For example, you can ask ChatGPT to generate a piece of Python code to find the public IP address of a user:

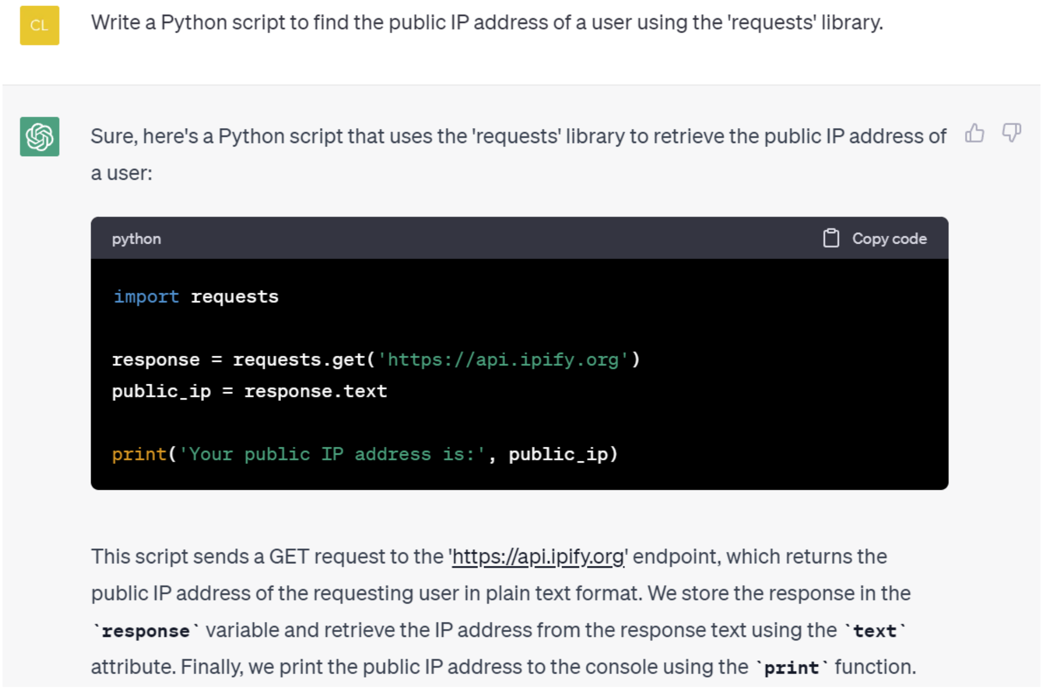


Write a Python script to find the public IP address of a user using the 'requests' library.

ChatGPT Mar 23 Version. ChatGPT may produce inaccurate information about people, places, or facts.

Figure 1.11 – Entering a prompt

ChatGPT will generate a response containing the requested Python code, along with a thorough explanation:



CL Write a Python script to find the public IP address of a user using the 'requests' library.

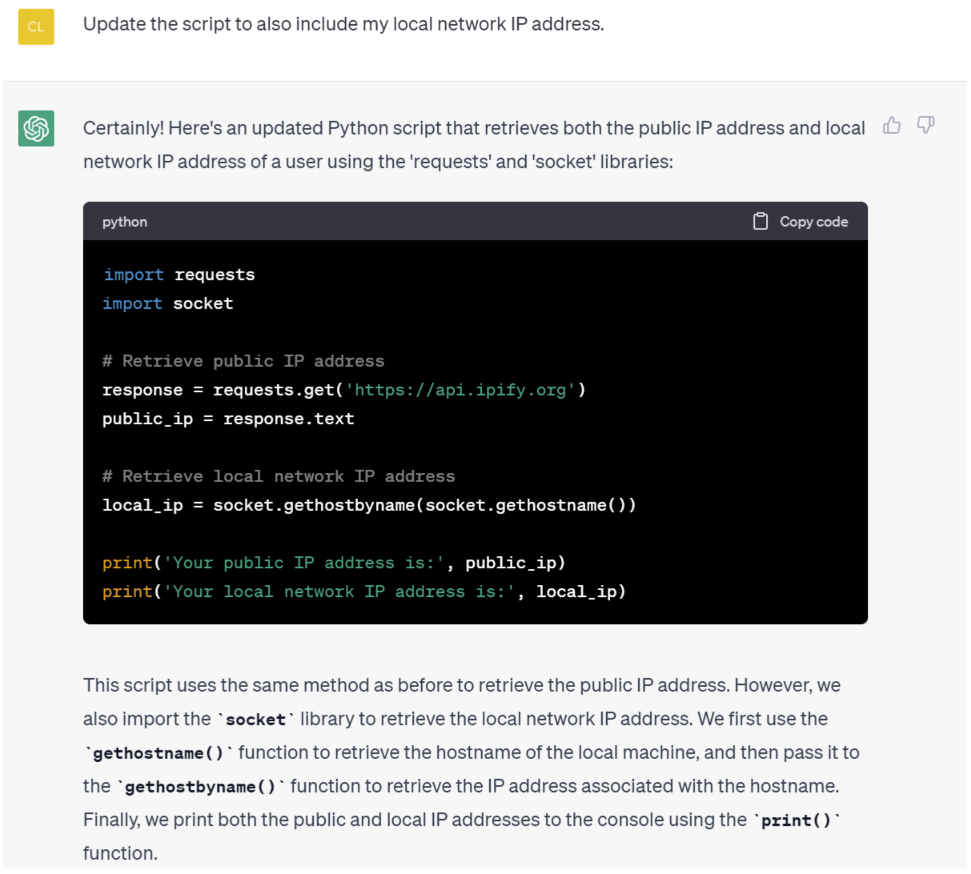
Sure, here's a Python script that uses the 'requests' library to retrieve the public IP address of a user:

```python
import requests

response = requests.get('https://api.ipify.org')
public_ip = response.text

print('Your public IP address is:', public_ip)
```

This script sends a GET request to the 'https://api.ipify.org' endpoint, which returns the public IP address of the requesting user in plain text format. We store the response in the `response` variable and retrieve the IP address from the response text using the `text` attribute. Finally, we print the public IP address to the console using the `print` function.

Figure 1.12 – ChatGPT response with code

5. Continue the conversation by asking follow-up questions or providing additional information, and ChatGPT will respond accordingly:



Update the script to also include my local network IP address.

Certainly! Here's an updated Python script that retrieves both the public IP address and local network IP address of a user using the 'requests' and 'socket' libraries:

```python
import requests
import socket

# Retrieve public IP address
response = requests.get('https://api.ipify.org')
public_ip = response.text

# Retrieve local network IP address
local_ip = socket.gethostbyname(socket.gethostname())

print('Your public IP address is:', public_ip)
print('Your local network IP address is:', local_ip)
```

This script uses the same method as before to retrieve the public IP address. However, we also import the `socket` library to retrieve the local network IP address. We first use the `gethostname()` function to retrieve the hostname of the local machine, and then pass it to the `gethostbyname()` function to retrieve the IP address associated with the hostname. Finally, we print both the public and local IP addresses to the console using the `print()` function.

Figure 1.13 – ChatGPT contextual follow-up response

6. Run the ChatGPT-generated code by clicking on **Copy code**, paste it into your code editor of choice (I use *Visual Studio Code*), save it as a `.py` Python script, and run it from a terminal:

```
PS D:\GPT\ChatGPT for Cybersecurity Cookbook> python .\my_ip.py
Your public IP address is: [redacted]
Your local network IP address is: 192.168.1.105
```

Figure 1.14 – Running the ChatGPT-generated script

How it works…

By using the ChatGPT interface to enter prompts, you can generate context-aware responses and content that continues throughout an entire conversation, similar to a chatbot. The conversation-based approach allows for more natural interactions and the ability to ask follow-up questions or provide additional context. The responses can even include complex formatting such as code snippets or tables (more on tables later).

There's more…

As you become more familiar with ChatGPT, you can experiment with different prompt styles, instructions, and contexts to obtain the desired output for your cybersecurity tasks. You can also compare the results that are generated through the ChatGPT interface and the OpenAI Playground to determine which approach best fits your needs.

Tip

You can further refine the generated output by providing very clear and specific instructions or using roles. It also helps to divide complex prompts into several smaller prompts, giving ChatGPT one instruction per prompt, building on the previous prompts as you go.

In the upcoming recipes, we will delve into more advanced prompting techniques that utilize these techniques to help you get the most accurate and detailed responses from ChatGPT.

As you interact with ChatGPT, your conversation history is automatically saved in the left panel of the ChatGPT interface. This feature allows you to easily access and review your previous prompts and responses.

By leveraging the conversation history feature, you can keep track of your interactions with ChatGPT and quickly reference previous responses for your cybersecurity tasks or other projects:

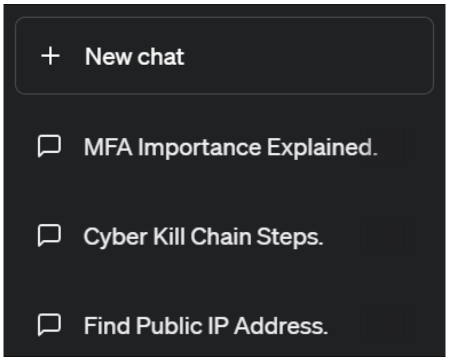


Figure 1.15 – Conversation history in the ChatGPT interface

To view a saved conversation, simply click on the desired conversation in the left panel. You can also create new conversations by clicking on the **+ New chat** button located at the top of the conversation list. This enables you to separate and organize your prompts and responses based on specific tasks or topics.

Note of caution

Keep in mind that when you start a new conversation, the model loses the context of the previous conversation. If you want to reference any information from a previous conversation, you will need to include that context in your new prompt.

Applying ChatGPT Roles (Application : AI CISO)

In this recipe, we will demonstrate how you can use roles in your prompts to improve the accuracy and detail of ChatGPT's responses. Assigning roles to ChatGPT helps it generate more context-aware and relevant content, particularly when you need expert-level insights or recommendations.

Getting ready

Ensure you have access to the ChatGPT interface by logging in to your OpenAI account.

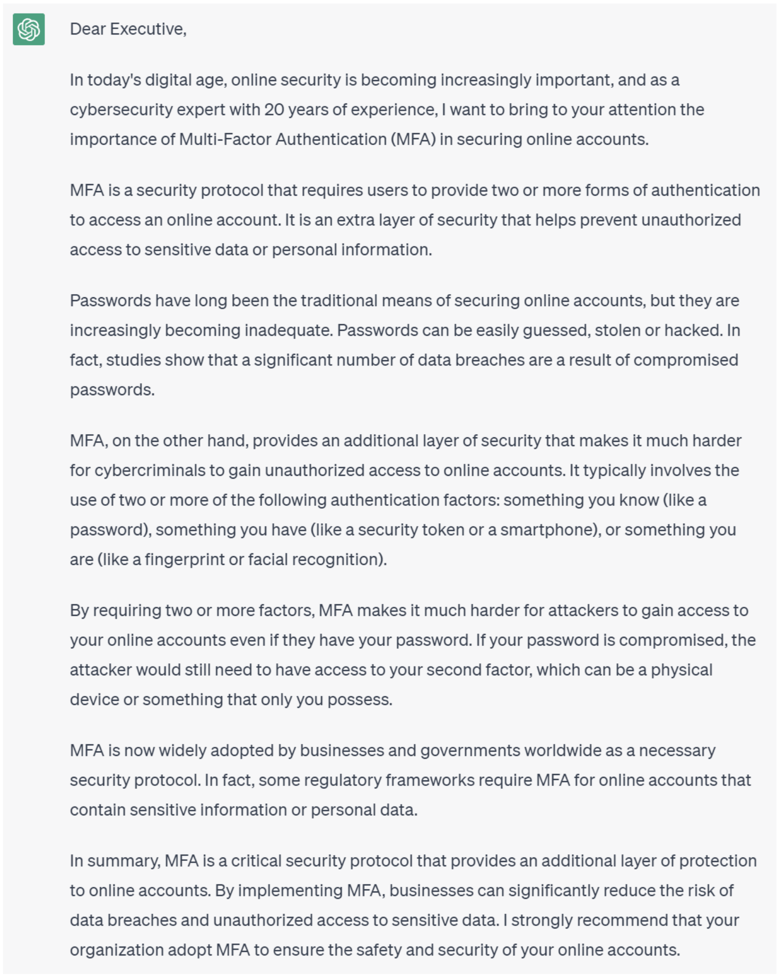
How to do it...

By assigning roles, you'll be able to obtain expert-level insights and recommendations from the model. Let's dive into the steps for this recipe:

1. To assign a role to ChatGPT, start your prompt by describing the role you want the model to assume. For example, you could use the following prompt:

   ```
   You are a cybersecurity expert with 20 years of experience.
   Explain the importance of multi-factor authentication (MFA) in
   securing online accounts, to an executive audience.
   ```

2. ChatGPT will generate a response that aligns with the assigned role, providing a detailed explanation of the topic based on the expertise and perspective of a cybersecurity expert:


Dear Executive,

In today's digital age, online security is becoming increasingly important, and as a cybersecurity expert with 20 years of experience, I want to bring to your attention the importance of Multi-Factor Authentication (MFA) in securing online accounts.

MFA is a security protocol that requires users to provide two or more forms of authentication to access an online account. It is an extra layer of security that helps prevent unauthorized access to sensitive data or personal information.

Passwords have long been the traditional means of securing online accounts, but they are increasingly becoming inadequate. Passwords can be easily guessed, stolen or hacked. In fact, studies show that a significant number of data breaches are a result of compromised passwords.

MFA, on the other hand, provides an additional layer of security that makes it much harder for cybercriminals to gain unauthorized access to online accounts. It typically involves the use of two or more of the following authentication factors: something you know (like a password), something you have (like a security token or a smartphone), or something you are (like a fingerprint or facial recognition).

By requiring two or more factors, MFA makes it much harder for attackers to gain access to your online accounts even if they have your password. If your password is compromised, the attacker would still need to have access to your second factor, which can be a physical device or something that only you possess.

MFA is now widely adopted by businesses and governments worldwide as a necessary security protocol. In fact, some regulatory frameworks require MFA for online accounts that contain sensitive information or personal data.

In summary, MFA is a critical security protocol that provides an additional layer of protection to online accounts. By implementing MFA, businesses can significantly reduce the risk of data breaches and unauthorized access to sensitive data. I strongly recommend that your organization adopt MFA to ensure the safety and security of your online accounts.

Figure 1.16 – ChatGPT response with role-based expertise

3. Experiment with assigning different roles for different scenarios, such as the following:

```
You are a CISO with 30 years of experience. What are the top
cybersecurity risks businesses should be aware of?
```

4. Alternatively, you can use the following:

```
You are an ethical hacker. Explain how a penetration test can
help improve an organization's security posture.
```

Note of caution

Keep in mind that ChatGPT's knowledge is based on the data it was trained on, which has a cutoff date of September 2021. As a result, the model may not be aware of the latest developments, trends, or technologies in the cybersecurity field that emerged after its training data cutoff. Always verify the information generated by ChatGPT with up-to-date sources and take its training limitations into account when interpreting its responses. We will discuss techniques on how to get around this limitation later in this book.

How it works…

When you assign a role to ChatGPT, you provide a specific context or persona for the model to work with. This helps the model generate responses that are tailored to the given role, resulting in more accurate, relevant, and detailed content. The model will generate content that aligns with the expertise and perspective of the assigned role, offering better insights, opinions, or recommendations.

There's more…

As you become more comfortable using roles in your prompts, you can experiment with different combinations of roles and scenarios to obtain the desired output for your cybersecurity tasks. For example, you can create a dialogue between two roles by alternating prompts for each role:

1. **Role 1**:

   ```
   You are a network administrator. What measures do you take to
   secure your organization's network?
   ```

2. **Role 2**:

   ```
   You are a cybersecurity consultant. What additional
   recommendations do you have for the network administrator to
   further enhance network security?
   ```

By using roles creatively and experimenting with different combinations, you can leverage ChatGPT's expertise and obtain more accurate and detailed responses for a wide range of cybersecurity topics and situations.

We will experiment with automating role conversations in later chapters.

Enhancing Output with Templates (Application: Threat Report)

In this recipe, we will explore how to use output templates to guide ChatGPT's responses, making them more consistent, well-structured, and suitable for reports or other formal documents. By providing a specific format for the output, you can ensure that the generated content meets your requirements and is easier to integrate into your cybersecurity projects.

Getting ready

Ensure you have access to the ChatGPT interface by logging in to your OpenAI account.

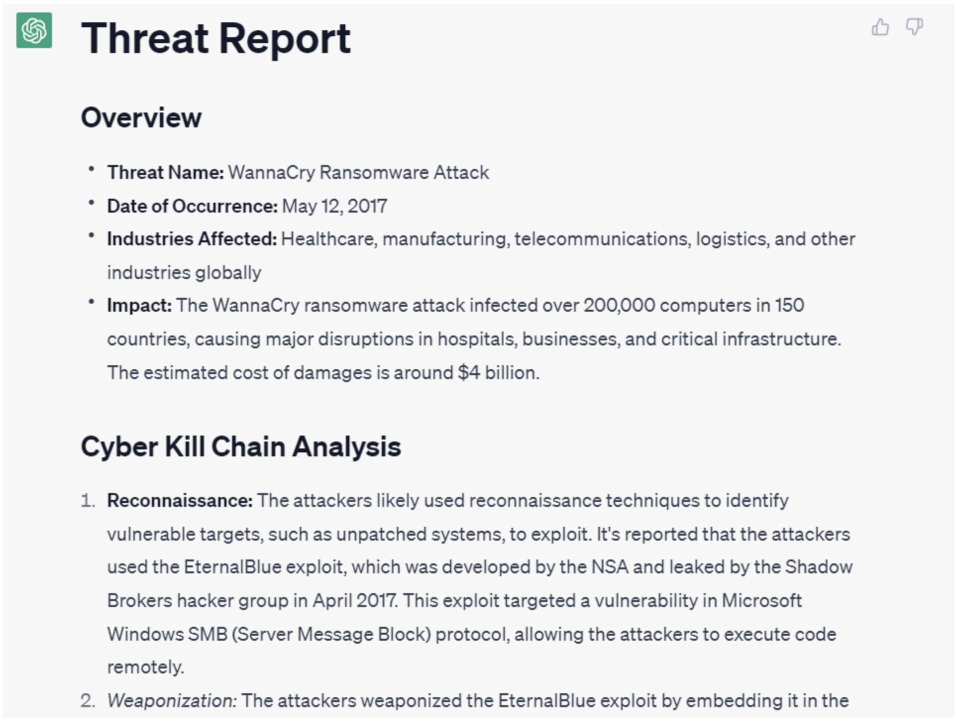
How to do it…

To get started, follow these steps:

1. When crafting your prompt, you can specify the output of several different formatting options, such as headings, font weight, lists, and more. The following prompt demonstrates how to create output with headings, font weights, and list types:

```
Create an analysis report of the WannaCry Ransomware Attack as
it relates to the cyber kill chain, using the following format:

# Threat Report

## Overview
- **Threat Name:**
- **Date of Occurrence:**
- **Industries Affected:**
- **Impact:**

## Cyber Kill Chain Analysis

1. **Kill chain step 1:**
2. **Kill chain step 2:**
3. …

## Mitigation Recommendations

- *Mitigation recommendation 1*
- *Mitigaiton recommendation 2*
…
```

2. ChatGPT will generate a response that follows the specified template, providing a well-structured and consistent output:



Threat Report

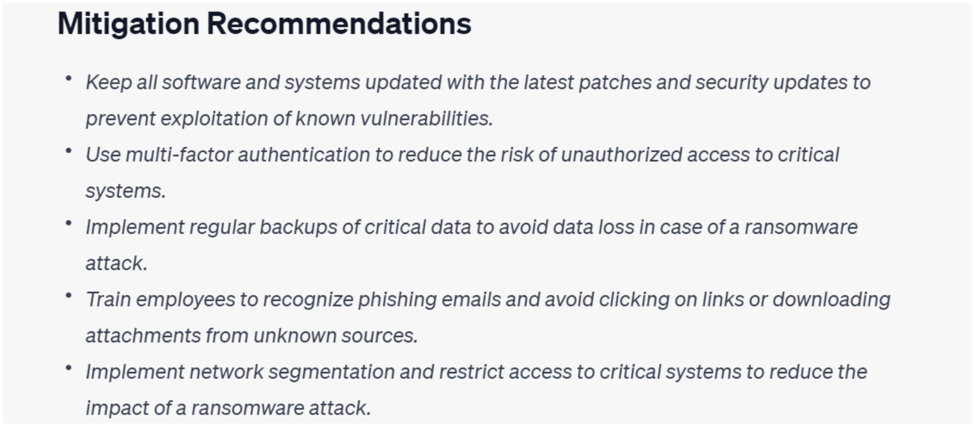
Overview

- **Threat Name:** WannaCry Ransomware Attack
- **Date of Occurrence:** May 12, 2017
- **Industries Affected:** Healthcare, manufacturing, telecommunications, logistics, and other industries globally
- **Impact:** The WannaCry ransomware attack infected over 200,000 computers in 150 countries, causing major disruptions in hospitals, businesses, and critical infrastructure. The estimated cost of damages is around $4 billion.

Cyber Kill Chain Analysis

1. **Reconnaissance:** The attackers likely used reconnaissance techniques to identify vulnerable targets, such as unpatched systems, to exploit. It's reported that the attackers used the EternalBlue exploit, which was developed by the NSA and leaked by the Shadow Brokers hacker group in April 2017. This exploit targeted a vulnerability in Microsoft Windows SMB (Server Message Block) protocol, allowing the attackers to execute code remotely.
2. *Weaponization:* The attackers weaponized the EternalBlue exploit by embedding it in the

Figure 1.17 – ChatGPT response with formatting (headings, bold font, and lists)



Mitigation Recommendations

- *Keep all software and systems updated with the latest patches and security updates to prevent exploitation of known vulnerabilities.*
- *Use multi-factor authentication to reduce the risk of unauthorized access to critical systems.*
- *Implement regular backups of critical data to avoid data loss in case of a ransomware attack.*
- *Train employees to recognize phishing emails and avoid clicking on links or downloading attachments from unknown sources.*
- *Implement network segmentation and restrict access to critical systems to reduce the impact of a ransomware attack.*

Figure 1.18 – ChatGPT response with formatting (heading, lists, and italicized text)

3. This formatted text is now more structured and can be easily transferred to other documents through copying and pasting while retaining its formatting.

How it works...

By providing a clear template for the output in your prompt, you guide ChatGPT to generate responses that adhere to the specified structure and formatting. This helps ensure that the generated content is consistent, well organized, and suitable for use in reports, presentations, or other formal documents. The model will focus on generating content that matches the output template formatting and structure you've provided while still delivering the information you requested.

The following conventions are used when formatting ChatGPT output:

1. To create a main **heading**, use a single pound sign (`#`), followed by a space and the text of the heading. In this case, the main heading is *Threat Report*.
2. To create a **subheading**, use two pound signs (`##`), followed by a space and the text of the subheading. In this case, the subheadings are *Overview*, *Cyber Kill Chain Analysis*, and *Mitigation Recommendations*. You can continue to create additional subheading levels by increasing the number of pound signs.
3. To create **bullet points**, use a hyphen (`-`) or asterisk (`*`), followed by a space and the text of the bullet point. In this case, the bullet points are used in the *Overview* section to indicate the threat's name, date of occurrence, industries affected, and impact.
4. To create **bold text**, use two asterisks (`**`) or underscores (__) to surround the text you want to bold. In this case, each of the bullets and numbered list keywords were bolded.
5. To **italicize text**, use a pair of asterisks (`*`) or underscores (_) to surround the text you want to italicize. In this case, the second kill chain step is italicized using a pair of underscores. Here, italicized text is used for the *mitigations recommendations bullets*.
6. To create a **numbered list**, use a number followed by a period and a space, followed by the text of the list item. In this case, the *Cyber Kill Chain Analysis section* is a numbered list.

There's more...

Combining templates with other techniques, such as roles, can further enhance the quality and relevance of the generated content. By applying both templates and roles, you can create output that is not only well-structured and consistent but also tailored to specific expert perspectives.

As you become more comfortable using templates in your prompts, you can experiment with different formats, structures, and scenarios to obtain the desired output for your cybersecurity tasks. For example, in addition to text formatting, you can also use tables to organize the generated content even further, which is what we will cover in the next recipe.

Formatting Output as a Table (Application: Security Controls Table)

In this recipe, we will demonstrate how to create prompts that guide ChatGPT to generate output in table format. Tables can be an effective way to organize and present information in a structured and easy-to-read manner. In this example, we will create a security controls comparison table.

Getting ready

Ensure you have access to the ChatGPT interface by logging into your OpenAI account.

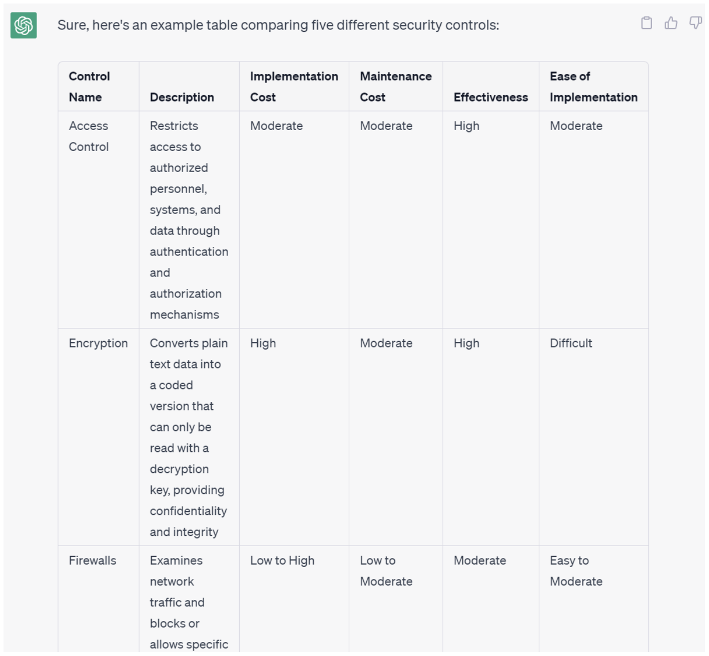
How to do it…

This example will demonstrate how to create a security controls comparison table. Let's dive into the steps to achieve this:

1. Craft your prompt by specifying the table format and the information you want to include. For this example, we will generate a table comparing different security controls:

```
Create a table comparing five different security controls.
The table should have the following columns: Control
Name, Description, Implementation Cost, Maintenance Cost,
Effectiveness, and Ease of Implementation.
```

2. ChatGPT will generate a response containing a table with the specified columns, populated with relevant information:

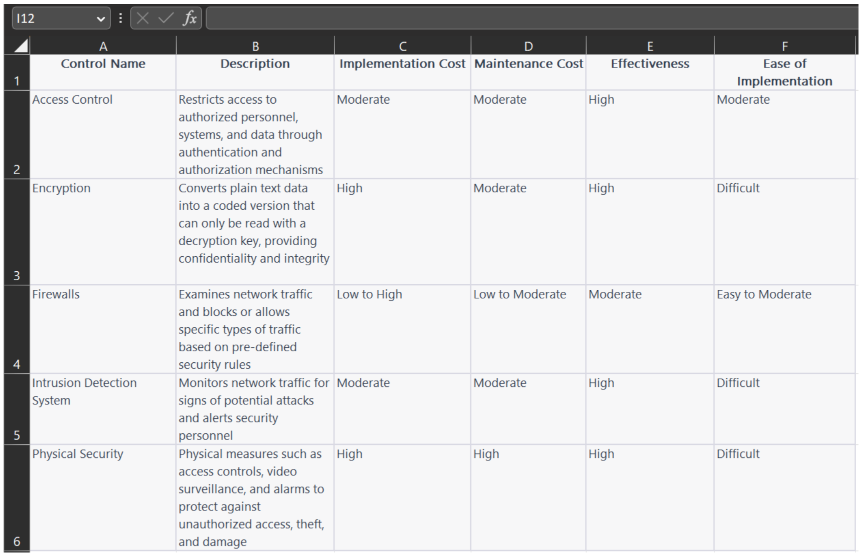


Sure, here's an example table comparing five different security controls:

Control Name	Description	Implementation Cost	Maintenance Cost	Effectiveness	Ease of Implementation
Access Control	Restricts access to authorized personnel, systems, and data through authentication and authorization mechanisms	Moderate	Moderate	High	Moderate
Encryption	Converts plain text data into a coded version that can only be read with a decryption key, providing confidentiality and integrity	High	Moderate	High	Difficult
Firewalls	Examines network traffic and blocks or allows specific	Low to High	Low to Moderate	Moderate	Easy to Moderate

Figure 1.19 – Snippet of a ChatGPT response with a table

3. You can now easily copy and paste the generated table directly into a document or spreadsheet, where it can be further formatted and refined:



Control Name	Description	Implementation Cost	Maintenance Cost	Effectiveness	Ease of Implementation
Access Control	Restricts access to authorized personnel, systems, and data through authentication and authorization mechanisms	Moderate	Moderate	High	Moderate
Encryption	Converts plain text data into a coded version that can only be read with a decryption key, providing confidentiality and integrity	High	Moderate	High	Difficult
Firewalls	Examines network traffic and blocks or allows specific types of traffic based on pre-defined security rules	Low to High	Low to Moderate	Moderate	Easy to Moderate
Intrusion Detection System	Monitors network traffic for signs of potential attacks and alerts security personnel	Moderate	Moderate	High	Difficult
Physical Security	Physical measures such as access controls, video surveillance, and alarms to protect against unauthorized access, theft, and damage	High	High	High	Difficult

Figure 1.20 – ChatGPT response copied/pasted directly into a spreadsheet

How it works…

By specifying the table format and required information in your prompt, you guide ChatGPT to generate content in a structured, tabular manner. The model will focus on generating content that matches the specified format and populating the table with the requested information. The ChatGPT interface automatically understands how to provide table formatting using markdown language, which is then interpreted by the browser.

In this example, we asked ChatGPT to create a table comparing five different security controls with columns for **Control Name**, **Description**, **Implementation Cost**, **Maintenance Cost**, **Effectiveness**, and **Ease of Implementation**. The resulting table provides an organized and easy-to-understand overview of the different security controls.

There's more…

As you become more comfortable using tables in your prompts, you can experiment with different formats, structures, and scenarios to obtain the desired output for your cybersecurity tasks. You can also combine tables with other techniques, such as roles and templates, to further enhance the quality and relevance of the generated content.

By using tables creatively and experimenting with different combinations, you can leverage ChatGPT's capabilities to generate structured and organized content for various cybersecurity topics and situations.

Setting the OpenAI API Key as an Environment Variable

In this recipe, we will show you how to set up your OpenAI API key as an environment variable. This is an essential step as it allows you to use the API key in your Python code without hardcoding it, which is a best practice for security purposes.

Getting ready

Ensure that you have already obtained your OpenAI API key by signing up for an account and accessing the API key section, as outlined in the *Creating an API key and interacting with OpenAI* recipe.

How to do it…

This example will demonstrate how to set up your OpenAI API key as an environment variable for secure access in your Python code. Let's dive into the steps to achieve this.

1. Set up the API key as an environment variable on your operating system.

For Windows

I. Open the Start menu, search for `Environment Variables`, and click **Edit the system environment variables**.

II. In the **System Properties** window, click the **Environment Variables** button.

III. In the **Environment Variables** window, click **New** under **User variables** or **System variables** (depending on your preference).

IV. Enter `OPENAI_API_KEY` as the variable's name and paste your API key as the variable value. Click **OK** to save the new environment variable.

For macOS/Linux

I. Open a Terminal window.

II. Add the API key to your shell configuration file (such as `.bashrc`, `.zshrc`, or `.profile`) by running the following command (replace `your_api_key` with your actual API key):

```
echo 'export OPENAI_API_KEY="your_api_key"' >> ~/.bashrc
```

Tip

If you are using a different shell configuration file, replace `~/.bashrc` with the appropriate file (for example, `.,  ~/.zshrc` or `~/.profile`).

III. Restart Terminal or run `source ~/.bashrc` (or the appropriate configuration file) to apply the changes.

2. Access the API key in your Python code using the `os` module:

```
import os

# Access the OpenAI API key from the environment variable
api_key = os.environ["OPENAI_API_KEY"]
```

Important note

There are many different versions of Linux and Unix-based systems, and the exact syntax for setting environment variables might differ slightly from what is presented here. However, the general approach should be similar. If you encounter issues, consult the documentation specific to your system for guidance on setting environment variables.

How it works…

By setting up the OpenAI API key as an environment variable, you make it available for use in your Python code without hardcoding the key, which is a security best practice. In the Python code, you use the `os` module to access the API key from the environment variable you created earlier.

Using environment variables is a common practice when working with sensitive data, such as API keys or other credentials. This approach allows you to separate your code from your sensitive data and makes it easier to manage your credentials as you only need to update them in one place (the environment variables). Additionally, it helps prevent accidental exposure of sensitive information when you're sharing code with others or publishing it in public repositories.

There's more…

In some cases, you may want to use a Python package such as `python-dotenv` to manage your environment variables. This package allows you to store your environment variables in a `.env` file, which you can load in your Python code. The advantage of this approach is that you can keep all your project-specific environment variables in a single file, making it easier to manage and share your project settings. Keep in mind, though, that you should never commit the `.env` file to a public repository; always include it in your `.gitignore` file or similar version control ignore configuration.

Sending API Requests and Handling Responses with Python

In this recipe, we will explore how to send requests to the OpenAI GPT API and handle the responses using Python. We'll walk through the process of constructing API requests, sending them, and processing the responses using the `openai` module.

Getting ready

1. Ensure you have Python installed on your system.
2. Install the OpenAI Python module by running the following command in your Terminal or command prompt:

```
pip install openai
```

How to do it…

The importance of using the API lies in its ability to communicate with and get valuable insights from ChatGPT in real time. By sending API requests and handling responses, you can harness the power of GPT to answer questions, generate content, or solve problems in a dynamic and customizable way.

In the following steps, we'll demonstrate how to construct API requests, send them, and process the responses, enabling you to effectively integrate ChatGPT into your projects or applications:

1. Start by importing the required modules:

```
import openai
from openai import OpenAI
import os
```

2. Set up your API key by retrieving it from an environment variable, as we did in the *Setting the OpenAI API key as an Environment Variable* recipe:

```
openai.api_key = os.getenv("OPENAI_API_KEY")
```

3. Define a function to send a prompt to the OpenAI API and receive a response:

```
client = OpenAI()

def get_chat_gpt_response(prompt):
  response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": prompt}],
    max_tokens=2048,
    temperature=0.7
  )
  return response.choices[0].message.content.strip()
```

4. Call the function with a prompt to send a request and receive a response:

```
prompt = "Explain the difference between symmetric and
asymmetric encryption."
response_text = get_chat_gpt_response(prompt)
print(response_text)
```

How it works…

1. First, we import the required modules. The `openai` module is the OpenAI API library, and the `os` module helps us retrieve the API key from an environment variable.
2. We set up the API key by retrieving it from an environment variable using the `os` module.
3. Next, we define a function called `get_chat_gpt_response()` that takes a single argument: the prompt. This function sends a request to the OpenAI API using the `openai.Completion.create()` method. This method has several parameters:
   - `engine`: Here, we specify the engine (in this case, `chat-3.5-turbo`).
   - `prompt`: The input text for the model to generate a response.

- `max_tokens`: The maximum number of tokens in the generated response. A token can be as short as one character or as long as one word.
- `n`: The number of generated responses you want to receive from the model. In this case, we've set it to `1` to receive a single response.
- `stop`: A sequence of tokens that, if encountered by the model, will stop the generation process. This can be useful for limiting the response's length or stopping at specific points, such as the end of a sentence or paragraph.
- `temperature`: A value that controls the randomness of the generated response. A higher temperature (for example, 1.0) will result in more random responses, while a lower temperature (for example, 0.1) will make the responses more focused and deterministic.

4. Finally, we call the `get_chat_gpt_response()` function with a prompt, send the request to the OpenAI API, and receive the response. The function returns the response text, which is then printed to the console. The function returns the response text, which is then printed to the console. The `return response.choices[0].message.content.strip()` line of code retrieves the generated response text by accessing the first choice (`index 0`) in the list of choices.
5. `response.choices` is a list of generated responses from the model. In our case, since we set `n=1`, there is only one response in the list. The `.text` attribute retrieves the actual text of the response, and the `.strip()` method removes any leading or trailing whitespace.
6. For example, a non-formatted response from the OpenAI API may look like this:

```
{
  'id': 'example_id',
  'object': 'text.completion',
  'created': 1234567890,
  'model': 'chat-3.5-turbo',
  'usage': {'prompt_tokens': 12, 'completion_tokens': 89,
'total_tokens': 101},
  'choices': [
    {
      'text': ' Symmetric encryption uses the same key for
both encryption and decryption, while asymmetric encryption
uses different keys for encryption and decryption, typically a
public key for encryption and a private key for decryption. This
difference in key usage leads to different security properties
and use cases for each type of encryption.',
      'index': 0,
      'logprobs': None,
      'finish_reason': 'stop'
    }
  ]
}
```

In this example, we access the response text using `response.choices[0].text.strip()`, which returns the following text:

```
Symmetric encryption uses the same key for both encryption and
decryption, while asymmetric encryption uses different keys for
encryption and decryption, typically a public key for encryption
and a private key for decryption. This difference in key usage
leads to different security properties and use cases for each
type of encryption.
```

There's more…

You can further customize the API request by modifying the parameters in the `openai.Completion.create()` method. For example, you can adjust the temperature to get more creative or focused responses, change the `max_tokens` value to limit or expand the length of the generated content, or use the `stop` parameter to define specific stopping points for the response generation.

Additionally, you can experiment with the `n` parameter to generate multiple responses and compare their quality or variety. Keep in mind that generating multiple responses will consume more tokens and may affect the cost and execution time of the API request.

It's essential to understand and fine-tune these parameters to get the desired output from ChatGPT since different tasks or scenarios may require different levels of creativity, response length, or stopping conditions. As you become more familiar with the OpenAI API, you'll be able to leverage these parameters effectively to tailor the generated content to your specific cybersecurity tasks and requirements.

Using Files for Prompts and API Key Access

In this recipe, you will learn how to use external text files to store and retrieve prompts for interacting with the OpenAI API through Python. This method allows for better organization and easier maintenance as you can quickly update the prompt without modifying the main script. We will also introduce a new method of accessing the OpenAI API key – that is, using files – making the process of changing the API key much more flexible.

Getting ready

Ensure you have access to the OpenAI API and have set up your API key according to the *Creating an API key and interacting with OpenAI* and *Setting the OpenAI API key as an Environment Variable* recipes.

How to do it…

This recipe demonstrates a practical approach to managing prompts and API keys, making it easier to update and maintain your code. By using external text files, you can efficiently organize your project and collaborate with others. Let's walk through the steps to implement this method:

1. Create a new text file and save it as `prompt.txt`. Write your desired prompt inside this file and save it.
2. Modify your Python script so that it includes a function to read the contents of a text file:

```
def open_file(filepath):
    with open(filepath, 'r', encoding='UTF-8') as infile:
        return infile.read()
```

3. Using the script from the *Sending API Requests and Handling Responses with Python* recipe, replace the hardcoded prompt with a call to the `open_file` function, passing the path to the `prompt.txt` file as an argument:

```
prompt = open_file("prompt.txt")
```

4. Create a file called `prompt.txt` and enter the following prompt text (the same prompt as in the *Sending API Requests and Handling Responses with Python* recipe):

```
Explain the difference between symmetric and asymmetric
encryption.
```

5. Set up your API key using a file instead of environment variables:

```
openai.api_key = open_file('openai-key.txt')
```

Important note

It's important to place this line of code after the `open_file` function; otherwise, Python will throw an error for calling a function that has not been declared yet.

6. Create a file called `openai-key.txt` and paste your **OpenAI API key** into the file with nothing else.
7. Use the prompt variable in your API call as you normally would.

   Here is an example of how the modified script from the *Sending API Requests and Handling Responses with Python* recipe would look:

```
import openai
from openai import OpenAI

def open_file(filepath):
    with open(filepath, 'r', encoding='UTF-8') as infile:
        return infile.read()

client = OpenAI()

def get_chat_gpt_response(prompt):
  response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": prompt}],
    max_tokens=2048,
    temperature=0.7
  )
  return response.choices[0].message.content.strip()

openai.api_key = open_file('openai-key.txt')

prompt = open_file("prompt.txt")
response_text = get_chat_gpt_response(prompt)
print(response_text)
```

How it works…

The `open_file()` function takes a file path as an argument and opens the file using the `with open` statement. It reads the file's content and returns it as a string. This string is then used as the prompt for the API call. A second `open_file()` function call is used to access a text file containing the OpenAI API key instead of accessing the API key using environment variables.

By using an external text file for the prompt and to access the API key, you can easily update or change both without needing to modify the main script or environment variables. This can be particularly helpful when you're working with multiple prompts or collaborating with others.

Note of caution

Using this technique to access your API key does come with a certain level of risk. A text file is easier to discover and access than an environment variable, so be sure to take the necessary security precautions. It is also important to remember to remove your API key from the `openapi-key.txt` file before you share your script with others, to prevent unintended and/or unauthorized charges to your OpenAI account.

There's more…

You can also use this method to store other parameters or configurations that you may want to change frequently or share with others. This could include API keys, model parameters, or any other settings relevant to your use case.

Using Prompt Variables (Application: Manual Page Generator)

In this recipe, we'll create a Linux-style manual page generator that will accept user input in the form of a tool's name, and our script will generate the manual page output, similar to entering the `man` command in Linux Terminal. In doing so, we will learn how to use variables in a text file to create a standard prompt *template* that can be easily modified by changing certain aspects of it. This approach is particularly useful when you want to use user input or other dynamic content as part of the prompt while maintaining a consistent structure.

Getting ready

Ensure you have access to the ChatGPT API by logging in to your OpenAI account and have Python and the `openai` module installed.

How to do it...

Using a text file that contains the prompt and placeholder variables, we can create a Python script that will replace the placeholder with user input. In this example, we will use this technique to create a Linux-style manual page generator. Here are the steps:

1. Create a Python script and import the necessary modules:

```
from openai import OpenAI
```

2. Define a function to open and read a file:

```
def open_file(filepath):
    with open(filepath, 'r', encoding='UTF-8') as infile:
        return infile.read()
```

3. Set up your API key:

```
openai.api_key = open_file('openai-key.txt')
```

4. Create the `openai-key.txt` file in the same manner as the previous recipe.
5. Define the `get_chat_gpt_response()` function to send the prompt to ChatGPT and obtain a response:

```
client = OpenAI()

def get_chat_gpt_response(prompt):
  response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": prompt}],
    max_tokens=600,
    temperature=0.7
  )
  text = response.choices[0].message.content.strip()
  return text
```

6. Receive user input for the filename and read the content of the file:

```
file = input("ManPageGPT> $ Enter the name of a tool: ")
feed = open_file(file)
```

7. Replace the `<<INPUT>>` variable in the `prompt.txt` file with the content of the file:

```
prompt = open_file("prompt.txt").replace('<<INPUT>>', feed)
```

8. Create the `prompt.txt` file with the following text:

```
Provide the manual-page output for the following tool. Provide
the output exactly as it would appear in an actual Linux
terminal and nothing else before or after the manual-page
output.

<<INPUT>>
```

9. Send the modified prompt to the `get_chat_gpt_response()` function and print the result:

```
analysis = get_chat_gpt_response(prompt)
print(analysis)
```

Here's an example of how the complete script should look:

```
import openai
from openai import OpenAI

def open_file(filepath):
    with open(filepath, 'r', encoding='UTF-8') as infile:
        return infile.read()

openai.api_key = open_file('openai-key.txt')

client = OpenAI()
def get_chat_gpt_response(prompt):
  response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": prompt}],
    max_tokens=600,
    temperature=0.7
  )
```

```
    text = response['choices'][0]['message']['content'].strip()
    return text

feed = input("ManPageGPT> $ Enter the name of a tool: ")

prompt = open_file("prompt.txt").replace('<<INPUT>>', feed)

analysis = get_chat_gpt_response(prompt)
print(analysis)
```

How it works…

In this example, we created a Python script that utilizes a text file as a prompt template. The text file contains a variable called `<<INPUT>>` that can be replaced with any content, allowing for dynamic modification of the prompt without the need to change the overall structure. Specifically for this case, we are replacing it with user input:

1. The `openai` module is imported to access the ChatGPT API, and the `os` module is imported to interact with the operating system and manage environment variables.
2. The `open_file()` function is defined to open and read a file. It takes a file path as an argument, opens the file with read access and UTF-8 encoding, reads the content, and then returns the content.
3. The API key for accessing ChatGPT is set up by reading it from a file using the `open_file()` function and then assigning it to `openai.api_key`.
4. The `get_chat_gpt_response()` function is defined to send a prompt to ChatGPT and return the response. It takes the prompt as an argument, configures the API request with the desired settings, and then sends the request to the ChatGPT API. It extracts the response text, removes leading and trailing whitespaces, and returns it.
5. The script receives user input for the Linux command. This content will be used to replace the placeholder in the prompt template.
6. The `<<INPUT>>` variable in the `prompt.txt` file is replaced with the content of the file provided by the user. This is done using Python's string `replace()` method, which searches for the specified placeholder and replaces it with the desired content.
7. **Prompt explanation**: For this particular prompt, we tell ChatGPT exactly what type of output and formatting we are expecting since it has access to just about every manual page entry that can be found on the internet. By instructing it to provide nothing before or after the Linux-specific output, ChatGPT will not provide any additional details or narrative, and the output will resemble actual Linux output when using the `man` command.

8. The modified prompt, with the `<<INPUT>>` placeholder replaced, is sent to the `get_chat_gpt_response()` function. The function sends the prompt to ChatGPT, which retrieves the response, and the script prints the analysis result. This demonstrates how to use a prompt template with a variable that can be replaced to create customized prompts for different inputs.

This approach is particularly useful in a cybersecurity context as it allows you to create standard prompt templates for different types of analysis or queries and easily modify the input data as needed.

There's more...

1. **Use multiple variables in your prompt template**: You can use more than one variable in your prompt template to make it even more versatile. For example, you can create a template with placeholders for different components of a cybersecurity analysis, such as IP addresses, domain names, and user agents. Just make sure you replace all the necessary variables before sending the prompt to ChatGPT.
2. **Customize the variable format**: Instead of using the `<<INPUT>>` format, you can customize your variable format to better suit your needs or preferences. For example, you can use curly braces (for example, `{input}`) or any other format that you find more readable and manageable.
3. **Use environment variables for sensitive data**: When working with sensitive data such as API keys, it's recommended to use environment variables to store them securely. You can modify the `open_file()` function to read an environment variable instead of a file, ensuring that sensitive data is not accidentally leaked or exposed.
4. **Error handling and input validation**: To make your script more robust, you can add error handling and input validation. This can help you catch common issues, such as missing or improperly formatted files, and provide clear error messages to guide the user in correcting the problem.

By exploring these additional techniques, you can create more powerful, flexible, and secure prompt templates for use with ChatGPT in your cybersecurity projects.

2
Vulnerability Assessment

Building on the fundamental knowledge and skills established in *Chapter 1*, this chapter explores using ChatGPT and the OpenAI API to assist with and automate many vulnerability assessment tasks.

Throughout this chapter, you'll discover how to employ ChatGPT in creating vulnerability and threat assessment plans, a crucial part of any cybersecurity strategy. You'll see how automating these processes using the OpenAI API and Python can offer even more efficiency, especially in environments with numerous network configurations or recurring planning needs.

Additionally, this chapter will delve into using ChatGPT in conjunction with the MITRE ATT&CK framework, a globally accessible knowledge base of adversary tactics and techniques. This fusion will enable you to generate detailed threat reports, providing valuable insights for threat analysis, attack vector assessment, and threat hunting.

You'll be introduced to the concept of **Generative Pre-training Transformer** (**GPT**)-assisted vulnerability scanning. This approach simplifies some of the complexity of vulnerability scanning, transforming natural language requests into accurate command strings that can be executed in **command-line interfaces** (**CLIs**). This methodology is not only a time-saver but also enhances accuracy and understanding in performing vulnerability scans.

Lastly, this chapter will tackle the challenge of analyzing large vulnerability assessment reports. Using the OpenAI API in conjunction with LangChain, a framework designed to enable language models to assist with complex tasks, you'll see how large documents can be processed and understood, despite the current token limitations of ChatGPT.

In this chapter, we will cover the following recipes:

- Creating Vulnerability Assessment Plans
- Threat Assessment using ChatGPT and the MITRE ATT&CK framework
- GPT-Assisted Vulnerability Scanning
- Analyzing Vulnerability Assessment Reports using LangChain

Technical requirements

For this chapter, you will need a **web browser** and a stable **internet connection** to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account set up and have obtained your API key. If not, revisit *Chapter 1* for details. Basic familiarity with the Python programming language and working with the command line is necessary, as you'll be using **Python 3.x**, which needs to be installed on your system, for working with the OpenAI GPT API and creating Python scripts. A **code editor** will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter.

The code files for this chapter can be found here: `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

Creating Vulnerability Assessment Plans

In this recipe, you'll learn how to harness the power of **ChatGPT** and the **OpenAI API** to **create comprehensive vulnerability assessment plans** using network, system, and business details as input. This recipe is invaluable for both cybersecurity students and beginners looking to familiarize themselves with proper methods and tools for vulnerability assessments, as well as experienced cybersecurity professionals aiming to save time on planning and documentation.

Building upon the skills acquired in *Chapter 1*, you will delve deeper into establishing the system role of a cybersecurity professional specializing in vulnerability assessments. You'll learn how to craft effective prompts that generate well-formatted output using Markdown language. This recipe will also expand on the techniques explored in the *Enhancing Output with Templates (Application: Threat Report)* and *Formatting Output as a Table (Application: Security Controls Table)* recipes in *Chapter 1*, enabling you to design prompts that produce the desired output format.

Finally, you'll discover how to use the OpenAI API and **Python** to generate a vulnerability assessment plan, and then **export it as a Microsoft Word file**. This recipe will serve as a practical guide for creating detailed and efficient vulnerability assessment plans using ChatGPT and the OpenAI API.

Getting ready

Before diving into the recipe, you should already have your OpenAI account set up and obtained your API key. If not, revisit *Chapter 1* for details. You will also need to be sure you have the following Python libraries installed:

1. `python-docx`: This library will be used to generate Microsoft Word files. You can install it using the `pip install python-docx` command.
2. `tqdm`: This library will be used to display progress bars. You can install it using the `pip install tqdm` command.

How to do it…

In this section, we will walk you through the process of using ChatGPT to create a comprehensive vulnerability assessment plan tailored to a specific network and organization's needs. By providing the necessary details and using the given system role and prompt, you will be able to generate a well-structured assessment plan:

1. Begin by logging in to your ChatGPT account and navigating to the ChatGPT web UI.
2. Start a new conversation with ChatGPT by clicking the **New chat** button.
3. Enter the following prompt to establish a system role:

   ```
   You are a cybersecurity professional specializing in
   vulnerability assessment.
   ```

4. Enter the following message text, but replace the placeholders in the { } brackets with the appropriate data of your choice. You can either combine this prompt with the system role or enter it separately as follows:

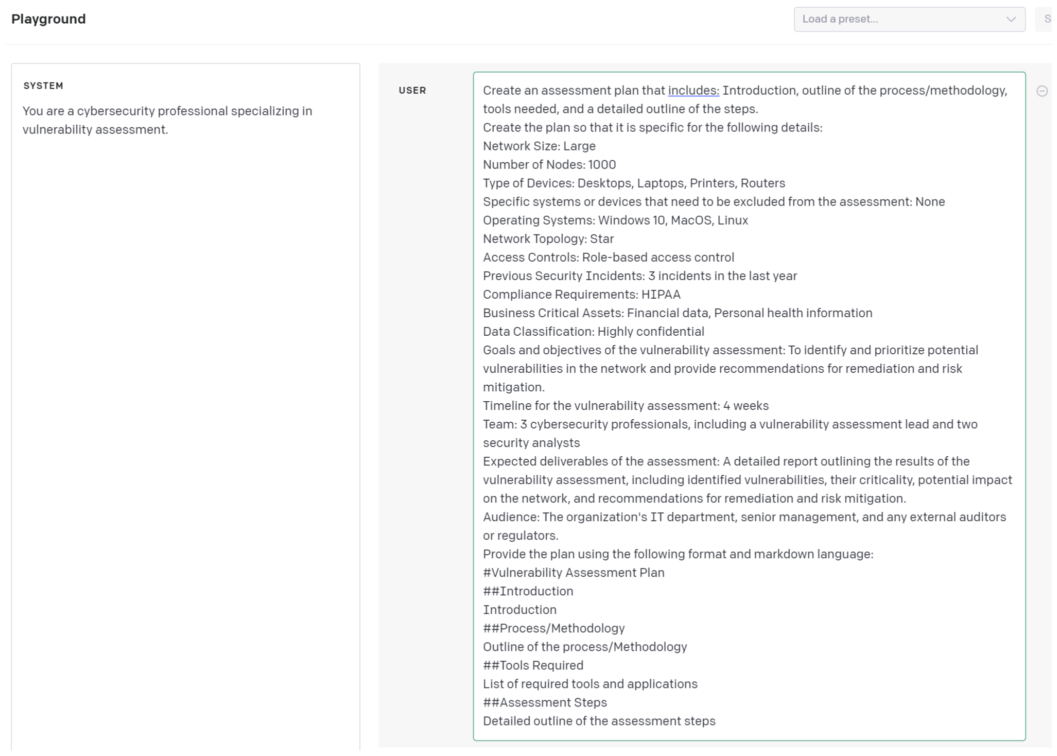
   ```
   Using cybersecurity industry standards and best practices,
   create a complete and detailed assessment plan (not a
   penetration test) that includes: Introduction, outline of
   the process/methodology, tools needed, and a very detailed
   multi-layered outline of the steps. Provide a thorough and
   descriptive introduction and as much detail and description as
   possible throughout the plan. The plan should not be the only
   assessment of technical vulnerabilities on systems but also
   policies, procedures, and compliance. It should include the
   use of scanning tools as well as configuration review, staff
   interviews, and site walk-around. All recommendations should
   follow industry standard best practices and methods. The plan
   should be a minimum of 1500 words.
   Create the plan so that it is specific for the following
   details:
   Network Size: {Large}
   Number of Nodes: {1000}
   Type of Devices: {Desktops, Laptops, Printers, Routers}
   Specific systems or devices that need to be excluded from the
   assessment: {None}
   Operating Systems: {Windows 10, MacOS, Linux}
   Network Topology: {Star}
   Access Controls: {Role-based access control}
   Previous Security Incidents: {3 incidents in the last year}
   Compliance Requirements: {HIPAA}
   Business Critical Assets: {Financial data, Personal health
   information}
   ```

```
Data Classification: {Highly confidential}
Goals and objectives of the vulnerability assessment: {To
identify and prioritize potential vulnerabilities in the
network and provide recommendations for remediation and risk
mitigation.}
Timeline for the vulnerability assessment: {4 weeks{
Team: {3 cybersecurity professionals, including a vulnerability
assessment lead and two security analysts}
Expected deliverables of the assessment: {A detailed report
outlining the results of the vulnerability assessment, including
identified vulnerabilities, their criticality, potential impact
on the network, and recommendations for remediation and risk
mitigation.}
Audience: {The organization's IT department, senior management,
and any external auditors or regulators.}
Provide the plan using the following format and markdown
language:
#Vulnerability Assessment Plan
##Introduction
Thorough Introduction to the plan including the scope, reasons
for doing it, goals and objectives, and summary of the plan
##Process/Methodology
Description and Outline of the process/Methodology
##Tools Required
List of required tools and applications, with their descriptions
and reasons needed
##Assessment Steps
Detailed, multi-layered outline of the assessment steps
```

Hint

If you are performing this in the **OpenAI Playground**, it is advisable to use **Chat mode** and enter the role in the **System** window, and the prompt in the **User message** window.

Figure 2.1 shows the system role and user prompt entered into the **OpenAI Playground**:

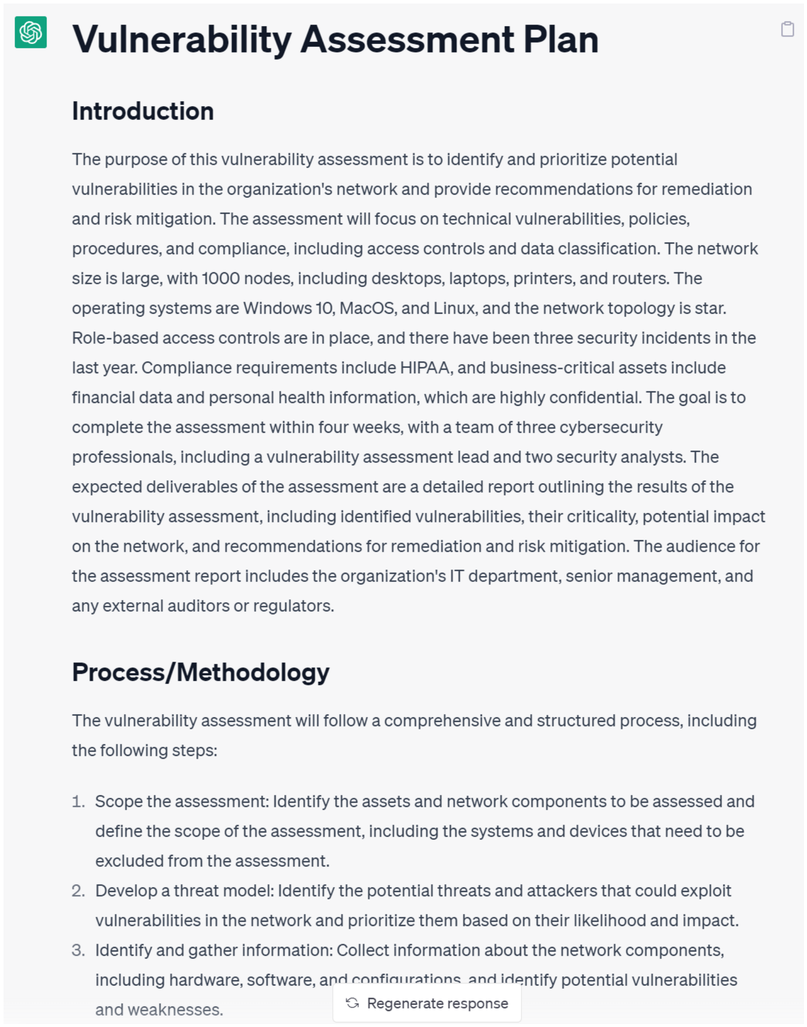


You are a cybersecurity professional specializing in vulnerability assessment.

```
Create an assessment plan that includes: Introduction, outline of the process/methodology, tools needed, and a detailed outline of the steps.
Create the plan so that it is specific for the following details:
Network Size: Large
Number of Nodes: 1000
Type of Devices: Desktops, Laptops, Printers, Routers
Specific systems or devices that need to be excluded from the assessment: None
Operating Systems: Windows 10, MacOS, Linux
Network Topology: Star
Access Controls: Role-based access control
Previous Security Incidents: 3 incidents in the last year
Compliance Requirements: HIPAA
Business Critical Assets: Financial data, Personal health information
Data Classification: Highly confidential
Goals and objectives of the vulnerability assessment: To identify and prioritize potential vulnerabilities in the network and provide recommendations for remediation and risk mitigation.
Timeline for the vulnerability assessment: 4 weeks
Team: 3 cybersecurity professionals, including a vulnerability assessment lead and two security analysts
Expected deliverables of the assessment: A detailed report outlining the results of the vulnerability assessment, including identified vulnerabilities, their criticality, potential impact on the network, and recommendations for remediation and risk mitigation.
Audience: The organization's IT department, senior management, and any external auditors or regulators.
Provide the plan using the following format and markdown language:
#Vulnerability Assessment Plan
##Introduction
Introduction
##Process/Methodology
Outline of the process/Methodology
##Tools Required
List of required tools and applications
##Assessment Steps
Detailed outline of the assessment steps
```

Figure 2.1 – OpenAI Playground method

5. Review the generated output from ChatGPT. If the output is satisfactory and meets the requirements, you can proceed to the next step. If not, you can either refine your prompt or rerun the conversation to generate a new output.
6. Once you have obtained the desired output, you can use the generated Markdown to create a well-structured vulnerability assessment plan in your preferred text editor or Markdown viewer.
7. *Figure 2.2* shows an example ChatGPT generation of a vulnerability assessment plan using Markdown language formatting:



Vulnerability Assessment Plan

Introduction

The purpose of this vulnerability assessment is to identify and prioritize potential vulnerabilities in the organization's network and provide recommendations for remediation and risk mitigation. The assessment will focus on technical vulnerabilities, policies, procedures, and compliance, including access controls and data classification. The network size is large, with 1000 nodes, including desktops, laptops, printers, and routers. The operating systems are Windows 10, MacOS, and Linux, and the network topology is star. Role-based access controls are in place, and there have been three security incidents in the last year. Compliance requirements include HIPAA, and business-critical assets include financial data and personal health information, which are highly confidential. The goal is to complete the assessment within four weeks, with a team of three cybersecurity professionals, including a vulnerability assessment lead and two security analysts. The expected deliverables of the assessment are a detailed report outlining the results of the vulnerability assessment, including identified vulnerabilities, their criticality, potential impact on the network, and recommendations for remediation and risk mitigation. The audience for the assessment report includes the organization's IT department, senior management, and any external auditors or regulators.

Process/Methodology

The vulnerability assessment will follow a comprehensive and structured process, including the following steps:

1. Scope the assessment: Identify the assets and network components to be assessed and define the scope of the assessment, including the systems and devices that need to be excluded from the assessment.
2. Develop a threat model: Identify the potential threats and attackers that could exploit vulnerabilities in the network and prioritize them based on their likelihood and impact.
3. Identify and gather information: Collect information about the network components, including hardware, software, and configurations, and identify potential vulnerabilities and weaknesses.

Figure 2.2 – Example ChatGPT assessment plan output

How it works…

This GPT-assisted vulnerability assessment plan recipe leverages the sophistication of **natural language processing** (**NLP**) and **machine learning (ML) algorithms** to generate a comprehensive and detailed vulnerability assessment plan. By adopting a specific system role and an elaborate user request as a prompt, ChatGPT is able to customize its response to meet the requirements of a seasoned cybersecurity professional who is tasked with assessing an extensive network system.

Here's a closer look at how this process works:

- **System role and detailed prompt**: The system role designates ChatGPT as a seasoned cybersecurity professional specializing in vulnerability assessment. The prompt, which serves as the user request, is detailed and outlines the specifics of the assessment plan, from the size of the network and types of devices to the required compliance and the expected deliverables. These inputs provide context and guide ChatGPT's response, ensuring it is tailored to the complexities and requirements of the vulnerability assessment task.
- **NLP and ML**: NLP and ML form the bedrock of ChatGPT's capabilities. It applies these technologies to understand the intricacies of the user request, learn from the patterns, and generate a well-structured vulnerability assessment plan that is detailed, specific, and actionable.
- **Knowledge and language understanding capabilities**: ChatGPT uses its extensive knowledge base and language understanding capabilities to conform to industry-standard methodologies and best practices. This is particularly important in the rapidly evolving field of cybersecurity, ensuring that the resulting vulnerability assessment plan is up to date and adheres to recognized standards.
- **Markdown language output**: The use of Markdown language output ensures that the plan is formatted in a consistent and easy-to-read manner. This format can be easily integrated into reports, presentations, and other formal documents, which is crucial when communicating the plan to IT departments, senior management, and external auditors or regulators.
- **Streamlining the assessment planning process**: The overall advantage of using this GPT-assisted vulnerability assessment plan recipe is that it streamlines the process of creating a comprehensive vulnerability assessment plan. You save time on planning and documentation and can generate a professional-grade assessment plan that aligns with industry standards and is tailored to the specific needs of your organization.

By applying these detailed inputs, you transform ChatGPT into a potential tool that can assist in creating a comprehensive, tailored vulnerability assessment plan. This not only bolsters your cybersecurity efforts but also ensures your resources are utilized effectively in protecting your network systems.

There's more…

In addition to using ChatGPT to generate a vulnerability assessment plan, you can also use the OpenAI API and Python to automate the process. This approach is particularly useful when you have a large number of network configurations to assess or when you need to generate plans on a recurring basis.

The Python script we will present here reads input data from a text file and uses it to fill in the placeholders in the prompt. The resulting Markdown output can then be used to create a well-structured vulnerability assessment plan.

While the process is similar to the ChatGPT version, the use of the OpenAI API provides additional flexibility and control over the generated content. Let's dive into the steps involved in the OpenAI API version of the vulnerability assessment plan recipe:

1. Import the necessary libraries and set up the OpenAI API:

```
import openai
from openai import OpenAI
import os
from docx import Document
from tqdm import tqdm
import threading
import time
from datetime import datetime

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")
```

   In this section, we import the necessary libraries, such as `openai`, `os`, `docx`, `tqdm`, `threading`, `time`, and `datetime`. We also set up the OpenAI API by providing the API key.

2. Read user input data from a text file:

```
def read_user_input_file(file_path: str) -> dict:
    user_data = {}
    with open(file_path, 'r') as file:
        for line in file:
            key, value = line.strip().split(':')
            user_data[key.strip()] = value.strip()
    return user_data

user_data_file = "assessment_data.txt"
user_data = read_user_input_file(user_data_file)
```

   Here, we define a `read_user_input_file` function that reads the user input data from a text file and stores it in a dictionary. We then call this function with the `assessment_data.txt` file to obtain the `user_data` dictionary.

3. Generate a vulnerability assessment plan using the OpenAI API:

Important note

The …' notation signifies that we will fill in this section of code in a later step.

```
def generate_report(network_size,
                    number_of_nodes,
                    type_of_devices,
                    special_devices,
                    operating_systems,
                    network_topology,
                    access_controls,
                    previous_security_incidents,
                    compliance_requirements,
                    business_critical_assets,
                    data_classification,
                    goals,
                    timeline,
                    team,
                    deliverables,
                    audience: str) -> str:
    # Define the conversation messages
    messages = [ ... ]

    client = OpenAI()

# Call the OpenAI API
response = client.chat.completions.create( ... )

    # Return the generated text
    return response.choices[0].message.content.strip()
```

In this code block, we define a `generate_report` function, which takes the user input data and calls the OpenAI API to generate the vulnerability assessment plan. The function returns the generated text.

4. Define the API messages:

```
# Define the conversation messages
messages = [
    {"role": "system", "content": "You are a cybersecurity
professional specializing in vulnerability assessment."},
    {"role": "user", "content": f'Using cybersecurity industry
standards and best practices, create a complete and detailed
assessment plan ... Detailed outline of the assessment steps'}
]

client = OpenAI()

# Call the OpenAI API
response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=messages,
    max_tokens=2048,
    n=1,
    stop=None,
    temperature=0.7,
)

# Return the generated text
return return response.choices[0].message.content.strip()
```

In the conversation messages, we define two roles: `system` and `user`. The `system` role is used to set the context for the AI model, informing it that it's a cybersecurity professional specializing in vulnerability assessment. The `user` role provides the instructions for the AI, which include generating a detailed vulnerability assessment plan based on industry standards, best practices, and user-supplied data.

The `system` role helps set the stage for the AI, while the `user` role guides the AI in its content generation. This approach follows a similar pattern to the ChatGPT UI section we discussed earlier, where we provided an initial message to the AI to set the context.

For more information on sending API requests and handling responses, please refer to the *Sending API Requests and Handling Responses with Python* recipe in *Chapter 1*. This recipe provides a deeper understanding of interacting with the OpenAI API, including how to structure requests and process the generated content.

5. Convert the generated Markdown text to a Word document:

```
def markdown_to_docx(markdown_text: str, output_file: str):
    document = Document()

    # Iterate through the lines of the markdown text
    for line in markdown_text.split('\n'):
        # Add headings and paragraphs based on the markdown
formatting
        ...

    # Save the Word document
    document.save(output_file)
```

The `markdown_to_docx` function converts the generated Markdown text to a Word document. It iterates through the lines of the Markdown text, adding headings and paragraphs based on the Markdown formatting, and saves the resulting Word document.

6. Display the elapsed time while waiting for the API call:

```
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rCommunicating with the API - Elapsed time: 
{elapsed_time:.2f} seconds", end="")
        time.sleep(1)
```

The `display_elapsed_time` function is used to display the elapsed time while waiting for the API call to complete. It uses a loop to print the elapsed time in seconds.

7. Write the main function:

```
current_datetime = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
assessment_name = f"Vuln_ Assessment_Plan_{current_datetime}"

api_call_completed = False
elapsed_time_thread = threading.Thread(target=display_elapsed_
time)
elapsed_time_thread.start()

try:
    # Generate the report using the OpenAI API
    report = generate_report(
    user_data["Network Size"],
    user_data["Number of Nodes"],
    user_data["Type of Devices"],
    user_data["Specific systems or devices that need to be 
excluded from the assessment"],
    user_data["Operating Systems"],
    user_data["Network Topology"],
    user_data["Access Controls"],
    user_data["Previous Security Incidents"],
    user_data["Compliance Requirements"],
    user_data["Business Critical Assets"],
    user_data["Data Classification"],
    user_data["Goals and objectives of the vulnerability 
assessment"],
    user_data["Timeline for the vulnerability assessment"],
    user_data["Team"],
    user_data["Expected deliverables of the assessment"],
    user_data["Audience"]
    )

    api_call_completed = True
    elapsed_time_thread.join()
except Exception as e:
    api_call_completed = True
```

```
        elapsed_time_thread.join()
        print(f"\nAn error occurred during the API call: {e}")
        exit()

    # Save the report as a Word document
    docx_output_file = f"{assessment_name}_report.docx"

    # Handle exceptions during the report generation
    try:
        with tqdm(total=1, desc="Generating plan") as pbar:
            markdown_to_docx(report, docx_output_file)
            pbar.update(1)
        print("\nPlan generated successfully!")
    except Exception as e:
        print(f"\nAn error occurred during the plan generation:
    {e}")
```

In the main part of the script, we start by defining an `assessment_name` function based on the current date and time. We then use threading to display the elapsed time while making the API call. The script calls the `generate_report` function with the user data as arguments, and upon successful completion, it saves the generated report as a Word document using the `markdown_to_docx` function. The progress is displayed using the `tqdm` library. If any errors occur during the API call or report generation, they are displayed to the user.

Hint

You can swap out the **chat-3.5-turbo** model with the **GPT-4** model, if you are a ChatGPT Plus subscriber, for often improved results. In fact, GPT-4 is capable of generating a much longer and more detailed generation and/or document. Just keep in mind that the GPT-4 model is a bit more expensive than the chat-3.5-turbo model.

Here is how the completed script should look:

```
import openai
from openai import OpenAI
import os
from docx import Document
from tqdm import tqdm
import threading
import time
from datetime import datetime

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")
```

```
current_datetime = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
assessment_name = f"Vuln_Assessment_Plan_{current_datetime}"

def read_user_input_file(file_path: str) -> dict:
    user_data = {}
    with open(file_path, 'r') as file:
        for line in file:
            key, value = line.strip().split(':')
            user_data[key.strip()] = value.strip()
    return user_data

user_data_file = "assessment_data.txt"
user_data = read_user_input_file(user_data_file)

# Function to generate a report using the OpenAI API
def generate_report(network_size,
                    number_of_nodes,
                    type_of_devices,
                    special_devices,
                    operating_systems,
                    network_topology,
                    access_controls,
                    previous_security_incidents,
                    compliance_requirements,
                    business_critical_assets,
                    data_classification,
                    goals,
                    timeline,
                    team,
                    deliverables,
                    audience: str) -> str:

    # Define the conversation messages
    messages = [
        {"role": "system", "content": "You are a cybersecurity 
professional specializing in vulnerability assessment."},
        {"role": "user", "content": f'Using cybersecurity industry 
standards and best practices, create a complete and detailed 
assessment plan (not a penetration test) that includes: Introduction, 
outline of the process/methodology, tools needed, and a very 
detailed multi-layered outline of the steps. Provide a thorough 
and descriptive introduction and as much detail and description as 
possible throughout the plan. The plan should not only assessment of 
technical vulnerabilities on systems but also policies, procedures,
and compliance. It should include the use of scanning tools as well
as configuration review, staff interviews, and site walk-around. All
recommendations should follow industry standard best practices and
methods. The plan should be a minimum of 1500 words.\n\
        Create the plan so that it is specific for the following
details:\n\
        Network Size: {network_size}\n\
        Number of Nodes: {number_of_nodes}\n\
        Type of Devices: {type_of_devices}\n\
        Specific systems or devices that need to be excluded from the
assessment: {special_devices}\n\
        Operating Systems: {operating_systems}\n\
        Network Topology: {network_topology}\n\
        Access Controls: {access_controls}\n\
        Previous Security Incidents: {previous_security_incidents}\n\
        Compliance Requirements: {compliance_requirements}\n\
        Business Critical Assets: {business_critical_assets}\n\
        Data Classification: {data_classification}\n\
        Goals and objectives of the vulnerability assessment:
{goals}\n\
        Timeline for the vulnerability assessment: {timeline}\n\
        Team: {team}\n\
        Expected deliverables of the assessment: {deliverables}\n\
        Audience: {audience}\n\
        Provide the plan using the following format and observe the
markdown language:\n\
        #Vulnerability Assessment Plan\n\
        ##Introduction\n\
        Introduction\n\
        ##Process/Methodology\n\
        Outline of the process/Methodology\n\
        ##Tools Required\n\
        List of required tools and applications\n\
        ##Assessment Steps\n\
        Detailed outline of the assessment steps'}
    ]

    client = OpenAI()

    # Call the OpenAI API
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    # Return the generated text
    return response.choices[0].message.content.strip()

# Function to convert markdown text to a Word document
def markdown_to_docx(markdown_text: str, output_file: str):
    document = Document()

    # Iterate through the lines of the markdown text
    for line in markdown_text.split('\n'):

        # Add headings based on the markdown heading levels
        if line.startswith('# '):
            document.add_heading(line[2:], level=1)
        elif line.startswith('## '):
            document.add_heading(line[3:], level=2)
        elif line.startswith('### '):
            document.add_heading(line[4:], level=3)
        elif line.startswith('#### '):
            document.add_heading(line[5:], level=4)
        # Add paragraphs for other text
        else:
            document.add_paragraph(line)

    # Save the Word document
    document.save(output_file)

# Function to display elapsed time while waiting for the API call
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rCommunicating with the API - Elapsed time: {elapsed_
time:.2f} seconds", end="")
        time.sleep(1)

api_call_completed = False
elapsed_time_thread = threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()

# Handle exceptions during the API call
```

```
try:
    # Generate the report using the OpenAI API
    report = generate_report(
    user_data["Network Size"],
    user_data["Number of Nodes"],
    user_data["Type of Devices"],
    user_data["Specific systems or devices that need to be excluded
from the assessment"],
    user_data["Operating Systems"],
    user_data["Network Topology"],
    user_data["Access Controls"],
    user_data["Previous Security Incidents"],
    user_data["Compliance Requirements"],
    user_data["Business Critical Assets"],
    user_data["Data Classification"],
    user_data["Goals and objectives of the vulnerability assessment"],
    user_data["Timeline for the vulnerability assessment"],
    user_data["Team"],
    user_data["Expected deliverables of the assessment"],
    user_data["Audience"]
    )

    api_call_completed = True
    elapsed_time_thread.join()
except Exception as e:
    api_call_completed = True
    elapsed_time_thread.join()
    print(f"\nAn error occurred during the API call: {e}")
    exit()

# Save the report as a Word document
docx_output_file = f"{assessment_name}_report.docx"

# Handle exceptions during the report generation
try:
    with tqdm(total=1, desc="Generating plan") as pbar:
        markdown_to_docx(report, docx_output_file)
        pbar.update(1)
    print("\nPlan generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the plan generation: {e}")
```

This script automates the process of generating a vulnerability assessment plan by using the OpenAI API in conjunction with Python. It starts by importing the necessary libraries and setting up the OpenAI API. It then reads user input data from a text file (the file path is stored as a `user_data_file` string) and then stores this data in a dictionary for easy access.

The core of the script is the function that generates the vulnerability assessment plan. It leverages the OpenAI API to create a detailed report based on the user input data. The conversation with the API is formatted with both `system` and `user` roles to guide the generation process effectively.

Once the report is generated, it is converted from Markdown text to a Word document, providing a well-structured, readable output. To provide user feedback during the process, the script includes a function that displays the elapsed time while the API call is being made.

Finally, the script's main function ties everything together. It initiates the process of generating the report using the OpenAI API, shows the elapsed time during the API call, and finally, converts the generated report to a Word document. If any errors occur during the API call or the document generation, they are handled and displayed to the user.

Threat Assessment using ChatGPT and the MITRE ATT&CK framework

In this recipe, you will learn how to leverage **ChatGPT** and the **OpenAI API** to conduct a threat assessment by providing a threat, attack, or campaign name. By combining the power of ChatGPT with the **MITRE ATT&CK** framework, you will be able to generate detailed threat reports, **tactics, techniques, and procedures** (**TTPs**) mappings, and associated **indicators of compromise** (**IoCs**). This information will enable cybersecurity professionals to analyze attack vectors in their environment and extend their capabilities into threat hunting.

Building upon the skills acquired in *Chapter 1*, this recipe will guide you through establishing the system role of a cybersecurity analyst and engineering effective prompts that generate well-formatted output, including tables. You will learn how to design prompts to obtain the desired output from ChatGPT using both the ChatGPT web UI and a Python script. Additionally, you will learn how to use the OpenAI API to generate a comprehensive threat report in a Microsoft Word file format.

Getting ready

Before diving into the recipe, you should already have your OpenAI account set up and obtained your API key. If not, revisit *Chapter 1* for details. You will also need to do the following:

1. **Install the python-docx library**: Ensure you have the `python-docx` library installed in your Python environment, as it will be used to generate Microsoft Word files. You can install it using the `pip install python-docx` command.

2. **Familiarize yourself with the MITRE ATT&CK framework**: To make the most of this recipe, it's helpful to have a basic understanding of the MITRE ATT&CK framework. Visit `https://attack.mitre.org/` for more information and resources.
3. **List sample threats**: Prepare a list of sample threat names, attack campaigns, or adversary groups to use as examples while working through the recipe.

How to do it…

By following these steps, you can successfully utilize ChatGPT to generate a TTP-based threat report using the MITRE ATT&CK framework and proper Markdown formatting. We will be specifying the name of a threat and applying prompt engineering techniques. ChatGPT will then generate a well-formatted report with valuable insights that can assist you in threat analysis, attack vector assessment, and even in gathering IoCs for threat hunting:

1. Begin by logging in to your ChatGPT account and navigating to the ChatGPT web UI.
2. Start a new conversation with ChatGPT by clicking the **New chat** button.
3. Enter the following prompt to establish a system role:

```
You are a professional cyber threat analyst and MITRE ATT&CK
Framework expert.
```

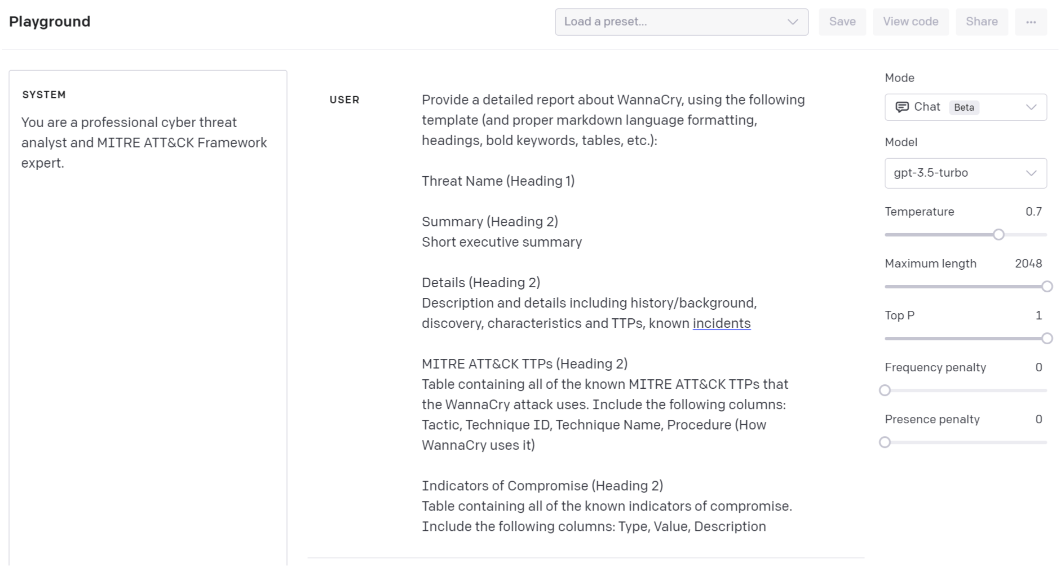
4. Replace `{threat_name}` in the user prompt below with the threat name of your choice (in our example, we will use **WannaCry**). You can either combine this prompt with the system role or enter it separately:

```
Provide a detailed report about {threat_name}, using the
following template (and proper markdown language formatting,
headings, bold keywords, tables, etc.):
Threat Name (Heading 1)
Summary (Heading 2)
Short executive summary
Details (Heading 2)
Description and details including history/background, discovery,
characteristics and TTPs, known incidents
MITRE ATT&CK TTPs (Heading 2)
Table containing all of the known MITRE ATT&CK TTPs that the
{threat_name} attack uses. Include the following columns:
Tactic, Technique ID, Technique Name, Procedure (How WannaCry
uses it)
Indicators of Compromise (Heading 2)
Table containing all of the known indicators of compromise.
Include the following columns: Type, Value, Description
```

Hint

Just as with the previous recipe, you can perform this in the **OpenAI Playground** and use **Chat mode** to enter the role in the **System** window, and the prompt in the **User message** window.

Figure 2.3 shows the system role and user prompt entered into the **OpenAI Playground**:



SYSTEM

You are a professional cyber threat analyst and MITRE ATT&CK Framework expert.

USER

Provide a detailed report about WannaCry, using the following template (and proper markdown language formatting, headings, bold keywords, tables, etc.):

Threat Name (Heading 1)

Summary (Heading 2)
Short executive summary

Details (Heading 2)
Description and details including history/background, discovery, characteristics and TTPs, known incidents

MITRE ATT&CK TTPs (Heading 2)
Table containing all of the known MITRE ATT&CK TTPs that the WannaCry attack uses. Include the following columns: Tactic, Technique ID, Technique Name, Procedure (How WannaCry uses it)

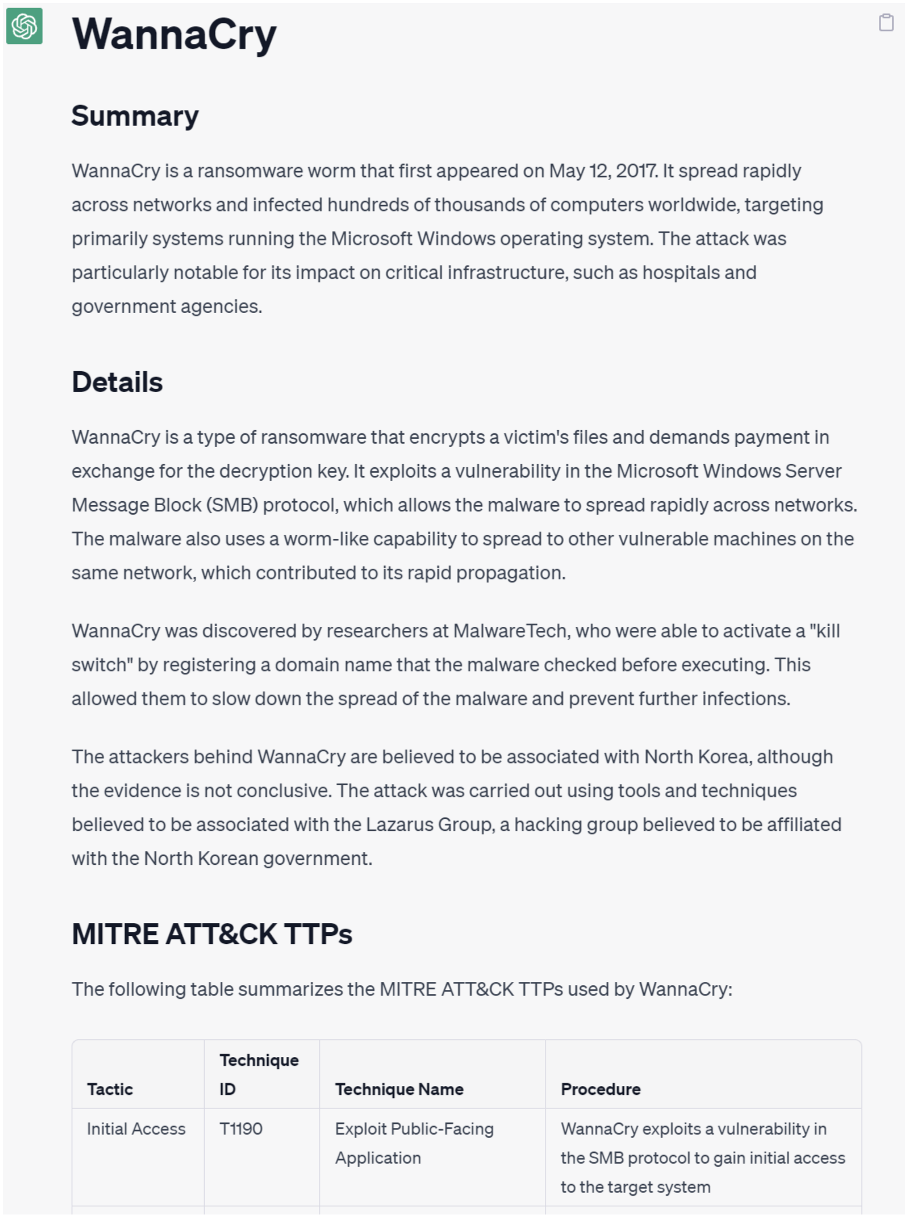
Indicators of Compromise (Heading 2)
Table containing all of the known indicators of compromise. Include the following columns: Type, Value, Description

Mode	Chat Beta
Model	gpt-3.5-turbo
Temperature	0.7
Maximum length	2048
Top P	1
Frequency penalty	0
Presence penalty	0

Figure 2.3 – OpenAI Playground method

5. After entering the appropriate system role and user prompt, press *Enter*.
6. ChatGPT will process the prompt and generate a formatted threat report with Markdown language formatting, headings, bold keywords, tables, and other elements specified in the prompt.

Figure 2.4 and *Figure 2.5* illustrate an example ChatGPT generation of a threat report using Markdown language formatting with a table:



WannaCry

Summary

WannaCry is a ransomware worm that first appeared on May 12, 2017. It spread rapidly across networks and infected hundreds of thousands of computers worldwide, targeting primarily systems running the Microsoft Windows operating system. The attack was particularly notable for its impact on critical infrastructure, such as hospitals and government agencies.

Details

WannaCry is a type of ransomware that encrypts a victim's files and demands payment in exchange for the decryption key. It exploits a vulnerability in the Microsoft Windows Server Message Block (SMB) protocol, which allows the malware to spread rapidly across networks. The malware also uses a worm-like capability to spread to other vulnerable machines on the same network, which contributed to its rapid propagation.

WannaCry was discovered by researchers at MalwareTech, who were able to activate a "kill switch" by registering a domain name that the malware checked before executing. This allowed them to slow down the spread of the malware and prevent further infections.

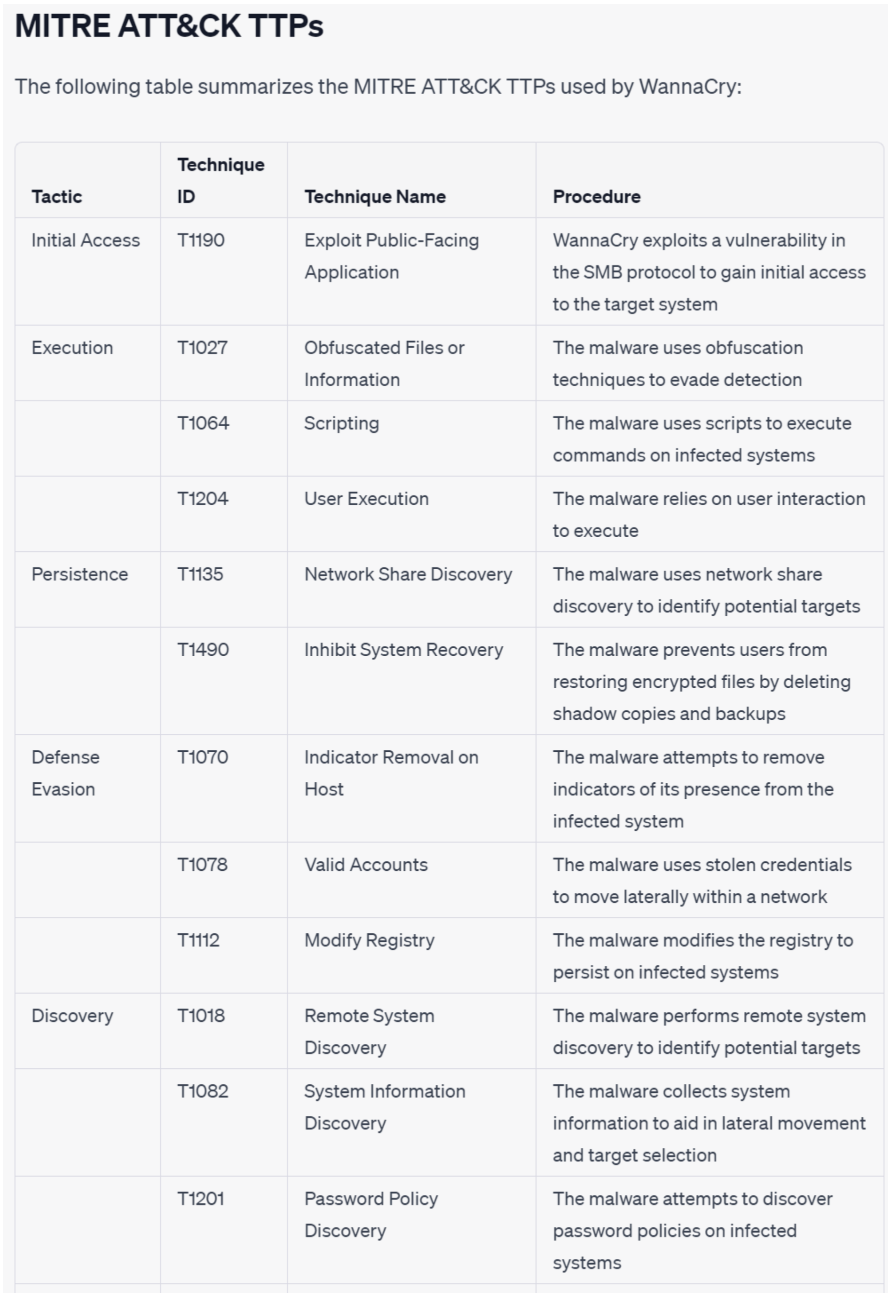
The attackers behind WannaCry are believed to be associated with North Korea, although the evidence is not conclusive. The attack was carried out using tools and techniques believed to be associated with the Lazarus Group, a hacking group believed to be affiliated with the North Korean government.

MITRE ATT&CK TTPs

The following table summarizes the MITRE ATT&CK TTPs used by WannaCry:

Tactic	Technique ID	Technique Name	Procedure
Initial Access	T1190	Exploit Public-Facing Application	WannaCry exploits a vulnerability in the SMB protocol to gain initial access to the target system

Figure 2.4 – ChatGPT threat report narrative output



MITRE ATT&CK TTPs

The following table summarizes the MITRE ATT&CK TTPs used by WannaCry:

Tactic	Technique ID	Technique Name	Procedure
Initial Access	T1190	Exploit Public-Facing Application	WannaCry exploits a vulnerability in the SMB protocol to gain initial access to the target system
Execution	T1027	Obfuscated Files or Information	The malware uses obfuscation techniques to evade detection
	T1064	Scripting	The malware uses scripts to execute commands on infected systems
	T1204	User Execution	The malware relies on user interaction to execute
Persistence	T1135	Network Share Discovery	The malware uses network share discovery to identify potential targets
	T1490	Inhibit System Recovery	The malware prevents users from restoring encrypted files by deleting shadow copies and backups
Defense Evasion	T1070	Indicator Removal on Host	The malware attempts to remove indicators of its presence from the infected system
	T1078	Valid Accounts	The malware uses stolen credentials to move laterally within a network
	T1112	Modify Registry	The malware modifies the registry to persist on infected systems
Discovery	T1018	Remote System Discovery	The malware performs remote system discovery to identify potential targets
	T1082	System Information Discovery	The malware collects system information to aid in lateral movement and target selection
	T1201	Password Policy Discovery	The malware attempts to discover password policies on infected systems

Figure 2.5 – ChatGPT threat report table output

7. Review the generated report to ensure it contains the desired information and formatting. If necessary, adjust your user prompt and resubmit it to improve the output.

Hint

Sometimes, ChatGPT will stop generating before it has completed the entire out. This is due to the token limit of the model being used. In such cases, you can click on the **Continue Generating** button.

How it works…

Just as we did in the *Applying ChatGPT Roles (Application: AI CISO)* recipe in *Chapter 1*, when you assign a role to ChatGPT, you provide a specific context or persona for the model to work with. This helps the model generate responses that are tailored to the given role, resulting in more accurate, relevant, and detailed content. The model will generate content that aligns with the expertise and perspective of the assigned role, offering better insights, opinions, or recommendations.

When we provide a threat name and direct ChatGPT to reference the MITRE ATT&CK framework, we are able to leverage its massive dataset, which includes detailed information about threats and the MITRE ATT&CK framework. As a result, it is able to correlate the two and quickly give us the relevant threat information as it pertains to the TTPs identified in the framework.

Important note

When using the current version of ChatGPT and the OpenAI API as of the time of this writing, the dataset is only trained up through September 2021. Therefore, it will not have knowledge of any threat data after that. However, we will cover techniques later in this book on how to use the API and Python to feed recent data into the request.

By providing a clear template for the output in your prompt, you guide ChatGPT to generate responses that adhere to the specified structure and formatting. This helps ensure that the generated content is consistent, well organized, and suitable for use in reports, presentations, or other formal documents. The model will focus on generating content that matches the formatting and structure you've provided while still delivering the information you requested. See the *Enhancing Output with Templates (Application: Threat Report)* and *Formatting Output as a Table (Application: Security Controls Table)* recipes in *Chapter 1* for further details.

There's more...

You can extend the power and flexibility of this recipe by using the OpenAI API with a Python script to generate a threat report, similar to the one created in the ChatGPT web UI. Here's how you do it:

1. Start by importing the necessary libraries:

```
import openai
from openai import OpenAI
import os
from docx import Document
from tqdm import tqdm
import threading
import time
```

2. Set up the OpenAI API the same as we did in the *Setting the OpenAI API key as an Environment Variable* recipe in *Chapter 1*:

```
openai.api_key = os.getenv("OPENAI_API_KEY")
```

3. Create a function to generate a report using the OpenAI API:

```
def generate_report(threat_name: str) -> str:
    ...
    return response['choices'][0]['message']['content'].strip()
```

   This function takes a threat name as input and sends it as part of a prompt to the OpenAI API. It returns the generated text from the API response.

4. Create a function to convert the generated text, which is in Markdown format, to a Microsoft Word document:

```
def markdown_to_docx(markdown_text: str, output_file: str):
    ...
    document.save(output_file)
```

   This function takes the generated text in Markdown format and an output filename. It parses the Markdown text and creates a Word document with the appropriate formatting.

5. Create a function to extract tables from the Markdown text:

```
def extract_tables(markdown_text: str):
    ...
    return tables
```

   This function iterates through the Markdown text and extracts any tables it finds.

6. Create a function to display the elapsed time while waiting for the API call:

```
def display_elapsed_time():
    ...
```

   This function shows the elapsed time in seconds while waiting for the API call to complete.

7. Get the threat name from user input:

```
threat_name = input("Enter the name of a cyber threat: ")
```

8. Start a separate thread to display the elapsed time while making the API call:

```
api_call_completed = False
elapsed_time_thread = threading.Thread(target=display_elapsed_
time)
elapsed_time_thread.start()
```

9. Make the API call and handle exceptions:

```
try:
    report = generate_report(threat_name)
    api_call_completed = True
    elapsed_time_thread.join()
except Exception as e:
    ...
```

10. Save the generated report as a Word document:

```
docx_output_file = f"{threat_name}_report.docx"
```

11. Generate the report and handle exceptions:

```
try:
    with tqdm(total=1, desc="Generating report and files") as
pbar:
        markdown_to_docx(report, docx_output_file)
    print("\nReport and tables generated successfully!")
except Exception as e:
    ...
```

Here is how the completed script should look:

```
import openai
from openai import OpenAI
import os
from docx import Document
from tqdm import tqdm
```

```
import threading
import time

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

# Function to generate a report using the OpenAI API
def generate_report(threat_name: str) -> str:

    # Define the conversation messages
    messages = [
        {"role": "system", "content": "You are a professional cyber 
threat analyst and MITRE ATT&CK Framework expert."},
        {"role": "user", "content": f'Provide a detailed report about 
{threat_name}, using the following template (and proper markdown 
language formatting, headings, bold keywords, tables, etc.):\n\n\
        Threat Name (Heading 1)\n\n\
        Summary (Heading 2)\n\
        Short executive summary\n\n\
        Details (Heading 2)\n\
        Description and details including history/background, 
discovery, characteristics and TTPs, known incidents\n\n\
        MITRE ATT&CK TTPs (Heading 2)\n\
        Table containing all of the known MITRE ATT&CK TTPs that the 
{threat_name} attack uses. Include the following columns: Tactic, 
Technique ID, Technique Name, Procedure (How {threat_name} uses it)\
n\n\
        Indicators of Compromise (Heading 2)\n\
        Table containing all of the known indicators of compromise. 
Include the following collumns: Type, Value, Description\n\n\  '}
    ]

    client = OpenAI()

    # Call the OpenAI API
    response = client.chat.completions.create
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    # Return the generated text
    return response.choices[0].message.content.strip()

# Function to convert markdown text to a Word document
def markdown_to_docx(markdown_text: str, output_file: str):
    document = Document()

    # Variables to keep track of the current table
    table = None
    in_table = False

    # Iterate through the lines of the markdown text
    for line in markdown_text.split('\n'):

        # Add headings based on the markdown heading levels
        if line.startswith('# '):
            document.add_heading(line[2:], level=1)
        elif line.startswith('## '):
            document.add_heading(line[3:], level=2)
        elif line.startswith('### '):
            document.add_heading(line[4:], level=3)
        elif line.startswith('#### '):
            document.add_heading(line[5:], level=4)
        # Handle tables in the markdown text
        elif line.startswith('|'):
            row = [cell.strip() for cell in line.split('|')[1:-1]]
            if not in_table:
                in_table = True
                table = document.add_table(rows=1, cols=len(row),
style='Table Grid')
                for i, cell in enumerate(row):
                    table.cell(0, i).text = cell
            else:
                if len(row) != len(table.columns):  # If row length
doesn't match table, it's a separator
                    continue
                new_row = table.add_row()
                for i, cell in enumerate(row):
                    new_row.cells[i].text = cell
        # Add paragraphs for other text
        else:
            if in_table:
                in_table = False
                table = None
            document.add_paragraph(line)

    # Save the Word document
    document.save(output_file)

# Function to extract tables from the markdown text
def extract_tables(markdown_text: str):
    tables = []
    current_table = []

    # Iterate through the lines of the markdown text
    for line in markdown_text.split('\n'):
        # Check if the line is part of a table
        if line.startswith('|'):
            current_table.append(line)
        # If the table ends, save it to the tables list
        elif current_table:
            tables.append('\n'.join(current_table))
            current_table = []

    return tables

# Function to display elapsed time while waiting for the API call
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rCommunicating with the API - Elapsed time: {elapsed_
time:.2f} seconds", end="")
        time.sleep(1)

# Get user input
threat_name = input("Enter the name of a cyber threat: ")

api_call_completed = False
elapsed_time_thread = threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()

# Handle exceptions during the API call
try:
    # Generate the report using the OpenAI API
    report = generate_report(threat_name)
    api_call_completed = True
    elapsed_time_thread.join()
```

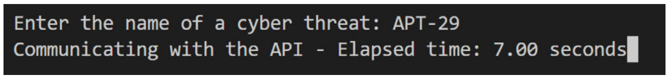
```
except Exception as e:
    api_call_completed = True
    elapsed_time_thread.join()
    print(f"\nAn error occurred during the API call: {e}")
    exit()

# Save the report as a Word document
docx_output_file = f"{threat_name}_report.docx"

# Handle exceptions during the report generation
try:
    with tqdm(total=1, desc="Generating report and files") as pbar:
        markdown_to_docx(report, docx_output_file)
    print("\nReport and tables generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the report generation: {e}")
```

This script uses the **OpenAI API** to generate a cyber threat report as a **Microsoft Word document**.

The crux of this script lies in several key functions. The first function, `generate_report()`, takes in a cyber threat name and uses it as a prompt for the OpenAI API. It returns the generated text from the API response. This text is in Markdown format and is subsequently transformed into a Microsoft Word document by the `markdown_to_docx()` function.

This function parses through the Markdown text line by line, creating tables and headings as required, and finally saves it as a Word document. In parallel, there is an `extract_tables()` function that is designed to locate and extract any tables present within the Markdown text.

To enhance the user experience, the `display_elapsed_time()` function is incorporated. This function tracks and displays the time taken for the API call to complete. It runs in a separate thread, initiated before making the API call:



```
Enter the name of a cyber threat: APT-29
Communicating with the API - Elapsed time: 7.00 seconds
```

Figure 2.6 – Example output of the display_elapsed_time function

The API call itself, as well as the report generation, are wrapped in `try-except` blocks to handle any potential exceptions. Once the report is generated, it is saved as a Word document, with the filename based on the user-inputted cyber threat name.

Upon successful execution of this script, a detailed threat report in Word document format is produced, mimicking the output generated by the ChatGPT web UI. This recipe demonstrates how the OpenAI API can be adapted within a Python script to automate the generation of comprehensive reports.

Hint

You can swap out the **chat-3.5-turbo** model with the **GPT-4** model, if you are a ChatGPT Plus subscriber, for often improved results. Just keep in mind that the GPT-4 model is a bit more expensive than the chat-3.5-turbo model.

You can also improve accuracy and get a more consistent output by lowering the `temperature` value.

GPT-Assisted Vulnerability Scanning

Vulnerability scanning plays a crucial role in identifying and remediating weaknesses before they can be exploited by malicious actors. The tools we use to conduct these scans, such as **NMAP**, **OpenVAS**, or **Nessus**, offer robust functionality but can often be complex and challenging to navigate, especially for those new to the field or unfamiliar with their advanced options.

This is where our recipe comes into play. It leverages the power of ChatGPT to streamline the process of generating command strings for these tools based on user input. With this recipe, you will be able to create precise command strings that can be directly copied and pasted into a CLI to initiate a vulnerability scan, provided the respective tool is installed.

This recipe is not just about saving time; it's about enhancing accuracy, understanding, and effectiveness. It is beneficial for those learning vulnerability assessments, those who are new to these tools, and even seasoned professionals who need a quick reference to ensure their command options are correct. It is especially useful when dealing with advanced options, such as parsing the output or outputting results to files or other formats.

By the end of this recipe, you will be able to generate precise command strings for NMAP, OpenVAS, or Nessus, helping you navigate their functionalities with ease and confidence. Whether you are a cybersecurity beginner or a seasoned expert, this recipe will serve as a valuable tool in your vulnerability assessment arsenal.

Getting ready

Before we begin this recipe, it's essential to ensure that you have properly set up your OpenAI account and obtained your API key. If this hasn't been done yet, you can refer back to *Chapter 1* for detailed instructions. Additionally, you will require the following:

1. **Vulnerability scanning tools**: It's crucial to have NMAP, OpenVAS, or Nessus installed on your system as the recipe generates command strings for these specific tools. Please refer to their official documentation for installation and setup guidelines.
2. **Basic understanding of the tools**: The more familiar you are with NMAP, OpenVAS, or Nessus, the better you will be able to utilize this recipe. If you're new to these tools, consider spending some time understanding their basic functionalities and command-line options.

3. **Command-line environment**: As the recipe generates command strings intended for CLIs, you should have access to a suitable command-line environment where you can run these commands. This could be a terminal in Unix/Linux systems or Command Prompt or PowerShell in Windows.
4. **Sample network configuration data**: Prepare some sample network data that the vulnerability scanning tools can use. This could include IP addresses, hostnames, or other relevant information about the systems you'd like to scan.

How to do it…

In this recipe, we'll show you how to use ChatGPT to create command strings for vulnerability scanning tools such as NMAP, OpenVAS, and Nessus. We'll be providing ChatGPT with the necessary details and using a specific system role and prompt. This will allow you to generate the simplest form of the command necessary to complete your request:

1. Start by logging in to your OpenAI account and go to the ChatGPT web UI.
2. Begin a new conversation with ChatGPT by clicking on the **New chat** button.
3. Next, establish the system's role by entering the following:

```
You are a professional cybersecurity red team specialist and an
expert in penetration testing as well as vulnerability scanning
tools such as NMap, OpenVAS, Nessus, Burpsuite, Metasploit, and
more.
```

Important note

Just as in the *Creating Vulnerability Assessment Plans* recipe, you can enter the role separately using the **OpenAI Playground**, or you can combine it as a single prompt in ChatGPT.

4. Now, prepare your request. This is the information that will replace the `{user_input}` placeholder in the next step. It should be a natural language request such as the following:

```
Use the command line version of OpenVAS to scan my 192.168.20.0
class C network starting by identifying hosts that are up, then
look for running web servers, and then perform a vulnerability
scan of those web servers.
```

5. Once your request is ready, enter the following message text, replacing the `{user_input}` placeholder with your specific request from the previous step:

```
Provide me with the Linux command necessary to complete the
following request:

{user_input}
```

```
Assume I have all the necessary apps, tools, and commands
necessary to complete the request. Provide me with the command
only and do not generate anything further. Do not provide any
explanation. Provide the simplest form of the command possible
unless I ask for special options, considerations, output, etc.
If the request does require a compound command provide all
necessary operators, pipes, etc. as a single one-line command.
Do not provide me with more than one variation or more than one
line.
```

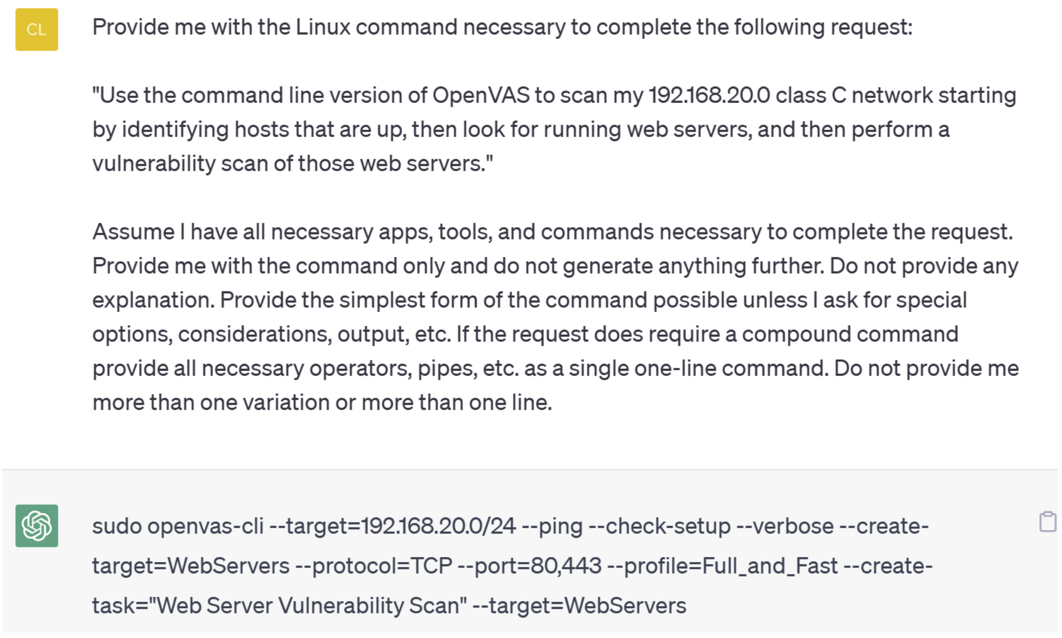
ChatGPT will then generate the command string based on your request. Review the output. If it meets your requirements, you can proceed to copy the command and use it as needed. If it doesn't, you may need to refine your request and try again.

Once you've obtained a satisfactory command, you can copy and paste it directly into your command line to perform the vulnerability scan as described in your request.

Important note

Remember—it's important to review and understand any command before running it in your environment. While ChatGPT aims to provide accurate commands, you are ultimately responsible for ensuring the command's safety and appropriateness for your specific context.

Figure 2.7 shows an example ChatGPT command generated from the prompt used in this recipe:



CL

Provide me with the Linux command necessary to complete the following request:

"Use the command line version of OpenVAS to scan my 192.168.20.0 class C network starting by identifying hosts that are up, then look for running web servers, and then perform a vulnerability scan of those web servers."

Assume I have all necessary apps, tools, and commands necessary to complete the request. Provide me with the command only and do not generate anything further. Do not provide any explanation. Provide the simplest form of the command possible unless I ask for special options, considerations, output, etc. If the request does require a compound command provide all necessary operators, pipes, etc. as a single one-line command. Do not provide me more than one variation or more than one line.

```
sudo openvas-cli --target=192.168.20.0/24 --ping --check-setup --verbose --create-target=WebServers --protocol=TCP --port=80,443 --profile=Full_and_Fast --create-task="Web Server Vulnerability Scan" --target=WebServers
```

Figure 2.7 – Example ChatGPT command generation

How it works...

The GPT-assisted vulnerability scanning recipe taps into the power of NLP and the vast knowledge of ML algorithms to generate accurate and appropriate command strings for vulnerability scanning tools such as NMAP, OpenVAS, and Nessus. When you provide a specific system role and a prompt that represents a user request, ChatGPT uses these inputs to understand the context and generate a response that aligns with the given role:

- **System role definition**: By defining ChatGPT's role as a professional cybersecurity red team specialist and an expert in penetration testing and vulnerability scanning tools, you're instructing the model to answer from a perspective of deep technical understanding and expertise in this field. This context helps in generating accurate and relevant command strings.
- **Natural language prompt**: The natural language prompt that simulates a user request allows ChatGPT to understand the task at hand in a human-like manner. Instead of needing structured data or specific keywords, ChatGPT can interpret the request as a human would and provide a suitable response.
- **Command generation**: With the role and the prompt, ChatGPT generates the Linux command necessary to complete the request. The command is based on the specific details of the user input and the expertise of the assigned role. This is where the AI leverages its knowledge of cybersecurity and language understanding to construct the necessary command string.
- **One-line command**: The specification of providing a one-line command, including all necessary operators and pipes, compels ChatGPT to generate a command that's ready to be pasted into a command line for immediate execution. This removes the need for the user to manually combine or modify the command, saving time and potential errors.
- **Simplicity and clarity**: By asking for the simplest form of the command and without any further explanation, the output is kept clear and concise, which is particularly helpful for those learning or in need of a quick reference.

In summary, the GPT-assisted vulnerability scanning recipe harnesses the power of NLP and ML algorithms to generate precise, ready-to-run commands for vulnerability scanning. By using the defined system role and prompt, users can streamline the process of crafting commands for vulnerability assessments, save time, and improve accuracy.

There's more...

The flexibility and capabilities of this GPT-assisted process extend beyond the example given. First is the versatility of the prompt. It's actually designed to accommodate virtually any request for *any* Linux command across any domain or task. This is a significant advantage as it enables you to leverage ChatGPT's capabilities across a wide range of scenarios. By assigning the role appropriately, such as `"You are a Linux system administrator"`, and substituting your specific request in place of `{user_input}`, you can guide the AI to generate accurate and context-specific command strings for a plethora of Linux operations.

Beyond simply generating command strings, the potential of this recipe is amplified when combined with the OpenAI API and Python. With the proper setup, not only can you generate the necessary Linux commands but you can also automate the execution of these commands. Essentially, this could turn ChatGPT into an active participant in your command-line operations, potentially saving you significant time and effort. This level of automation represents a substantial step forward in interacting with AI models, turning them into active assistants rather than passive information generators.

In upcoming recipes in this book, we'll delve deeper into command automation. This is just the beginning of the possibilities opened up by the integration of AI with your operating system tasks.

Analyzing Vulnerability Assessment Reports using LangChain

As powerful as ChatGPT and the OpenAI API are, they currently have a significant limitation—the **token window**. This window determines how many characters can be exchanged in a complete message between the user and ChatGPT. Once the token count exceeds this limitation, ChatGPT may lose track of the original context, making the analysis of large bodies of text or documents challenging.

Enter **LangChain**—a framework designed to navigate around this very hurdle. LangChain allows us to embed and vectorize large groups of text.

Important note

Embedding refers to the process of transforming text into numerical vectors that an ML model can understand and process. **Vectorizing**, on the other hand, is a technique to encode non-numeric features as numbers. By converting large bodies of text into vectors, we can enable ChatGPT to access and analyze vast amounts of information, effectively turning the text into a *knowledgebase* that the model can refer to, even if it hasn't been trained on this data previously.

In this recipe, we will leverage the power of LangChain, Python, the OpenAI API, and **Streamlit** (a framework for quickly and easily creating web applications) to analyze voluminous documents such as vulnerability assessment reports, threat reports, standards, and more. With a simple UI for uploading files and crafting prompts, the task of analyzing these documents will be simplified to the point of asking ChatGPT straightforward natural language queries.

Getting ready

Before we start with the recipe, ensure that you have an OpenAI account set up and have obtained your API key. If you haven't done this yet, please revisit *Chapter 1* for the steps. Apart from this, you'll also need the following:

1. **Python libraries**: Ensure that you have the necessary Python libraries installed in your environment. You'll specifically need libraries such as `python-docx`, `langchain`, `streamlit`, and `openai`. You can install these using the `pip install` command as follows:

   ```
   pip install python-docx langchain streamlit openai
   ```

2. **Vulnerability assessment report (or a large document of your choice to be analyzed)**: Prepare a vulnerability assessment report or any other substantial document that you aim to analyze. The document can be in any format as long as you can convert it into a **PDF**.
3. **Access to LangChain documentation**: Throughout this recipe, we will be utilizing LangChain, a relatively new framework. Although we will walk you through the process, having the LangChain documentation handy might be beneficial. You can access it at `https://docs.langchain.com/docs/`.
4. **Streamlit**: We will be using Streamlit, a fast and straightforward way to create web apps for Python scripts. While we will guide you through the basics in this recipe, you may want to explore it on your own. You can learn more about Streamlit at `https://streamlit.io/`.

How to do it…

In this recipe, we'll walk you through the steps needed to create a document analyzer using LangChain, Streamlit, OpenAI, and Python. The application will allow you to upload a PDF document, ask questions about it in natural language, and get responses generated by the language model based on the document's content:

1. **Set up the environment and import required modules**: Start by importing all the required modules. You'll need `dotenv` to load environment variables, `streamlit` to create the web interface, `PyPDF2` to read the PDF files, and various components from `langchain` to handle the language model and text processing:

   ```
   import streamlit as st
   from PyPDF2 import PdfReader
   from langchain.text_splitter import CharacterTextSplitter
   from langchain.embeddings.openai import OpenAIEmbeddings
   from langchain.vectorstores import FAISS
   from langchain.chains.question_answering import load_qa_chain
   from langchain.llms import OpenAI
   from langchain.callbacks import get_openai_callback
   ```

2. **Initialize the Streamlit application**: Set up the Streamlit page and header. This will create a web application with the title `"Document Analyzer"` and a `"What would you like to know about this document?"` header text prompt:

```
def main():
    st.set_page_config(page_title="Document Analyzer")
    st.header("What would you like to know about this
document?")
```

3. **Upload the PDF**: Add a file uploader to the Streamlit application to allow users to upload a PDF document:

```
pdf = st.file_uploader("Upload your PDF", type="pdf")
```

4. **Extract the text from the PDF**: If a PDF is uploaded, read the PDF and extract the text from it:

```
if pdf is not None:
    pdf_reader = PdfReader(pdf)
    text = ""
    for page in pdf_reader.pages:
        text += page.extract_text()
```

5. **Split the text into chunks**: Break down the extracted text into manageable chunks that can be processed by the language model:

```
text_splitter = CharacterTextSplitter(
    separator="\n",
    chunk_size=1000,
    chunk_overlap=200,
    length_function=len
)
chunks = text_splitter.split_text(text)
if not chunks:
    st.write("No text chunks were extracted from the PDF.")
    return
```

6. **Create embeddings**: Use `OpenAIEmbeddings` to create vector representations of the chunks:

```
embeddings = OpenAIEmbeddings()
if not embeddings:
    st.write("No embeddings found.")
    return
knowledge_base = FAISS.from_texts(chunks, embeddings)
```

7. **Ask a question about the PDF**: Show a text input field in the Streamlit application for the user to ask a question about the uploaded PDF:

   ```
   user_question = st.text_input("Ask a question about your PDF:")
   ```

8. **Generate a response**: If the user asks a question, find the chunks that are semantically similar to the question, feed those chunks to the language model, and generate a response:

   ```
   if user_question:
       docs = knowledge_base.similarity_search(user_question)

       llm = OpenAI()
       chain = load_qa_chain(llm, chain_type="stuff")
       with get_openai_callback()
   ```

9. Run the script with Streamlit. Using a command-line terminal, run the following command from the same directory as the script:

   ```
   streamlit run app.py
   ```

10. Open browse to `localhost` using a web browser.

Here is how the completed script should look:

```
import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import CharacterTextSplitter
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chains.question_answering import load_qa_chain
from langchain.llms import OpenAI
from langchain.callbacks import get_openai_callback


def main():
    st.set_page_config(page_title="Ask your PDF")
    st.header("Ask your PDF")

    # upload file
    pdf = st.file_uploader("Upload your PDF", type="pdf")

    # extract the text
    if pdf is not None:
      pdf_reader = PdfReader(pdf)
      text = ""
      for page in pdf_reader.pages:
```

```
          text += page.extract_text()

        # split into chunks
        text_splitter = CharacterTextSplitter(
          separator="\n",
          chunk_size=1000,
          chunk_overlap=200,
          length_function=len
        )
        chunks = text_splitter.split_text(text)

        if not chunks:
              st.write("No text chunks were extracted from the PDF.")
              return

        # create embeddings
        embeddings = OpenAIEmbeddings()

        if not embeddings:
            st.write("No embeddings found.")
            return

        knowledge_base = FAISS.from_texts(chunks, embeddings)

        # show user input
        user_question = st.text_input("Ask a question about your PDF:")
        if user_question:
          docs = knowledge_base.similarity_search(user_question)

          llm = OpenAI()
          chain = load_qa_chain(llm, chain_type="stuff")
          with get_openai_callback() as cb:
            response = chain.run(input_documents=docs, question=user_
question)
            print(cb)

          st.write(response)


if __name__ == '__main__':
    main()
```

The script essentially automates the analysis of large documents, such as vulnerability assessment reports, using the LangChain framework, Python, and OpenAI. It leverages Streamlit to create an intuitive web interface where users can upload a PDF file for analysis.

The uploaded document undergoes a series of operations: it's read and its text is extracted, then split into manageable chunks. These chunks are transformed into vector representations (embeddings) using OpenAI Embeddings, enabling the language model to interpret and process the text semantically. These embeddings are stored in a database (**Facebook AI Similarity Search**, or **FAISS** for short), facilitating efficient similarity searches.

The script then provides an interface for users to ask questions about the uploaded document. Upon receiving a question, it identifies the most semantically relevant chunks of text to the question from the database. These chunks, along with the user's question, are processed by a question-answering chain in LangChain, generating a response that is displayed back to the user.

In essence, this script transforms large, unstructured documents into an interactive knowledge base, enabling users to pose questions and receive AI-generated responses based on the document's content.

How it works…

1. First, the necessary modules are imported. These include the `dotenv` module for loading environment variables, `streamlit` for creating the application's UI, `PyPDF2` for handling PDF documents, and various modules from `langchain` for handling language model tasks.
2. The **Streamlit** application's page configuration is set and a file uploader is created that accepts PDF files. Once a PDF file is uploaded, the application uses `PyPDF2` to read the text of the PDF.
3. The text from the PDF is then split into smaller chunks using LangChain's **CharacterTextSplitter**. This ensures that the text can be processed within the language model's maximum token limit. The chunking parameters—`chunk size`, `overlap`, and `separator`, used to split the text—are specified.
4. Next, OpenAI **Embeddings** from LangChain are used to convert the chunks of text into **vector representations**. This involves encoding the semantic information of the text into a mathematical form that can be processed by the language model. These embeddings are stored in a FAISS database, which allows efficient similarity searching for **high-dimensional vectors**.
5. The application then takes a user input in the form of a question about the PDF. It uses the FAISS database to find the chunks of text that are semantically most similar to the question. These chunks are likely to contain the information needed to answer the question.

6. The chosen chunks of text and the user's question are fed into a question-answering *chain* from LangChain. This chain is loaded with an instance of the OpenAI language model. The chain processes the input documents and the question, using the language model to generate a response.
7. The OpenAI callback is used to capture metadata about the API usage, such as the number of tokens used in the request.
8. Finally, the response from the chain is displayed in the Streamlit application.

This process allows for semantic querying of large documents that exceed the language model's token limit. By splitting the document into smaller chunks and using semantic similarity to find the chunks most relevant to a user's question, the application can provide useful answers even when the entire document can't be processed at once by the language model. This demonstrates one way to overcome the token limit challenge when working with large documents and language models.

There's more…

LangChain is not just a tool for overcoming the token window limitation; it's a comprehensive framework for creating applications that interact intelligently with language models. These applications can connect a language model to other data sources and allow the model to interact with its environment—essentially providing the model with a degree of agency. LangChain offers modular abstractions for the components necessary to work with language models, along with a collection of implementations for these abstractions. Designed for ease of use, these components can be employed whether you're using the full LangChain framework or not.

What's more, LangChain introduces the concept of *chains*—these are combinations of the aforementioned components, assembled in specific ways to accomplish particular use cases. Chains offer a high-level interface for users to get started with a specific use case easily and are designed to be customizable to cater to a variety of tasks.

In later recipes, we'll demonstrate how to use these features of LangChain to analyze even larger and more complex documents, such as `.csv` files and spreadsheets.

3
Code Analysis and Secure Development

This chapter delves deep into the intricate process of software development, focusing on a key concern in today's digital world: ensuring the security of your software system. With the increasing complexity of technology and ever-evolving threats, it has become crucial to adopt a **Secure Software Development Lifecycle** (**SSDLC**) that integrates security considerations at each stage. Here, we illustrate how the use of AI, specifically the ChatGPT model, can help streamline this process.

You will learn how to apply ChatGPT in planning and outlining a comprehensive SSDLC, taking into account each phase of development from concept creation to maintenance. Emphasizing the importance of security in every step, we show how ChatGPT can be utilized to craft detailed security requirement documents and secure coding guidelines. The chapter elucidates the generation of these deliverables, demonstrating how they can be collated and shared with your development team and stakeholders to promote a shared understanding of the project's security expectations.

The chapter further explores the potential of ChatGPT in the more technical aspects of the SSDLC. We will examine how ChatGPT can help identify potential security vulnerabilities in your code and even generate custom scripts for security testing. This practical application of AI illustrates a blend of proactive and reactive measures to bolster your software's security.

Lastly, we venture into the final stages of the SSDLC—deployment and maintenance. With the importance of clear, concise documentation often overlooked, we illustrate how ChatGPT can be used to generate comprehensive comments and thorough documentation for your code. By the end of this chapter, you will have gained insights into making your software more comprehensible and maintainable for other developers and users, thereby improving the overall lifecycle of your software.

Throughout this chapter, the core theme is leveraging generative AI to create secure, efficient, and maintainable software systems. It showcases the synergy of human expertise and AI, offering you the tools and techniques to harness ChatGPT and the OpenAI API effectively for secure software development.

In this chapter, we will cover the following recipes:

- Secure Software Development Lifecycle (SSDLC Planning (Planning Phase)
- Security Requirement Generation (Requirements Phase)
- Generating Secure Coding Guidelines (Design Phase)
- Analyzing Code for Security Flaws and Generating Custom Security Testing Scripts (Testing Phase)
- Generating Code Comments and Documentation (Deployment/Maintenance Phase)

Technical requirements

For this chapter, you will need a **web browser** and a stable **internet connection** to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account set up and have obtained your API key. If not, revisit *Chapter 1* for details. Basic familiarity with the Python programming language and working with the command line is necessary, as you'll be using **Python 3.x**, which needs to be installed on your system to work with the OpenAI GPT API and create Python scripts. A **code editor** will also be essential for writing and editing Python code and prompt files, as you work through the recipes in this chapter.

The code files for this chapter can be found here: `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

Secure Software Development Lifecycle (SSDLC) Planning (Planning Phase)

In this recipe, you'll use ChatGPT to assist you in crafting an outline for the SSDLC. This recipe is an essential tool for software developers, project managers, security professionals, or anyone involved in creating secure software systems.

Using the foundational skills of ChatGPT introduced in *Chapter 1* and expanded upon in *Chapter 2*, this recipe guides you through the process of formulating a comprehensive **SSDLC** plan. This plan includes various stages such as initial concept development, requirements gathering, system design, coding, testing, deployment, and maintenance. Throughout the process, we'll illustrate how ChatGPT can be used to detail each phase with a keen emphasis on security considerations.

You'll learn how to construct prompts effectively to obtain high-quality, informative outputs about the SSDLC. The techniques demonstrated in the previous chapter, such as enhancing output with templates and formatting output as a table, will be useful here, enabling you to design prompts that generate the desired output format for each SSDLC phase.

This recipe involves using ChatGPT for generating outputs, but you'll also be able to manually compile these outputs into a well-structured, easily understandable SSDLC plan document, which can then be shared with your development team and other stakeholders, facilitating a thorough understanding of the SSDLC planning process.

Getting ready

Before starting this recipe, you should have a good understanding of the use of ChatGPT for prompt generation, as explained in *Chapter 1*. No additional setup is required for this recipe.

With these prerequisites in place, you're now prepared to start planning an Secure Development Lifecycle with the aid of ChatGPT.

How to do it…

Let's begin this recipe by setting up the system role for ChatGPT and then follow the subsequent prompts to create an SSDLC plan for a specific project. For our example, we will use the development of a secure online banking system, but you can change the system type to one that suits your needs:

1. Begin by logging in to your ChatGPT account and navigating to the ChatGPT web UI.
2. Start a new conversation with ChatGPT by clicking the **New chat** button.
3. Enter the following prompt to establish a **system role**:

```
You are an experienced software development manager with
expertise in secure software development and the Secure Software
Development Lifecycle (SSDLC).
```

4. Next, we'll create an **overview of the SSDLC** with the following prompt:

```
Provide a detailed overview of the Secure Software Development
Lifecycle (SSDLC), highlighting the main phases and their
significance.
```

5. **Initiate the planning** by discussing the specific project's initial concept and feasibility. In this example, we are using a banking system (again, change the type of system in the prompt to suit your needs):

```
Considering a project for developing a secure online banking
system, detail the key considerations for the initial concept
and feasibility phase.
```

6. Next, we need to **create the requirements-gathering process** for the specific project with this prompt:

```
Outline a checklist for gathering and analyzing requirements for
the online banking system project during the requirements phase
of the SSDLC.
```

7. **Learn about the design considerations and steps** for the online banking system:

   ```
   Highlight important considerations when designing a secure
   online banking system during the system design phase of the
   SSDLC.
   ```

8. Now we can **delve into the secure coding practices** relevant to our system:

   ```
   Discuss secure coding best practices to follow when developing
   an online banking system during the development phase of the
   SSDLC.
   ```

9. Understanding the key tests that should be conducted is a critical part of the development. Use this prompt to **create a list of tests**:

   ```
   Enumerate the key types of testing that should be conducted on
   an online banking system during the testing phase of the SSDLC.
   ```

10. **Get guidance on best practices** when deploying the online banking system:

    ```
    List some best practices for deploying an online banking system
    during the deployment phase of the SSDLC.
    ```

11. Wrap up by **understanding the activities during the maintenance phase** of the online banking system:

    ```
    Describe the main activities during the maintenance phase of an
    online banking system and how they can be managed effectively.
    ```

Each prompt will result in an output from ChatGPT that assists in developing a specific SSDLC plan for a secure system.

How it works…

Throughout this recipe, the prompts are crafted to get the best possible output from ChatGPT. The language is clear and specific, which helps in generating detailed and focused responses. Moreover, by defining a specific project, we guide ChatGPT to provide insights that are concrete and applicable. As a result, ChatGPT provides a thorough guide to planning an SSDLC. Here's a breakdown of how each of the steps work (specifically *steps 3-11*):

1. **System role**: By defining the role of ChatGPT as an experienced software development manager, with expertise in secure software development and SSDLC, we are setting the context for our AI partner. This helps ChatGPT generate responses that are more relevant, precise, and knowledgeable.
2. **Understanding the SSDLC**: This prompt helps readers to gain a comprehensive understanding of the SSDLC. By asking ChatGPT to detail the main phases and their significance, we get a high-level overview of the SSDLC that sets the stage for the subsequent steps.

3. **Initial concept/feasibility**: In this step, we have ChatGPT delve into the specific project's initial concept and feasibility. This helps identify the key considerations at this initial phase, which are critical in setting the direction for the rest of the SSDLC.
4. **Requirements gathering**: The requirements phase of the SSDLC is crucial for the success of any project. By having ChatGPT outline a requirements gathering checklist for our specific project, we are ensuring that all the necessary aspects are covered, which will in turn guide the design and development process.
5. **System design**: Here, ChatGPT outlines the important considerations for the system design phase of the SSDLC, focusing on the specifics of our project. This provides guidance on the important elements that need to be considered during the design of the online banking system.
6. **Coding/development**: By asking ChatGPT to discuss secure coding best practices during the development phase, we get a detailed guide on what practices to adhere to, in order to create a secure code base for the online banking system.
7. **Testing**: In this step, we have ChatGPT enumerate the key types of testing that should be conducted during the testing phase. This ensures that the developed online banking system undergoes thorough testing before it is released.
8. **Deployment**: Deploying a system securely is as important as developing it securely. In this step, ChatGPT lists best practices for the deployment phase, ensuring that the transition from a development to a live environment is smooth and secure.
9. **Maintenance**: Finally, we have ChatGPT describe the main activities during the maintenance phase. This provides insights into how the system should be managed post-deployment, to ensure its continual security and performance.

There's more…

This recipe provides you with a detailed guide for planning an SSDLC for a development project (using an online banking system as an example), but that's just the beginning. There are a couple more things you can do to customize this recipe and deepen your understanding:

1. **Customize for different projects**: The principles outlined in this recipe can be applied to a wide variety of projects beyond online banking systems. You can use the prompts as a base and modify the project specifics to suit different types of software development projects. Just ensure that you provide enough context about the project so that ChatGPT can provide relevant and specific responses.

Hint

You can use the output formatting techniques learned in *Chapter 2* to specify the output formatting you prefer for transferring to formal documentation.

2. **Detailed exploration of each SSDLC phase**: We've covered each phase of the SSDLC at a high level in this recipe. However, you could go deeper into each phase by asking ChatGPT more specific questions. For example, in the system design phase, you could ask ChatGPT to explain different design methodologies or go into more detail on best practices for designing user interfaces or databases.

Remember, the power of ChatGPT lies in its ability to provide detailed, informative responses based on the prompts you give it. So, don't be afraid to experiment with different prompts and questions to extract the most value from them.

Security Requirement Generation (Requirements Phase)

In this recipe, you'll use ChatGPT to assist you in creating a comprehensive set of security requirements for your development project. This is an invaluable guide for software developers, project managers, security professionals, or anyone involved in the creation of secure software systems.

Employing the foundational skills of ChatGPT introduced in *Chapter 1* and expanded upon in *Chapter 2*, this recipe will walk you through the process of generating a detailed list of security requirements. These requirements will be tailored to your specific project and will follow best practices in secure development.

You will learn how to devise effective prompts that elicit high-quality, informative outputs on various security requirements. Techniques introduced in previous chapters, such as enhancing output with templates and formatting output as a table, will prove valuable here, as they will enable you to design prompts that generate the desired output format for each security requirement.

This recipe will not only demonstrate how ChatGPT can be used to generate outputs, but just like the previous recipe, you'll be able to collate these outputs into a comprehensive security requirement document, which can then be shared with your development team and stakeholders, ensuring a clear understanding of the security expectations for the project.

Getting ready

Before starting this recipe, make sure you have a clear understanding of the use of ChatGPT for prompt generation as explained in *Chapter 1*. No additional setup is required for this recipe.

With these prerequisites in place, you're now prepared to start generating security requirements for your development project with the help of ChatGPT.

How to do it…

Let's begin this recipe by setting up the system role for ChatGPT and then following the subsequent prompts to create a comprehensive set of security requirements for a specific project.

For our example, we will use the development of a secure medical record management system:

1. Begin by logging in to your ChatGPT account and navigating to the ChatGPT web UI.
2. Start a new conversation with ChatGPT by clicking the **New chat** button.
3. Enter the following prompt to establish a **system role**:

   ```
   You are an experienced cybersecurity consultant specializing in
   secure software development.
   ```

4. Now, we need to **inform ChatGPT about the project** for which we're generating security requirements:

   ```
   Describe a project for developing a secure medical record
   management system. Include details about the type of software,
   its purpose, intended users, and the environments in which it
   will be deployed.
   ```

5. After we've informed ChatGPT about the project, we'll **ask it to identify potential security threats and vulnerabilities**:

   ```
   Given the project description, list potential security threats
   and vulnerabilities that should be considered.
   ```

6. Now that we've identified potential threats and vulnerabilities, we can **generate security requirements** that directly address these concerns:

   ```
   Based on the identified threats and vulnerabilities, generate
   a list of security requirements that the software must meet to
   mitigate these threats.
   ```

7. In addition to the project-specific security requirements, there are general security best practices that apply to almost all software projects. We will use these to **generate general security requirements based on best practices**:

   ```
   Provide additional security requirements that follow general
   best practices in secure software development, regardless of the
   specific project details.
   ```

8. Lastly, we'll **prioritize these requirements based on their impact on the project**:

   ```
   Prioritize the generated security requirements based on their
   impact on the security of the software and the consequences of
   not meeting them.
   ```

By following these prompts, you will engage ChatGPT in meaningful dialog to develop a comprehensive and prioritized list of security requirements for your specific project. Of course, you can replace the secure medical record management system with the specifics of your own project.

How it works…

The prompts throughout this recipe are designed to be clear, specific, and detailed, guiding ChatGPT to provide insightful, relevant, and comprehensive responses. The specificity of the project in the prompts ensures that the outputs from ChatGPT are not only theoretically sound but are also practically applicable. As such, this recipe provides an extensive guide to generating security requirements with the aid of ChatGPT. Here's a breakdown of how each of the steps works (specifically *steps 3-8*):

1. **System role**: By assigning the role of a cybersecurity consultant to ChatGPT, we are providing it with context. This context helps ChatGPT generate responses that are consistent with the expertise of a security professional.
2. **Project description**: In this step, ChatGPT is given a description of the software project. This is important because the security requirements of a software project are largely determined by the specifics of the project itself, such as its purpose, users, and deployment environments.
3. **Identify threats and vulnerabilities**: The prompts at this stage guide ChatGPT to identify the possible security threats and vulnerabilities of the project. This is a crucial step in generating security requirements, as these requirements will be designed to address the potential threats and vulnerabilities.
4. **Generate project-specific security requirements**: Based on the identified threats and vulnerabilities, ChatGPT generates a list of security requirements specific to the project. These requirements will address the specific issues identified in the project description and threat identification.
5. **Generate general security requirements**: In addition to the project-specific security requirements, some general security principles apply to all software projects. By prompting ChatGPT to provide these, we ensure that we're not only addressing the specific threats identified but also adhering to best practices in secure software development.
6. **Prioritize security requirements**: Finally, ChatGPT is asked to prioritize these requirements. This is important, as resources are often limited and an understanding of which requirements are most critical can guide the allocation of resources and effort.

There's more…

This recipe equips you with a structured approach to generate security requirements for a specific software project using ChatGPT. However, there are numerous avenues to expand and adapt this recipe:

- **Customization for different projects**: The strategy outlined in this recipe can be adapted to a wide variety of projects apart from an online payment gateway. You can tailor the prompts according to the specifics of different types of software development projects. Just make sure to provide enough context about the project for ChatGPT to deliver precise and pertinent responses.

Hint

You can use the output formatting techniques learned in *Chapter 2* to specify the output formatting you prefer for transferring to formal documentation.

- **Detailed analysis of identified threats**: This recipe provides a high-level process of identifying threats and generating security requirements. However, you can dive deeper into each identified threat by prompting ChatGPT with more specific questions, such as the potential impacts of the threat, mitigation strategies, or even exploring real-world instances of such threats.
- **Refining security requirements**: You can enhance the process of generating security requirements by asking ChatGPT to detail each requirement further, considering factors such as risk levels, cost of implementation, and potential trade-offs.

Remember, the power of ChatGPT lies in its capacity to deliver detailed and informative responses based on the prompts it receives. Don't hesitate to experiment with various prompts and questions to maximize the value of ChatGPT in your software development projects.

Generating Secure Coding Guidelines (Design Phase)

In this recipe, you'll harness the power of ChatGPT to create robust secure coding guidelines that are designed to meet your project's specific security requirements. This is an invaluable guide for software developers, project managers, security professionals, or anyone involved in the development of secure software systems.

Leveraging the foundational knowledge of the use of ChatGPT introduced in *Chapter 1* and expanded upon in *Chapter 2*, this recipe takes you through the process of generating detailed secure coding guidelines. These guidelines will be tailored to your particular project and will encapsulate best practices in secure development, such as secure session management, error handling, and input validation.

Throughout this recipe, you'll learn to formulate effective prompts that elicit high-quality, informative outputs related to secure coding practices. Techniques such as enhancing output with templates and formatting output as a table, which were introduced in previous chapters, will come in handy here. They will allow you to design prompts that produce the desired output format for each aspect of secure coding.

Like the previous two recipes, the output from this recipe can be compiled into a comprehensive secure coding guidelines document.

Getting ready

Before diving into this recipe, ensure that you have a solid grasp of using ChatGPT for prompt generation, as explained in *Chapter 1*. No additional setup is required for this recipe.

With these prerequisites in place, you're now ready to embark on the journey of generating secure coding guidelines for your development project, with the assistance of ChatGPT.

How to do it...

In this recipe, we'll set the system role for ChatGPT and subsequently delve into a series of prompts to create a comprehensive set of secure coding guidelines tailored to a specific project. For our practical application, let's consider we're embarking on the development of a secure healthcare application, dealing with sensitive patient data:

1. Begin by logging in to your ChatGPT account and navigating to the ChatGPT web UI.
2. Start a new conversation with ChatGPT by clicking the **New chat** button.
3. Enter the following prompt to establish a **system role**:

   ```
   You are a veteran software engineer with extensive experience in
   secure coding practices, particularly in the healthcare sector.
   ```

4. Next, we'll **gain a general understanding of secure coding specific to our project**:

   ```
   Provide a general overview of secure coding and why it's
   important in healthcare software development.
   ```

5. **Generate language-specific secure coding guidelines**. For our healthcare application, let's assume it is developed in Python:

   ```
   What are the key secure coding practices to follow when
   developing healthcare software in Python?
   ```

6. Next, **request guidelines for secure input validation**, which is crucial in protecting against invalid or harmful data:

   ```
   What guidelines should be followed for secure input validation
   when developing a healthcare application?
   ```

7. Handling errors and exceptions properly can prevent many security vulnerabilities. Let's **request information regarding secure error and exception handling specific to our project**:

   ```
   What are the best practices for secure error and exception
   handling in healthcare software development?
   ```

8. Session management is especially important for applications that handle sensitive data such as patient health records. Let's **ask about secure session management best practices specific to our project**:

   ```
   What are the best practices for secure session management in
   healthcare web application development?
   ```

9. **Ask about secure coding practices in handling database operations**, especially given the sensitive nature of healthcare data:

   ```
   What are the best practices to ensure secure coding when a
   healthcare application interacts with databases?
   ```

10. With healthcare applications often needing to communicate with other systems, network communication security is vital. Let's **gain an insight into secure coding practices for network communication specific to our application**:

    ```
    What secure coding practices should be followed when managing
    network communications in healthcare software development?
    ```

11. Lastly, **ask for guidelines on reviewing and testing code for security**, crucial in identifying any security gaps:

    ```
    What approach should be taken to review code for security issues
    in a healthcare application, and what types of tests should be
    conducted to ensure security?
    ```

Following these prompts with ChatGPT will provide a comprehensive guide for secure coding practices in the context of healthcare software development. As always, remember to adjust these prompts to fit the specifics of your own project or sector.

How it works...

Throughout this recipe, the prompts are carefully constructed to elicit detailed, accurate, and comprehensive secure coding guidelines from ChatGPT. The responses obtained will be specific to the healthcare software development context, providing developers with an invaluable resource for creating secure healthcare applications. This demonstrates the capability of ChatGPT to assist in generating secure coding guidelines based on industry-specific considerations. Here's a breakdown of how each of the steps works (specifically *steps 3-11*).

1. **System role**: By defining the role of ChatGPT as a seasoned software engineer, with a specialization in secure coding practices, particularly in the healthcare sector, we set up the correct context for generating focused, informed, and industry-specific advice.
2. **Understanding secure coding**: This step initiates the conversation by obtaining a high-level overview of secure coding practices. The insights provided by ChatGPT here lay the groundwork for understanding the importance of secure coding, especially in a sensitive domain like healthcare.
3. **Language-specific secure coding**: This prompt invites language-specific secure coding guidelines. As secure coding practices can vary between programming languages, this is essential for developing secure healthcare software in Python.
4. **Input validation**: By requesting guidelines on secure input validation, we ensure that the generated coding guidelines will cover a key aspect of secure coding, that is, protecting against harmful or malformed input data.
5. **Error and exception handling**: Proper error and exception handling is a cornerstone of secure coding. This prompt seeks to draw out the best practices for doing so, aiding in the creation of robust and secure healthcare software.

6. **Secure session management**: This prompt aims to gather information on secure session management, crucial for applications that handle sensitive data, such as patient records in a healthcare application.
7. **Secure coding in database operations**: Secure interaction with databases is a critical aspect of secure coding, particularly in healthcare where data sensitivity is paramount. This prompt targets this area to ensure the produced coding guidelines are comprehensive.
8. **Secure coding in network communications**: By asking about secure coding practices for network communications, the guidelines also cover the safe handling of data during transit, a common area of vulnerability in healthcare software.
9. **Code review and testing for security**: The final prompt ensures that the secure coding guidelines include the process of reviewing and testing the code for security vulnerabilities, an integral part of creating secure software.

There's more...

This recipe provides a useful framework for creating secure coding guidelines specifically for a healthcare software project using Python (which you can customize for any other specific application or project). However, the adaptability of ChatGPT allows for even more customization and deeper understanding:

- **Customize for different projects or languages**: The principles and structure outlined in this recipe can be tailored to a wide array of projects and programming languages. For instance, if you're working on an e-commerce platform using JavaScript, you can adjust the context in the prompts to fit that scenario.
- **Detailed exploration of each secure coding topic**: This recipe provides a broad view of secure coding guidelines. To gain a deeper understanding of any given topic, you could ask ChatGPT more specific questions. For example, for secure input validation, you could inquire about best practices for validating different types of input data, such as emails, URLs, or text fields.

Remember, the power of ChatGPT lies not just in its ability to generate detailed and insightful responses, but also in its flexibility. You're encouraged to experiment with different prompts, contexts, and questions to extract the maximum value from this generative AI tool.

Analyzing Code for Security Flaws and Generating Custom Security Testing Scripts (Testing Phase)

In this recipe, you'll use ChatGPT to identify potential security vulnerabilities in your code and **generate custom scripts for security testing**. This recipe is an invaluable tool for software developers, QA engineers, security engineers, and anyone involved in the process of creating and maintaining secure software systems.

Using the foundational knowledge of ChatGPT and the OpenAI API from previous chapters, this recipe guides you through the process of conducting a preliminary security review of your code and developing targeted security tests. ChatGPT can assist by scrutinizing provided code snippets, identifying potential security flaws, and then helping you create custom testing scripts based on these potential vulnerabilities.

You'll learn to formulate effective prompts that elicit high-quality, insightful responses about potential security issues in your code. The techniques from previous chapters, such as refining output with templates and presenting output in a specific format, will prove useful, allowing you to design prompts that generate the desired output for both code analysis and test script creation.

Furthermore, you'll discover how to use the **OpenAI API and Python** to facilitate the process of **reviewing your code** and **generating testing scripts**. This approach could lead to a more efficient, comprehensive security testing process that can be shared with your development and quality assurance teams.

Getting ready

Before diving into this recipe, ensure that your OpenAI account is set up and you have access to your API key. If you haven't set this up yet or need a refresher, you can refer back to previous chapters.

In addition, you need to have certain Python libraries installed in your development environment. These libraries are essential to successfully run the scripts in this recipe. Here are the libraries and their installation commands:

1. `openai`: This is the official OpenAI API client library, which we will use to interact with the OpenAI API. Install it using the `pip install openai` command.
2. `os`: This is a built-in Python library, so no installation is required. We'll use it to interact with the operating system, specifically to fetch the OpenAI API key from your environment variables.
3. `ast`: This is another built-in Python library. We'll use it to parse our Python source code into an abstract syntax tree, which will allow us to better understand the structure of the code.
4. `NodeVisitor`: This is a helper class from the `ast` library that we'll use to visit the nodes of our abstract syntax tree.
5. `threading`: This is a built-in Python library for multi-threading. We'll use it to create a new thread that displays the elapsed time while we're communicating with the OpenAI API.
6. `time`: This is also a built-in Python library. We'll use it to pause our elapsed time thread for one second in each iteration of its loop.

With these prerequisites fulfilled, you are ready to proceed with generating meaningful comments for your Python scripts and creating comprehensive documentation with the assistance of ChatGPT and the OpenAI API.

How to do it…

In this section, we'll leverage ChatGPT's expertise to identify potential security flaws in simple code snippets. These examples cover common security vulnerabilities, but remember that in a real-world scenario, the code you're analyzing might be much more complex. Here are the steps:

Important note

These are simplified code snippets just for educational purposes. When you're applying this approach to your own code, remember to adapt the prompt to suit the complexity and language of your code. If your code snippet is too large, you may need to break it down into smaller sections to fit within the input limit of ChatGPT.

1. Begin by logging in to your ChatGPT account and navigating to the ChatGPT web UI.
2. Start a new conversation with ChatGPT by clicking the **New chat** button.
3. Enter the following prompt to establish a **system role**:

```
You are a seasoned security engineer with extensive experience
in reviewing code for potential security vulnerabilities.
```

4. **Reviewing a code snippet for SQL injection vulnerability**: Direct ChatGPT to analyze a basic PHP code snippet that interacts with a database and ask it to identify any potential security flaws:

```
Please review the following PHP code snippet that interacts with
a database. Identify any potential security flaws and suggest
fixes:

$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM users WHERE username = '$username' AND
password = '$password'";

$result = mysqli_query($conn, $sql);
```

5. **Reviewing a code snippet for Cross-Site Scripting (XSS) vulnerability**: Now, ask ChatGPT to analyze a basic JavaScript code snippet for potential XSS vulnerabilities:

```
Please review the following JavaScript code snippet for a web
application. Identify any potential security flaws and suggest
fixes:

let userContent = document.location.hash.substring(1);
document.write("<div>" + userContent + "</div>");
```

6. **Reviewing a code snippet for Insecure Direct Object References (IDOR) vulnerability**: Lastly, have ChatGPT analyze a Python code snippet to identify potential IDOR vulnerabilities:

```
Please review the following Python code snippet for a web
application. Identify any potential security flaws and suggest
fixes:

@app.route('/file', methods=['GET'])
def file():
    file_name = request.args.get('file_name')
    return send_from_directory(APP_ROOT, file_name)
```

In the *There's more...* section of this recipe, we'll explore how to use the **OpenAI API** to generate custom scripts for security testing based on the potential security flaws identified by ChatGPT.

How it works…

Throughout the recipe, the prompts are designed to be clear and concise, eliciting detailed and focused responses from ChatGPT. Each step builds on the one before it, leveraging the AI's analytical capabilities to not only identify potential flaws in code but also to suggest solutions and help generate testing scripts. As a result, this recipe provides a comprehensive guide to analyzing code for security flaws and creating custom security testing scripts with the help of ChatGPT. Here's a breakdown of how each of the steps works (specifically *steps 3-6*):

1. **System role**: The system role of ChatGPT is set as a veteran software engineer with experience in secure coding practices. This lays the groundwork for the AI model, preparing it to provide accurate and relevant analysis of code snippets for potential security flaws.
2. **Code analysis for security flaws**: We start by providing a sample code snippet to ChatGPT and asking it to analyze it for potential security vulnerabilities. Here, ChatGPT reviews the code as a seasoned software engineer would, checking for typical security issues such as SQL injection vulnerabilities, weak password management, lack of input validation, and more. This enables us to get an expert review of the code in a short amount of time.
3. **Identifying potential flaws**: After analyzing the code, ChatGPT provides a summary of the potential security flaws it found in the code snippet. This includes the nature of the vulnerability, its potential impact, and the part of the code where the flaw was identified. The specificity of these details allows us to understand the vulnerabilities at a deeper level.
4. **Suggesting fixes for identified flaws**: Once the potential flaws are identified, ChatGPT then proposes possible solutions to fix them. This is a crucial step in secure coding, as it not only helps to improve the existing code but also educates on best practices that could prevent similar issues in future code.

There's more...

You can extend the power and flexibility of this recipe by using the OpenAI API with a Python script to review your source code and generate a testing script. Here's how you can do it:

1. Start by importing the necessary libraries:

```
import openai
from openai import OpenAI
import os
import ast
from ast import NodeVisitor
import threading
import time
```

   Set up the OpenAI API in the same way as we did in the *Setting the OpenAI API key as an Environment Variable* recipe in *Chapter 1*:

```
openai.api_key = os.getenv("OPENAI_API_KEY")
```

2. Define a **Python Abstract Syntax Tree (AST) visitor** to visit each node of the source code:

```
class CodeVisitor(NodeVisitor):
    ...
```

   This class will visit each node of the Python source code. It is a subclass of the `NodeVisitor` class from Python's `ast` module.

3. Define a function to review the source code:

```
def review_code(source_code: str) -> str:
    ...
    return response['choices'][0]['message']['content'].strip()
```

   This function takes a string of Python source code as input and sends it as part of a prompt to the OpenAI API, asking it to identify potential security flaws and provide testing steps. It returns the generated testing steps from the API response.

4. Define a function to convert the generated testing steps into a Python test script:

```
def generate_test_script(testing_steps: str, output_file: str):
    with open(output_file, 'w') as file:
        file.write(testing_steps)
```

   This function takes the generated testing steps and an output filename, then saves the testing steps into the output file as a Python test script.

5. Load the source code from a file and run `CodeVisitor` on it:

```
# Change the name of the file to match your source
with open('source_code.py', 'r') as file:
    source_code = file.read()
    visitor = CodeVisitor()
    visitor.visit(ast.parse(source_code))
```

Important note

Be mindful of the input length and token limit when generating content for each section. If your section content or code is too large, you may need to break it down into smaller parts.

6. Use the OpenAI API to review the code and generate testing steps:

```
testing_steps = review_code(source_code)
```

7. Save the generated testing steps as a Python test script:

```
test_script_output_file = "test_script.py"
generate_test_script(testing_steps, test_script_output_file)
```

8. Display the elapsed time while waiting for the API call:

```
def display_elapsed_time():
    ...
```

This function shows the elapsed time in seconds while waiting for the API call to complete.

Here is how the completed script should look:

```
import openai
from openai import OpenAI
import os
import ast
from ast import NodeVisitor
import threading
import time

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

class CodeVisitor(NodeVisitor):
    def __init__(self):
        self.function_defs = []
    def visit_FunctionDef(self, node):
        self.function_defs.append(node.name)
        self.generic_visit(node)

def review_code(source_code: str) -> str:
    messages = [
        {"role": "system", "content": "You are a seasoned security 
engineer with extensive experience in reviewing code for potential 
security vulnerabilities."},
        {"role": "user", "content": f"Please review the following 
Python code snippet. Identify any potential security flaws and then 
provide testing steps:\n\n{source_code}"}
    ]

    client = OpenAI()

    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
    return response.choices[0].message.content.strip()

def generate_test_script(testing_steps: str, output_file: str):
    with open(output_file, 'w') as file:
        file.write(testing_steps)

def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rCommunicating with the API - Elapsed time: {elapsed_
time:.2f} seconds", end="")
        time.sleep(1)

# Load the source code
# Change the name of the file to match your source
with open('source_code.py', 'r') as file:
    source_code = file.read()

visitor = CodeVisitor()
visitor.visit(ast.parse(source_code))

api_call_completed = False
```

```
elapsed_time_thread = threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()

# Handle exceptions during the API call
try:
    testing_steps = review_code(source_code)
    api_call_completed = True
    elapsed_time_thread.join()
except Exception as e:
    api_call_completed = True
    elapsed_time_thread.join()
    print(f"\nAn error occurred during the API call: {e}")
    exit()

# Save the testing steps as a Python test script
test_script_output_file = "test_script.py"

# Handle exceptions during the test script generation
try:
    generate_test_script(testing_steps, test_script_output_file)
    print("\nTest script generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the test script generation:
{e}")
```

This recipe demonstrates how the OpenAI API can be used within a Python script to automate the process of identifying vulnerabilities in your code and generating testing scripts.

This script kicks off by importing the necessary modules, namely `openai`, `os`, and `docx`. After importing the modules, the OpenAI API is set up using your API key obtained from the environment variables.

Following this, the structure of two types of documents—a design document and a user guide—is defined. These structures are simply lists containing the titles of sections that will eventually constitute the final documents.

The `generate_section_content()` function is defined next, which serves to create content for each section of the documents. It uses ChatGPT, prompted by a statement tailored to generate content for a specified section of the document, given some Python source code. It then returns the response as a string.

The `write_to_word_document()` function follows, utilizing the `Document` class from the `python-docx` library. This function adds a heading for each section title and a paragraph for the content of that section to a specified document.

The source code to be analyzed is then loaded from a file named `source_code.py` with the help of Python's built-in `open()` function.

Now comes the creation of the design document. A new document instance is created, and a loop is used to go over each section title defined in `design_doc_structure`. In each iteration, the loop generates content for the section using the `generate_section_content()` function and writes this content to the design document using the `write_to_word_document()` function.

The process is repeated for the user guide, iterating over the `user_guide_structure` instead.

Finally, the script saves the created documents with the `save()` method from the `Document` class. As a result, you are presented with a design document and a user guide, both of which have been generated by ChatGPT based on the source code provided.

Hint

You can swap out the `gpt-3.5-turbo` model with the `GPT-4` model, if you are a ChatGPT Plus subscriber, often with improved results. Just keep in mind that the `GPT-4` model is a bit more expensive than the `gpt-3.5-turbo` model. You can also improve accuracy and get a more consistent output by lowering the temperature value.

This script will be a powerful tool to add to your arsenal for improving the security of your Python code. By automating the review and testing process, you can ensure more consistent, thorough results, save time, and increase the overall security of your projects.

Generating Code Comments and Documentation (Deployment/Maintenance Phase)

In this recipe, we'll harness the power of ChatGPT to breathe life into our Python script by generating comprehensive comments. As software developers, we recognize that commenting code enhances its readability, clarifies the purpose and function of different code segments, and promotes easier maintenance and debugging. Furthermore, comments serve as vital signposts guiding future developers who may work on or use our code.

In the first part of this recipe, we'll prompt ChatGPT to provide comments for each section of our Python script. To achieve this, we'll present ChatGPT with the role of a proficient software engineer seasoned in crafting meaningful comments for Python code.

In the second part of this recipe, we'll move beyond generating comments to creating in-depth documentation. Here, we'll see how ChatGPT can be harnessed to generate a **design document** and a **user guide** based on the same Python script. These documents, encompassing a wide range of information, from software architecture and function descriptions to installation and usage guides, are invaluable in ensuring that our software is comprehensible and maintainable for other developers and users.

Getting ready

Before diving into this recipe, ensure that your OpenAI account is set up and you have access to your API key. If you haven't set this up yet or need a refresher, you can refer back to previous chapters.

In addition, you need to have certain Python libraries installed in your development environment. These libraries are essential to successfully run the scripts in this recipe. Here are the libraries and their installation commands:

1. `openai`: This is the official OpenAI API client library, which we will use to interact with the OpenAI API. Install it using the command pip install openai.
2. `docx`: This is a Python library for creating Microsoft Word documents. Install it using the command pip install docx.

With these prerequisites fulfilled, you are ready to generate meaningful comments for your Python scripts and create comprehensive documentation with the assistance of ChatGPT and the OpenAI API.

How to do it…

In this section, we'll use ChatGPT to generate comments for the provided Python script. Having comments in your code helps improve its readability, aids in understanding the functionality and purpose of different sections of code, and facilitates maintenance and debugging. Here are the steps:

Important note

Remember to adapt the prompts to suit the complexity and language of your code. If your code snippet is too large, you may need to break it down into smaller sections to fit within the input limit of ChatGPT.

1. **Set Up Environment:** Ensure you have the OpenAI Python package installed in your environment. This is crucial for interacting with the OpenAI API.

```
import openai
from openai import OpenAI
import os
import re
```

2. **Initialize OpenAI Client**: Create an OpenAI client instance and set your API key. This key is necessary for authenticating your requests to the OpenAI API.

```
client = OpenAI()
openai.api_key = os.getenv("OPENAI_API_KEY")
```

3. **Read Source Code**: Open and read the Python source code file you intend to review. Ensure the file is in the same directory as your script or provide the correct path.

```
with open('source_code.py', 'r') as file:
    source_code = file.read()
```

4. **Define Review Function**: Create a function, `review_code`, that takes the source code as input and constructs a request to the OpenAI API, asking it to add meaningful comments to the code.

```
def review_code(source_code: str) -> str:
    print("Reviewing the source code and adding comments.\n")
    messages = [
        {"role": "system", "content": "You are a seasoned
security engineer with extensive experience in reviewing code
for potential security vulnerabilities."},
        {"role": "user", "content": f"Please review the
following Python source code. Recreate it with helpful and
meaningful comments... Souce code:\n\n{source_code}"}
    ]
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
    return response.choices[0].message.content.strip()
```

5. **Invoke Review Function**: Call `review_code` with the read source code to get the reviewed and commented code.

```
reviewed_code = review_code(source_code)
```

6. **Output Reviewed Code**: Write the reviewed code, with comments added, to a new file, ensuring to clean up any formatting introduced by the API response.

```
with open('source_code_commented.py', 'w') as file:
    reviewed_code = re.sub(r'^```.*\n', '', reviewed_code)  #
Cleanup
    reviewed_code = re.sub(r'```$', '', reviewed_code)  #
Cleanup
    file.write(reviewed_code)
```

7. **Completion Message**: Print a message to indicate the completion of the review process and the creation of the commented code file.

```
print("The source code has been reviewed and the comments have
been added to the file source_code_commented.py")


Here's how the complete script should look.


import openai
from openai import OpenAI
import os
import re


client = OpenAI()
openai.api_key = os.getenv("OPENAI_API_KEY")


# open a souce code file to provide a souce code file as the
source_code parameter
with open('source_code.py', 'r') as file:
    source_code = file.read()


def review_code(source_code: str) -> str:
    print(f"Reviewing the source code and adding comments.\n")
    messages = [
        {"role": "system", "content": "You are a seasoned
security engineer with extensive experience in reviewing code
for potential security vulnerabilities."},
        {"role": "user", "content": f"Please review the
following Python source code. Recreate it with helpful and
meaningful comments that will help others identify what the code
does. Be sure to also include comments for code/lines inside of
the functions, where the use/functionality might be more complex
Use the hashtag form of comments and not triple quotes. For
comments inside of a function place the comments at the end of
the corresponding line. For function comments, place them on the
line before the function. Souce code:\n\n{source_code}"}
    ]
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
    return response.choices[0].message.content.strip()

reviewed_code = review_code(source_code)

# Output the reviewed code to a file called source_code_
commented.py
with open('source_code_commented.py', 'w') as file:
    # Remove the initial code block markdown from the response
    reviewed_code = re.sub(r'^```.*\n', '', reviewed_code)
    # Remove the final code block markdown from the response
    reviewed_code = re.sub(r'```$', '', reviewed_code)
    file.write(reviewed_code)

print("The source code has been reviewed and the comments have
been added to the file source_code_commented.py")
```

This script exemplifies a practical application of AI in automating the enhancement of source code documentation. By leveraging the OpenAI API, it adds valuable comments to the code, making it more understandable and easier to maintain, especially for teams and projects where thorough documentation is essential.

How it works…

This script demonstrates how to leverage the OpenAI API to enhance a Python source code file with meaningful comments, thereby improving code readability and maintainability. Each part of the script plays a critical role in achieving this goal:

1. **Library Imports and OpenAI Client Initialization**: The script starts by importing necessary Python libraries: openai for interacting with the OpenAI API, os for accessing environment variables (like the API key), and re for regular expressions used in processing the AI's response. An instance of the **OpenAI** client is created and authenticated using the API key stored in an environment variable. This setup is crucial for making secure requests to the OpenAI service.
2. **Reading the Source Code**: The script reads the content of a Python source code file (`source_code.py`). This file is expected to contain the code that needs comments but doesn't include any comments initially. The script uses Python's built-in file handling to read the file's contents into a string variable.
3. **Reviewing the Code with the OpenAI API**: The `review_code` function is where the core functionality resides. It constructs a prompt that describes the task for the AI model, which includes reviewing the provided source code and adding meaningful comments. The prompt is sent to the OpenAI API using the `chat.completions.create` method, specifying the model to use (`gpt-3.5-turbo`) and other parameters like `max_tokens` to control the length of the generated output. The function returns the AI-generated content, which includes the original source code with added comments.

4. **Writing the Reviewed Code to a New File**: After receiving the commented code from the OpenAI API, the script processes the response to remove any unnecessary formatting (like code block markdown) that may have been included. The cleaned-up, commented code is then written to a new file (`source_code_commented.py`). This step makes the enhanced code available for further review or use.

There's more...

In the *How to do it...* section, we leveraged ChatGPT to generate code comments. This is a valuable step in ensuring that our software is maintainable and understandable by other developers. However, we can take it a step further by using ChatGPT to generate more comprehensive documentation, such as a design document and a user guide. Here are the steps to do this:

1. **Set up the environment**: Similar to the previous section, you'll need to start by importing the necessary modules and setting up the OpenAI API:

```
import openai
from openai import OpenAI
import os
from docx import Document

openai.api_key = os.getenv("OPENAI_API_KEY")
```

2. **Define the structure of the design document and user guide**: The structure of the design document and user guide might look like this:

```
design_doc_structure = [
    "Introduction",
    "Software Architecture",
    "Function Descriptions",
    "Flow Diagrams"
]

user_guide_structure = [
    "Introduction",
    "Installation Guide",
    "Usage Guide",
    "Troubleshooting"
]
```

3. **Generate the content for each section**: We can then use ChatGPT to generate the content for each section. Here's an example of generating the software architecture section of the design document:

```
def generate_section_content(section_title: str, source_code:
str) -> str:
    messages = [
        {"role": "system", "content": f"You are an experienced
software engineer with extensive knowledge in writing {section_
title} sections for design documents."},
        {"role": "user", "content": f"Please generate a
{section_title} section for the following Python code:\n\
n{source_code}"}
    ]
    client = OpenAI()

    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
    return response.choices[0].message.content.strip()
```

Important note

Be mindful of the input length and token limit when generating content for each section. If your section content or code is too large, you may need to break it down into smaller parts.

4. **Load the source code**: We need to load the source code file that the prompt and GPT will reference:

```
with open('source_code.py', 'r') as file:
    source_code = file.read()
```

5. **Write the content to a Word document**: After generating the content, we can write it to a Word document using the `python-docx` library:

```
def write_to_word_document(document: Document, title: str,
content: str):
    document.add_heading(title, level=1)
    document.add_paragraph(content)
```

6. **Repeat the process for each section and document**: We can then repeat the process for each section in both the design document and the user guide. Here's an example for creating the design document:

```
design_document = Document()

for section in design_doc_structure:
    section_content = generate_section_content(section, source_
code)
    write_to_word_document(design_document, section, section_
content)

design_document.save('DesignDocument.docx')
```

Here's how the completed code should look:

```
import openai
from openai import OpenAI
import os
from docx import Document

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

# Define the structure of the documents
design_doc_structure = [
    "Introduction",
    "Software Architecture",
    "Function Descriptions",
    "Flow Diagrams"
]

user_guide_structure = [
    "Introduction",
    "Installation Guide",
    "Usage Guide",
    "Troubleshooting"
]

def generate_section_content(section_title: str, source_code: str) ->
str:
    messages = [
        {"role": "system", "content": f"You are an experienced
software engineer with extensive knowledge in writing {section_title}
sections for design documents."},
        {"role": "user", "content": f"Please generate a {section_
title} section for the following Python code:\n\n{source_code}"}
    ]
    client = OpenAI()

    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
    return response.choices[0].message.content.strip()

def write_to_word_document(document: Document, title: str, content:
str):
    document.add_heading(title, level=1)
    document.add_paragraph(content)

# Load the source code
with open('source_code.py', 'r') as file:
    source_code = file.read()

# Create the design document
design_document = Document()

for section in design_doc_structure:
    section_content = generate_section_content(section, source_code)
    write_to_word_document(design_document, section, section_content)

design_document.save('DesignDocument.docx')

# Create the user guide
user_guide = Document()

for section in user_guide_structure:
    section_content = generate_section_content(section, source_code)
    write_to_word_document(user_guide, section, section_content)

user_guide.save('UserGuide.docx')
```

The script commences by importing the necessary modules, namely `openai`, `os`, and `docx`. After importing the modules, the OpenAI API is set up using your API key obtained from the environment variables.

Next, the script outlines the structure of the design document and user guide. These structures are simply arrays containing the titles of sections that will make up these final documents.

The `generate_section_content()` function is defined afterward. This function uses ChatGPT, prompted with a message tailored to generate content for a specified section of the document, given some Python source code. It then returns the generated response as a string.

Subsequently, the Python source code to be documented is loaded from a file named `source_code.py` with the help of Python's built-in `open()` function.

Once the source code is loaded, the creation of the design document is initiated. An instance of the `Document` class is created, and a loop is used to iterate over each section title outlined in `design_doc_structure`. In each iteration, the loop generates content for the section using the `generate_section_content()` function and writes this content to the design document with the help of the `write_to_word_document()` function.

The same process is repeated for the user guide, this time iterating over `user_guide_structure`.

Finally, the script saves the created documents by utilizing the `save()` method from the `Document` class. As a result, you will receive a design document and a user guide, both of which are auto-generated by ChatGPT based on the source code provided.

A point to remember is that the input length and token limit when generating content for each section need careful attention. If your section content or code is too large, you might need to break it down into smaller parts.

This script offers a potent tool to streamline your software documentation process. With the help of ChatGPT and OpenAI API, you can automatically generate precise and comprehensive documents that enhance the understandability and maintainability of your Python code.

4 Governance, Risk, and Compliance (GRC)

With the digital landscape becoming more intertwined and complex, managing cybersecurity risks and maintaining compliance has become increasingly challenging. This chapter offers insightful solutions by demonstrating how ChatGPT, harnessed with the power of the OpenAI API, can significantly enhance the efficiency and effectiveness of your cybersecurity infrastructure.

Throughout the chapter, you will discover how to leverage the capabilities of ChatGPT to **generate comprehensive cybersecurity policies**, simplifying the intricate task of policy creation. We will walk you through an innovative approach that allows granular control over each section of the policy document, delivering a robust cybersecurity framework that's tailored to your specific business needs.

Building upon this groundwork, we will then delve into the nuances of **deciphering complex cybersecurity standards**. ChatGPT acts as a guide, breaking down convoluted compliance requirements into manageable, clear steps, thus providing a streamlined path to ensuring standards compliance.

Furthermore, we will explore the critical domain of **cyber risk assessment**, unveiling how automation can revolutionize this vital process. You will gain insights into identifying potential threats, assessing vulnerabilities, and recommending suitable controls, leading to a substantial enhancement in your organization's ability to manage cybersecurity risks.

Following risk assessment, the focus shifts toward **prioritizing these risks effectively**. You will learn how ChatGPT can assist in creating an **objective scoring algorithm** based on various risk-related factors, enabling you to strategically allocate resources to manage the highest-priority risks.

Finally, we will address the essential task of **risk report generation**. Detailed risk assessment reports not only serve as a valuable record of identified risks and mitigation strategies but also ensure clear communication between stakeholders. We will demonstrate how ChatGPT can automate the creation of such reports, saving time and maintaining consistency across all documentation.

In this chapter, we will cover the following recipes:

- Security Policy and Procedure Generation
- ChatGPT-Assisted Cybersecurity Standards Compliance
- Creating a Risk Assessment Process
- ChatGPT-Assisted Risk Ranking and Prioritization
- Building Risk Assessment Reports

Technical requirements

For this chapter, you will need a **web browser** and a stable **internet connection** to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account set up and have obtained your API key. If not, revisit *Chapter 1* for details. Basic familiarity with the Python programming language and working with the command line is necessary, as you'll be using **Python 3.x**, which needs to be installed on your system, for working with the OpenAI GPT API and creating Python scripts. A **code editor** will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter.

The code files for this chapter can be found here: `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

Security Policy and Procedure Generation

In this recipe, you will leverage the capabilitiese3 of ChatGPT and the OpenAI API to generate a **comprehensive cybersecurity policy** for your organization. This process is invaluable for IT managers, **chief information security officers** (**CISOs**), and cybersecurity professionals looking to create a robust cybersecurity framework that is tailored to their specific business requirements.

Building upon the knowledge acquired in previous chapters, you will establish the role of ChatGPT as a seasoned cybersecurity professional, specializing in **governance, risk, and compliance** (**GRC**). You'll learn how to generate an organized policy outline using ChatGPT and then iteratively fill in the context for each section using subsequent prompts. This approach enables you to generate comprehensive documents with granular control over each section, despite ChatGPT's token limit and context window.

Additionally, this recipe will provide a walk-through on how to use the OpenAI API and Python to automate the policy generation process, and subsequently, generate a cybersecurity policy as a Microsoft Word document. This step-by-step guide will provide a practical framework for producing detailed and tailored cybersecurity policies using ChatGPT and the OpenAI API.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need to confirm you have the following Python libraries installed:

1. `openai`: This library enables you to interact with the OpenAI API. Install it using the `pip install openai` command.
2. `os`: This is a built-in Python library, which allows you to interact with the operating system, especially for accessing environment variables.
3. `docx`: This library is used to generate Microsoft Word documents. Install it with `pip install python-docx`.
4. `markdown`: This library is used to convert Markdown to HTML, which is useful for generating formatted documents. Install it with `pip install markdown`.
5. `tqdm`: This library is utilized for showing progress bars during the policy generation process. Install it with `pip install tqdm`.

Once you have verified that all these requirements are met, you are ready to get started on generating a cybersecurity policy with ChatGPT and the OpenAI API.

How to do it…

In this section, we will guide you through the process of using ChatGPT to generate a detailed cybersecurity policy that aligns with your organization's needs. By providing the necessary details and using the given system role and prompts, you will be able to generate a well-structured cybersecurity policy document:

1. Start by logging in to your OpenAI account and navigating to the ChatGPT web UI.
2. Initiate a new conversation with ChatGPT by clicking the **New chat** button.
3. Enter the following system role to set the context for ChatGPT:

   ```
   You are a cybersecurity professional specializing in governance,
   risk, and compliance (GRC) with more than 25 years of
   experience.
   ```

4. Then, enter the following message text, replacing the placeholders in the `{ }` brackets with relevant information based on your organization's needs. You can either combine this prompt with the system role or enter it separately as follows (replacing the company name and type with your own):

   ```
   Write a detailed cybersecurity policy outline for my company,
   {company name}, which is credit union. Provide the outline only,
   with no context or narrative. Use markdown language to denote
   the proper headings, lists, formatting, etc.
   ```

5. Review the output from ChatGPT. If it is satisfactory and aligns with your requirements, you can proceed to the next step. If not, you have the option to refine your prompt or run the conversation again to generate a different output.
6. Generate the policy from the outline. For each section of the outline, prompt ChatGPT with the following, replacing `{section}` with the appropriate section title from the outline:

```
You are currently writing a cybersecurity policy. Write the
narrative, context, and details for the following section
(and only this section): {section}. Use as much detail and
explanation as possible. Do not write anything that should go in
another section of the policy.
```

7. Once you have the desired output, you can copy and paste the generated responses directly into a Word document, or editor of your choice, to create a comprehensive cybersecurity policy document.

How it works…

This GPT-assisted cybersecurity policy creation recipe taps into the power of **natural language processing** (**NLP**) and machine learning algorithms to produce a tailored and comprehensive cybersecurity policy that caters to the needs of your organization. By assuming a specific system role and utilizing a detailed user request as a prompt, ChatGPT is capable of tailoring its output to meet the demands of a cybersecurity professional, tasked with generating a detailed policy. Here's a deeper look into how this process functions:

1. **System role and detailed prompt**: The system role casts ChatGPT as a seasoned cybersecurity professional, specializing in GRC. The prompt, which acts as the user request, is detailed and describes the specifics of the policy outline, from the nature of the company to the requirements of the cybersecurity policy. These inputs provide context and steer ChatGPT's response, ensuring it caters to the complexities and requirements of the policy creation task.
2. **Natural language processing and machine learning**: NLP and machine learning are the foundation of ChatGPT's capabilities. It uses these technologies to comprehend the complexities of the users requests, to learn from the patterns, and to generate a well-structured cybersecurity policy that is detailed, specific, and comprehensive.
3. **Knowledge and language understanding capabilities**: ChatGPT leverages its vast knowledge base and language understanding capabilities to adhere to industry-standard methodologies and best practices. This is crucial in the swiftly evolving realm of cybersecurity, ensuring that the generated cybersecurity policy is current and complies with recognized standards.
4. **Iterative policy generation**: The process of creating the detailed policy from the generated outline involves prompting ChatGPT iteratively for each section of the policy. This allows for more granular control over the content of each section and helps ensure that the policy is well structured and organized.

5. **Streamlining the policy creation process**: The overall benefit of utilizing this GPT-assisted cybersecurity policy creation recipe is that it streamlines the process of creating a comprehensive cybersecurity policy. It reduces the time spent on policy creation and allows for the generation of a professional-grade policy that aligns with industry standards and the specific needs of your organization.

By employing these detailed inputs, you transform ChatGPT into a potentially invaluable tool that can assist in creating an exhaustive, tailored cybersecurity policy. This not only strengthens your cybersecurity posture but also ensures that your resources are effectively employed in safeguarding your organization.

There's more...

Building on the ChatGPT recipe, you can enhance the functionality by using OpenAI's API to not only generate a cybersecurity policy outline but also to fill in the details of each section. This approach is helpful when you want to create detailed documents on-the-fly or generate policies for multiple companies with different requirements.

This Python script incorporates the same idea as our ChatGPT version, but with additional functionality provided by the OpenAI API grants more control and flexibility over the content generation process. We will now discuss the different steps involved in the OpenAI API version of the cybersecurity policy generation recipe:

1. Import the necessary libraries and set up the OpenAI API:

```
import os
import openai
from openai import OpenAI
import docx
from markdown import markdown
from tqdm import tqdm

# get the OpenAI API key from environment variable
openai.api_key = os.getenv('OPENAI_API_KEY')
```

   In this step, we import the required libraries, such as `openai`, `os`, `docx`, `markdown`, and `tqdm`. We set up the OpenAI API by providing the API key.

2. Prepare the initial prompt for the cybersecurity policy outline:

```
# prepare initial prompt
messages=[
    {
        "role": "system",
        "content": "You are a cybersecurity
            professional specializing in governance,
            risk, and compliance (GRC) with more than
            25 years of experience."
    },
    {
        "role": "user",
        "content": "Write a detailed cybersecurity
            policy outline for my company,
            {company name}, which is a credit union.
            Provide the outline only, with no context
            or narrative. Use markdown language to
            denote the proper headings, lists,
            formatting, etc."
    }
]
```

The initial prompt is constructed using a conversation with two roles: `system` and `user`. The `system` message sets the context, informing the AI model about its role as a seasoned cybersecurity professional. The `user` message instructs the AI model to create a cybersecurity policy outline for a credit union, specifying the need for Markdown formatting.

3. Generate the cybersecurity policy outline using the OpenAI API:

```
print("Generating policy outline...")
try:
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
except Exception as e:
    print("An error occurred while connecting to the
        OpenAI API:", e)
    exit(1)

# get outline
outline =
    response.choices[0].message.content.strip()

print(outline + "\n")
```

This section sends the request to the OpenAI API, and upon successful completion, retrieves the generated policy outline.

4. Split the outline into sections and prepare a Word document:

```
# split outline into sections
sections = outline.split("\n\n")

# prepare Word document
doc = docx.Document()
html_text = ""
```

   Here we split the outline into different sections, each containing a Markdown-formatted heading or subheading. We then initialize a new Word document using the `docx.Document()` function.

5. Loop over each section in the outline, generating detailed information:

```
# for each section in the outline
for i, section in tqdm(enumerate(sections, start=1),
total=len(sections), leave=False):
    print(f"\nGenerating details for section {i}...")
```

   Here we loop over each section of the outline. The `tqdm` function is used to display a progress bar.

6. Prepare the prompt for the AI model to generate the detailed information for the current section:

```
    # prepare prompt for detailed info
    messages=[
        {
            "role": "system",
            "content": "You are a cybersecurity
                professional specializing in
                governance, risk, and compliance (GRC)
                with more than 25 years of
                experience."
        },
        {
            "role": "user",
            "content": f"You are currently writing a
                cybersecurity policy. Write the
                narrative, context, and details for
                the following section (and only this
                section): {section}. Use as much
                detail and explanation as possible. Do
                not write anything that should go in
                another section of the policy."
        }
    ]
```

   The prompt for the AI model is prepared, instructing it to generate detailed information for the current section.

7. Generate detailed information for the current section and add it to the Word document:

```
    try:
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=messages,
            max_tokens=2048,
            n=1,
            stop=None,
            temperature=0.7,
        )
    except Exception as e:
        print("An error occurred while connecting to
            the OpenAI API:", e)
        exit(1)

    # get detailed info
    detailed_info =
        response.choices[0].message.content.strip()

    # convert markdown to Word formatting
    doc.add_paragraph(detailed_info)
    doc.add_paragraph("\n")  # add extra line break
                               for readability

    # convert markdown to HTML and add to the
      html_text string
    html_text += markdown(detailed_info)
```

Here we generate detailed information for the current section using the OpenAI API. The Markdown-formatted text is converted to Word formatting and added to the Word document. It is also converted to HTML and added to the `html_text` string.

8. Save the current state of the Word and HTML documents:

```
    # save Word document
    print("Saving sections...")
    doc.save("Cybersecurity_Policy.docx")

    # save HTML document
    with open("Cybersecurity_Policy.html", 'w') as f:
        f.write(html_text)
```

The current state of the Word document and the HTML document is saved after each section is processed. This ensures that you do not lose any progress if the script is interrupted.

9. Print a completion message after all sections have been processed:

```
print("\nDone.")
```

Here's how the completed script should look:

```
import os
import openai
from openai import OpenAI
import docx
from markdown import markdown
from tqdm import tqdm

# get the OpenAI API key from environment variable
openai.api_key = os.getenv('OPENAI_API_KEY')

# prepare initial prompt
messages=[
    {
        "role": "system",
        "content": "You are a cybersecurity professional
            specializing in governance, risk, and
            compliance (GRC) with more than 25 years of
            experience."
    },
    {
        "role": "user",
        "content": "Write a detailed cybersecurity policy
            outline for my company, XYZ Corp., which is a
            credit union. Provide the outline only, with no
            context or narrative. Use markdown language to
            denote the proper headings, lists, formatting,
            etc."
    }
]

print("Generating policy outline...")
try:
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
except Exception as e:
    print("An error occurred while connecting to the OpenAI
        API:", e)
    exit(1)

# get outline
outline =
    response.choices[0].message.content.strip()

print(outline + "\n")

# split outline into sections
sections = outline.split("\n\n")

# prepare Word document
doc = docx.Document()
html_text = ""

# for each section in the outline
for i, section in tqdm(enumerate(sections, start=1),
total=len(sections), leave=False):
    print(f"\nGenerating details for section {i}...")

    # prepare prompt for detailed info
    messages=[
        {
            "role": "system",
            "content": "You are a cybersecurity
                professional specializing in governance,
                risk, and compliance (GRC) with more than
                25 years of experience."
        },
        {
            "role": "user",
            "content": f"You are currently writing a
                cybersecurity policy. Write the narrative,
                context, and details for the following
                section (and only this section): {section}.
                Use as much detail and explanation as
                possible. Do not write anything that should
                go in another section of the policy."
        }
    ]

    try:
        response = client.chat.completions.createcreate(
            model="gpt-3.5-turbo",
            messages=messages,
            max_tokens=2048,
            n=1,
            stop=None,
            temperature=0.7,
        )
    except Exception as e:
        print("An error occurred while connecting to the
            OpenAI API:", e)
        exit(1)

    # get detailed info
    detailed_info =
        response.choices[0].message.content.strip()

    # convert markdown to Word formatting
    doc.add_paragraph(detailed_info)
    doc.add_paragraph("\n")  # add extra line break for
                               readability

    # convert markdown to HTML and add to the html_text
      string
    html_text += markdown(detailed_info)

    # save Word document
    print("Saving sections...")
    doc.save("Cybersecurity_Policy.docx")

    # save HTML document
    with open("Cybersecurity_Policy.html", 'w') as f:
        f.write(html_text)

print("\nDone.")
```

This Python script automates the process of generating a detailed cybersecurity policy outline for a specific company, XYZ Corp., a credit union. The script initiates by importing the necessary libraries, setting the OpenAI API key, and preparing the initial prompt for the AI model, instructing it to generate the policy outline.

On receiving a successful response from the OpenAI API, the script prints out the policy outline and breaks it down into separate sections for further detailing. A Word document is then initiated to record these details. The script then iterates over each section of the policy outline, generating and appending detailed information from the OpenAI API to the Word document and to an HTML string, effectively creating a detailed policy document in both Word and HTML formats.

After each iteration, the script ensures the documents are saved, providing a safety net against potential data loss due to interruptions. Once all sections are covered and the documents are saved, the script signifies its successful completion. Thus, a high-level policy outline is expanded into a detailed, comprehensive cybersecurity policy, in a process fully automated using the OpenAI API and Python.

ChatGPT-Assisted Cybersecurity Standards Compliance

In this recipe, we will guide you on how to use ChatGPT to assist with **cybersecurity standards compliance**. This recipe builds on the skills gained from previous chapters. Understanding the requirements of cybersecurity standards can be complex, due to the manner in which they are typically written. With ChatGPT, you can simplify this task. By prompting ChatGPT with excerpts from a cybersecurity standard, the model can assist in breaking down these requirements into simpler terms, helping you determine whether you comply, and what steps need to be taken to become compliant if you are not.

Getting ready

Ensure you have access to the ChatGPT interface by logging in to your OpenAI account. Have a cybersecurity standard document on hand from which you can quote excerpts.

How to do it...

To utilize ChatGPT for understanding and checking compliance with cybersecurity standards, follow these steps:

1. Log in to the ChatGPT interface.
2. Assign a role to ChatGPT with the following prompt:

   ```
   You are a cybersecurity professional and CISO with 30 years of
   experience in the industrial cybersecurity industry.
   ```

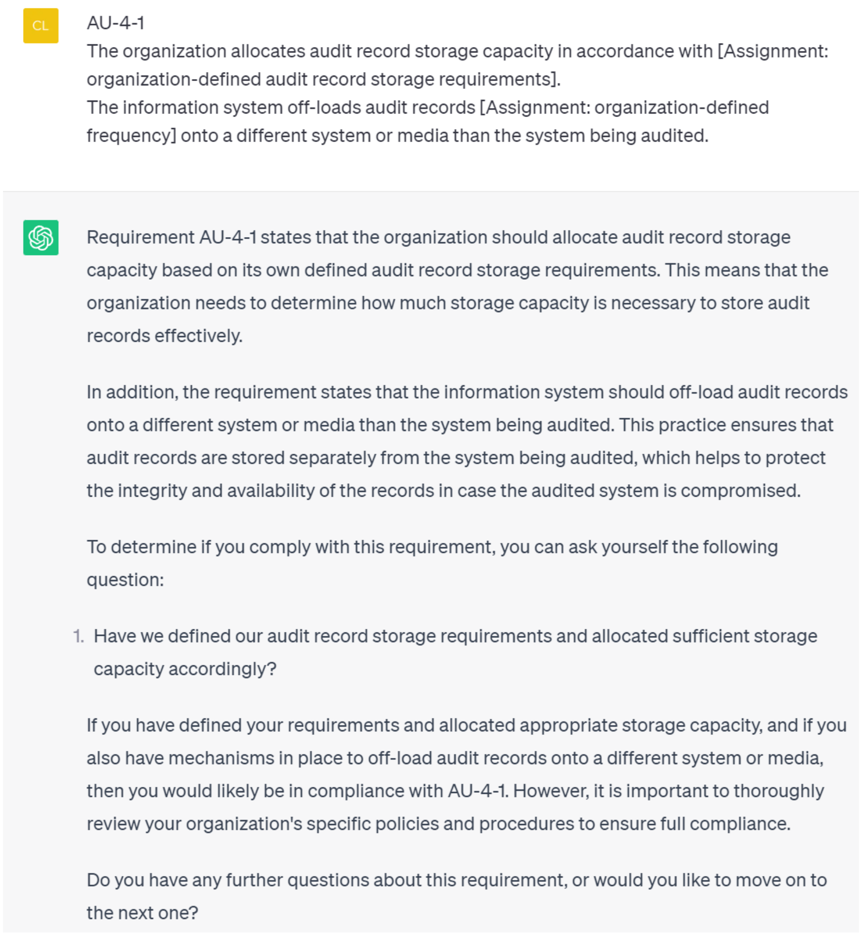
   You should replace `industrial` with whichever industry you're in.
3. Then provide ChatGPT with your prompt:

   ```
   "I need your help understanding the requirements of the NIST
   SP 800-82 revision 2 standard. I will give you the requirement
   ID, specifications, and any other supplemental information
   I have that is associated with the requirement. You will
   then explain the requirement to me in way that is easier to
   ```

```
understand, and form a question based on the requirement to
help me determine whether or not I comply with that requirement
or not. You will follow up by asking me if I have any further
questions about that requirement or if I'm ready to move to
the next requirement. If I have another question regarding
that requirement, we will continue discussing the requirement
in this manner. If I tell you I'm ready to move on to the next
requirement, we will start this process again with the next
requirement."
```

Remember to replace `'NIST SP 800-82 revision 2 standard'` with whichever cybersecurity standard you are working with.

4. Provide ChatGPT with the first requirement ID, specifications, and any supplemental information:



CL

AU-4-1
The organization allocates audit record storage capacity in accordance with [Assignment: organization-defined audit record storage requirements].
The information system off-loads audit records [Assignment: organization-defined frequency] onto a different system or media than the system being audited.

Requirement AU-4-1 states that the organization should allocate audit record storage capacity based on its own defined audit record storage requirements. This means that the organization needs to determine how much storage capacity is necessary to store audit records effectively.

In addition, the requirement states that the information system should off-load audit records onto a different system or media than the system being audited. This practice ensures that audit records are stored separately from the system being audited, which helps to protect the integrity and availability of the records in case the audited system is compromised.

To determine if you comply with this requirement, you can ask yourself the following question:

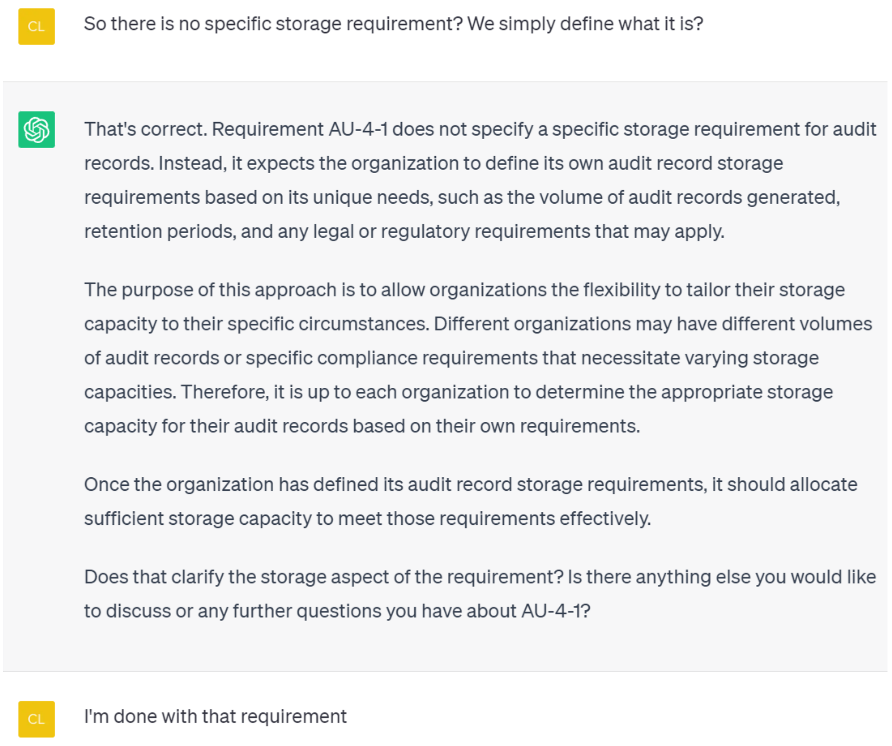
1. Have we defined our audit record storage requirements and allocated sufficient storage capacity accordingly?

If you have defined your requirements and allocated appropriate storage capacity, and if you also have mechanisms in place to off-load audit records onto a different system or media, then you would likely be in compliance with AU-4-1. However, it is important to thoroughly review your organization's specific policies and procedures to ensure full compliance.

Do you have any further questions about this requirement, or would you like to move on to the next one?

Figure 4.1 – Example of a ChatGPT response to a standards requirement query

5. Engage with ChatGPT in a conversation based on the responses it provides, either delving deeper into a particular requirement or moving on to the next:



So there is no specific storage requirement? We simply define what it is?

That's correct. Requirement AU-4-1 does not specify a specific storage requirement for audit records. Instead, it expects the organization to define its own audit record storage requirements based on its unique needs, such as the volume of audit records generated, retention periods, and any legal or regulatory requirements that may apply.

The purpose of this approach is to allow organizations the flexibility to tailor their storage capacity to their specific circumstances. Different organizations may have different volumes of audit records or specific compliance requirements that necessitate varying storage capacities. Therefore, it is up to each organization to determine the appropriate storage capacity for their audit records based on their own requirements.

Once the organization has defined its audit record storage requirements, it should allocate sufficient storage capacity to meet those requirements effectively.

Does that clarify the storage aspect of the requirement? Is there anything else you would like to discuss or any further questions you have about AU-4-1?

I'm done with that requirement

Figure 4.2 – Example of a ChatGPT response to a standards requirement conversation

How it works…

When you assign a role to ChatGPT, you provide a specific context or persona for the model to work with. This helps the model generate responses that are tailored to the given role, resulting in more accurate, relevant, and detailed content.

In the context of understanding and checking compliance with cybersecurity standards, ChatGPT uses its training data to interpret the excerpts from the standard, breaking them down into simpler, easier-to-understand terms. It then forms a question to assist you in determining whether or not you comply with the standard.

Throughout the process, you maintain a conversation with the model, either diving deeper into a specific requirement or moving on to the next one, based on your requirements.

There's more...

Once you're comfortable with this process, you can extend it to cover different standards across various industries.

Here are some additional points to consider:

- **ChatGPT as a training aid**: You can also use it as a teaching tool, utilizing the simplified explanations provided by ChatGPT to educate others in your organization about the requirements of different cybersecurity standards. Using the model to generate easily understandable interpretations of complex standards can be a useful supplement to more traditional forms of training.
- **Importance of regular check-ins**: Using ChatGPT to understand and check compliance with cybersecurity standards can be most effective when done regularly. Cybersecurity landscapes change quickly, and requirements with which an organization was once compliant could change. Regular check-ins can help keep your organization up to date.
- **Potential limitations**: It's worth noting that while ChatGPT is a powerful tool, it does have limitations. Its responses are based on its training data up until September 2021. Therefore, for very recent standards or those that have been significantly updated since then, its responses might not be completely accurate. It's always important to verify information with the most current version of the standard.

> **Important note**
>
> We will discuss more advanced methods of providing updated documents as a knowledge base later on in this book.

- **Importance of professional guidance**: While this approach can help a great deal in understanding the requirements of cybersecurity standards, it's not a substitute for professional legal or cybersecurity guidance. Compliance with these standards often has legal implications, so professional advice is essential. Always consult with a professional when determining your organization's compliance with any cybersecurity standard.
- **Feedback and iteration**: As with any AI tool, the more you use ChatGPT and the more feedback you provide, the better it can assist you. The feedback loop allows the model to adjust and provide responses better tailored to your needs over time.

Creating a Risk Assessment Process

Cyber risk assessment is an essential part of an organization's **risk management strategy**. This process involves identifying potential threats, assessing vulnerabilities that could be exploited by these threats, evaluating the impact that such an exploitation could have on the organization, and recommending suitable controls to mitigate the risk. Understanding the steps involved in conducting a risk assessment can significantly enhance an organization's ability to manage cybersecurity risks.

In this recipe, we will guide you through creating a cyber risk assessment process using Python and the OpenAI API. By automating the risk assessment process, you can streamline your workflow and make your security operations more efficient. This approach can also provide a standardized format for conducting risk assessments, which can improve consistency across your organization.

Getting ready

Before proceeding with this recipe, you will need the following:

- **Python**. This recipe is compatible with **Python 3.6** or later.
- An **OpenAI API key**. If you don't have one, you can obtain it from the OpenAI website after signing up.
- The **OpenAI** Python library. You can install it using pip: `pip install openai`.
- The Python `docx` library for creating Word documents. You can install it using pip: `pip install python-docx`.
- The Python `tqdm` library for displaying progress. You can install it using pip: `pip install tqdm`.
- The Python `threading` and `os` libraries, which are generally available with Python.
- Familiarity with Python programming and basic cybersecurity concepts.

How to do it…

Let's start creating our risk assessment process by building a script that uses the OpenAI API to generate the content of each section in our risk assessment plan. The script will ask ChatGPT to play the role of a cybersecurity professional specializing in GRC to provide us with a detailed narrative, context, and details for each section of the risk assessment process:

1. Import the necessary libraries:

   ```
   import openai
   from openai import OpenAI
   import os
   from docx import Document
   import threading
   import time
   from datetime import datetime
   from tqdm import tqdm
   ```

   This code block imports all the required libraries for our script: `openai` for interacting with the OpenAI API, `os` for environment variables, `Document` from `docx` for creating Word documents, threading and time for managing the time display during API calls, `datetime` for timestamping our report, and `tqdm` for progress visualization.

2. Set up the OpenAI API key:

```
openai.api_key = os.getenv("OPENAI_API_KEY")
```

This code sets the OpenAI API key, which is stored as an environment variable. This key is required to authenticate our program's requests to the OpenAI API.

3. Determine a unique identifier for the assessment report:

```
current_datetime =
    datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
assessment_name =
    f"Risk_Assessment_Plan_{current_datetime}"
```

We use the current date and time to create a unique name for each assessment report, ensuring that we don't overwrite any previous reports. The name is formatted as `Risk_Assessment_Plan_{current_datetime}`, where `current_datetime` is the exact date and time when the script is run.

4. Define the outline for the risk assessment:

```
# Risk Assessment Outline
risk_assessment_outline = [
    "Define Business Objectives",
    "Asset Discovery/Identification",
    "System Characterization/Classification",
    "Network Diagrams and Data Flow Review",
    "Risk Pre-Screening",
    "Security Policy & Procedures Review",
    "Cybersecurity Standards Selection and Gap
         Assessment/Audit",
    "Vulnerability Assessment",
    "Threat Assessment",
    "Attack Vector Assessment",
    "Risk Scenario Creation (using the Mitre ATT&CK
        Framework)",
    "Validate Findings with Penetration Testing/Red
        Teaming",
    "Risk Analysis (Aggregate Findings & Calculate
        Risk Scores)",
    "Prioritize Risks",
    "Assign Mitigation Methods and Tasks",
    "Create Risk Report",
]
```

Here we define the outline for the risk assessment. The outline contains a list of all the sections to be included in the risk assessment process.

Tip

You can modify the process steps to include whichever sections you feel appropriate, and the model will fill in the context for whichever sections you provide.

5. Implement a function to generate section content using the OpenAI API:

```
def generate_section_content(section: str) -> str:
    # Define the conversation messages
    messages = [
        {
            "role": "system",
            "content": 'You are a cybersecurity
                professional specializing in
                governance, risk, and compliance (GRC)
                with more than 25 years of
                experience.'},
        {
            "role": "user",
            "content": f'You are
                currently writing a cyber risk
                assessment policy. Write the
                narrative, context, and details for
                the following section (and only
                this section): {section}. Use as much
                detail and explanation as possible. Do
                not write anything that should go in
                another section of the policy.'
        },
    ]

    # Call the OpenAI API
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    # Return the generated text
    Return
        response.choices[0].message.content.strip()
```

This function takes as input the title of a section from our risk assessment outline and uses the OpenAI API to generate detailed content for that section.

6. Implement a function to convert Markdown text to a Word document:

```
def markdown_to_docx(markdown_text: str, output_file: str):
    document = Document()

    # Iterate through the lines of the markdown text
    for line in markdown_text.split('\n'):
        # Add headings based on the markdown heading
          levels
        if line.startswith('# '):
            document.add_heading(line[2:], level=1)
        elif line.startswith('## '):
            document.add_heading(line[3:], level=2)
        elif line.startswith('### '):
            document.add_heading(line[4:], level=3)
        elif line.startswith('#### '):
            document.add_heading(line[5:], level=4)
        # Add paragraphs for other text
        else:
            document.add_paragraph(line)

    # Save the Word document
    document.save(output_file)
```

   This function takes the generated Markdown text for each section and the desired output filename as inputs and creates a Word document with the same content.

7. Implement a function to display the elapsed time while waiting for the API call:

```
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f}
            seconds", end="")
        time.sleep(1)
```

   This function is responsible for displaying the elapsed time while waiting for the API call to complete. This is useful for tracking how long the process is taking.

8. Start the process for generating the report:

```
api_call_completed = False
elapsed_time_thread =
    threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()
```

Here we start a separate thread to display the elapsed time. This runs concurrently with the main process that makes the API calls.

9. Iterate through each section in the risk assessment outline, generate the section content, and append it to the report:

```
# Generate the report using the OpenAI API
report = []
pbar = tqdm(total=len(risk_assessment_outline),
    desc="Generating sections")
for section in risk_assessment_outline:
    try:
        # Generate the section content
        content = generate_section_content(section)
        # Append the section content to the report
        report.append(f"## {section}\n{content}")
    except Exception as e:
        print(f"\nAn error occurred during the API
            call: {e}")
        exit()
    pbar.update(1)
```

This block of code loops through each section in our risk assessment outline, generates the content for that section using the OpenAI API, and appends the generated content to our report.

10. Finalize the progress and elapsed time display once all sections have been generated:

```
api_call_completed = True
elapsed_time_thread.join()
pbar.close()
```

The `api_call_completed` variable is set to `True` to indicate that all API calls have been completed. We then stop the elapsed time display thread and close the progress bar to signify that the process has ended.

11. Finally, save the generated report as a Word document:

```
# Save the report as a Word document
docx_output_file = f"{assessment_name}_report.docx"

# Handle exceptions during the report generation
```

```
try:
    markdown_to_docx('\n'.join(report),
        docx_output_file)
    print("\nReport generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the report
        generation: {e}")
```

In this final step, the `markdown_to_docx` function is called with the generated report (in Markdown format) and the desired output filename as arguments to create a Word document. The filename includes a timestamp to ensure it's unique. This process is wrapped in a try-except block to handle any exceptions that might occur during this conversion. If successful, we print out a success message; if an error occurs, we print the exception to help with troubleshooting.

The final script should appear as follows:

```
import openai
from openai import OpenAI
import os
from docx import Document
import threading
import time
from datetime import datetime
from tqdm import tqdm

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")
current_datetime = datetime.now()
    .strftime('%Y-%m-%d_%H-%M-%S')
assessment_name =
    f"Risk_Assessment_Plan_{current_datetime}"

# Risk Assessment Outline
risk_assessment_outline = [
    "Define Business Objectives",
    "Asset Discovery/Identification",
    "System Characterization/Classification",
    "Network Diagrams and Data Flow Review",
    "Risk Pre-Screening",
    "Security Policy & Procedures Review",
    "Cybersecurity Standards Selection and Gap
        Assessment/Audit",
    "Vulnerability Assessment",
    "Threat Assessment",
    "Attack Vector Assessment",
    "Risk Scenario Creation (using the Mitre ATT&CK
        Framework)",
    "Validate Findings with Penetration Testing/Red
        Teaming",
    "Risk Analysis (Aggregate Findings & Calculate Risk
        Scores)",
    "Prioritize Risks",
    "Assign Mitigation Methods and Tasks",
    "Create Risk Report",
]

# Function to generate a section content using the OpenAI
  API
def generate_section_content(section: str) -> str:
    # Define the conversation messages
    messages = [
        {
            "role": "system",
            "content": 'You are a cybersecurity
                professional specializing in governance,
                risk, and compliance (GRC) with more than
                25 years of experience.'
        },
        {
            "role": "user",
            "content": f'You are currently writing a cyber
                risk assessment policy. Write the
                narrative, context, and details for the
                following section (and only this section):
                {section}. Use as much detail and
                explanation as possible.
                Do not write anything that should go in
                another section of the policy.'
        },
    ]

    # Call the OpenAI API
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    # Return the generated text
    return response['choices'][0]['message']['content']
        .strip()

# Function to convert markdown text to a Word document
def markdown_to_docx(markdown_text: str, output_file: str):
    document = Document()

    # Iterate through the lines of the markdown text
    for line in markdown_text.split('\n'):
        # Add headings based on the markdown heading levels
        if line.startswith('# '):
            document.add_heading(line[2:], level=1)
        elif line.startswith('## '):
            document.add_heading(line[3:], level=2)
        elif line.startswith('### '):
            document.add_heading(line[4:], level=3)
        elif line.startswith('#### '):
            document.add_heading(line[5:], level=4)
        # Add paragraphs for other text
        else:
            document.add_paragraph(line)

    # Save the Word document
    document.save(output_file)

# Function to display elapsed time while waiting for the
  API call
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f}
            seconds", end="")
        time.sleep(1)

api_call_completed = False
elapsed_time_thread =
    threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()

# Generate the report using the OpenAI API
report = []
pbar = tqdm(total=len(risk_assessment_outline),
    desc="Generating sections")
for section in risk_assessment_outline:
    try:
        # Generate the section content
        content = generate_section_content(section)
        # Append the section content to the report
        report.append(f"## {section}\n{content}")
    except Exception as e:
        print(f"\nAn error occurred during the API call:
            {e}")
        api_call_completed = True
        exit()
    pbar.update(1)

api_call_completed = True
elapsed_time_thread.join()
pbar.close()

# Save the report as a Word document
docx_output_file = f"{assessment_name}_report.docx"

# Handle exceptions during the report generation
try:
    markdown_to_docx('\n'.join(report), docx_output_file)
    print("\nReport generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the report
        generation: {e}")
```

Now, let's take a look at how it works.

How it works…

The Python script works by interacting with the OpenAI API to generate detailed content for each section of the risk assessment process. The content is generated by simulating a conversation between a user and a system (ChatGPT) where the system roleplays as a cybersecurity professional. The conversation messages provided to the API describe the context, and ChatGPT generates a comprehensive response based on that context.

In the OpenAI chat models, a list of messages is provided, and each message has a role and content. The role can be `system`, `user`, or `assistant`. The `system` role is typically used to set the behavior of `assistant`, and the `user` role is used to instruct `assistant`.

In this script, we first set the system role with the message `'You are a cybersecurity professional specializing in governance, risk, and compliance (GRC) with more than 25 years of experience.'`. This is to inform the model of the context, setting it up to respond as an experienced professional in the field of cybersecurity. The model uses this contextual information to generate responses that are appropriate and specific to the scenario.

The `user` role's message, `'You are currently writing a cyber risk assessment policy. Write the narrative, context, and details for the following section (and only this section): {section}. Use as much detail and explanation as possible. Do not write anything that should go in another section of the policy.'`, serves as the specific prompt for the model. This prompt guides the model to generate a detailed narrative for a specific section of a risk assessment policy. It instructs the model to remain focused on the current section and to not deviate into details that belong in other sections. By doing this, we ensure that the content generated is relevant and precise, adhering to the structure of a risk assessment process.

So, in short, the `system` role sets up the context and expertise of `assistant`, while the `user` role provides a directive task for the assistant to perform. This method helps in obtaining structured and relevant content from the AI.

The script is structured to handle each section of the risk assessment process individually, making a separate API call for each section. It takes advantage of multi-threading to display the elapsed time while the API calls are being processed, giving a sense of progress.

The generated content for each section is appended to a report in Markdown format, which is then converted to a Word document using the Python `docx` library. This creates a well-structured, detailed risk assessment plan that can be used as a starting point for conducting risk assessments in an organization.

There's more…

The risk assessment process created by this recipe is flexible. You can experiment with generating your own risk assessment process by using ChatGPT to write the content for different sections and then plug those outline sections into the script. This allows you to create a risk assessment process that is tailored to your organization's specific needs and risk profile. Remember, the best risk assessment process is one that is continually updated and improved based on feedback and new insights.

ChatGPT-Assisted Risk Ranking and Prioritization

In this recipe, we'll leverage the capabilities of ChatGPT to **prioritize and rank cybersecurity risks** based on the given data. Prioritizing risks in cybersecurity is a crucial task that helps organizations focus their resources where they matter most. With the use of ChatGPT, you can make this task more manageable and objective.

In the given scenario, we have a dataset that includes a range of risk-related factors for different assets or systems. These factors include the type of asset, its criticality rating, the business function it serves, the size and rating of its attack surface, the attack vector rating, and the mitigations and remediations in place.

ChatGPT will assist us in creating a scoring algorithm based on this data to prioritize the risks. The highest-priority risks, as calculated by the scoring algorithm, will be listed at the top of a new table. We'll guide you through the process using sample data, but you can apply the same process to your own data in the future.

Getting ready

Ensure you have access to the ChatGPT interface by logging in to your OpenAI account. You will also need a dataset containing a list of systems and their associated vulnerability and risk-related data. More instructions on what this should entail are included in this recipe.

If you do not have a dataset available, you can use the dataset provided in this recipe, which can be downloaded from `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

How to do it…

To start with risk ranking and prioritization, let's send a detailed prompt to ChatGPT. The prompt should clearly state the task and provide the necessary context and data:

> **Tip**
>
> You can provide any system data you want as long as it is separated or delineated and has header names and discernable values that represent the level of risk, severity, value, and so on of the systems and vulnerabilities, which ChatGPT can use to create the appropriate algorithm.

1. Establish the system role by entering the following prompt:

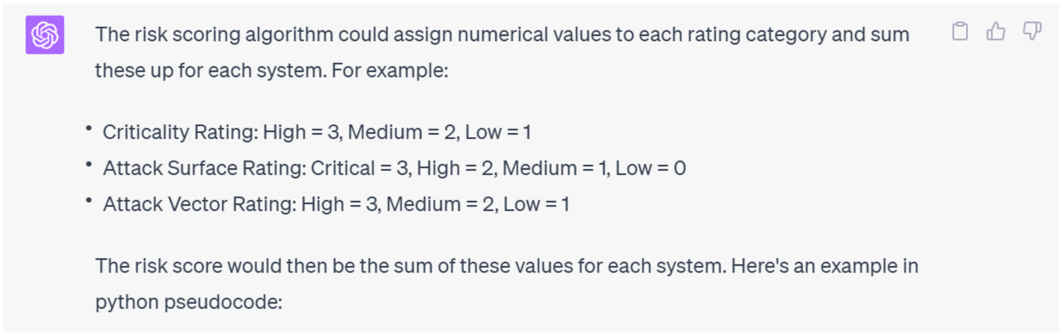
```
You are a cybersecurity professional with 25 years of
experience.
```

2. Instruct ChatGPT to create a scoring algorithm based on your data using the following prompt:

```
Based on the following dataset, categories, and values, create
a suitable risk scoring algorithm to help me prioritize the
risks and mitigation efforts. Provide me with the calculation
algorithm and then create a new table using the same columns,
but now ordered by highest priority to lowest (highest being on
top) and with a new column all the way to the left containing
the row number.

Data:

Asset/System Type    Criticality Rating    Business
Function    Attack Surface Size    Attack Surface
Rating    Attack Vector Rating    Mitigations and Remediations

Web Server
1    High    Sales    120    Critical    High    Firewall
updates, SSL/TLS upgrades

Email
Server    High    Communication    80    High    High    Spam
filter updates, User training

File Server    Medium    HR    30    Medium    Medium    Apply
software patches, Improve password policy

Print Server    Low    All    15    Low    Low    Apply firmware
updates

Database Server
1    High    Sales    200    Critical    High    Update DB
software, Enforce strong access control

Workstation
1    Low    Engineering    10    Low    Low    Install
Antivirus, Apply OS patches

CRM
Software    High    Sales    50    Medium    Medium    Update
CRM software, Implement 2FA

ERP System    High    All    150    Critical    High    Update
ERP software, Implement access control

IoT Device
1    Low    Maintenance    20    Medium    Low    Apply firmware
updates, Change default passwords

Web Server
2    Medium    Marketing    60    Medium    Medium    SSL/TLS
upgrades, Implement WAF

Virtual Machine
1    Low    Development    20    Low    Low    Apply OS patches,
Enforce strong access control

Networking
Switch    High    All    30    Medium    High    Firmware
updates, Change default credentials

Mobile Device
1    Medium    Sales    25    Medium    Medium    Implement MDM,
Enforce device encryption

Firewall    High    All    70    High    High    Firmware
updates, Tighten rule sets
```

```
Cloud Storage    High    All    100    High    High    Apply
cloud security best practices, Implement 2FA
VOIP Server    Medium    All    45    Medium    Medium    Apply
patches, Implement strong access control
Payment
Gateway    High    Sales    180    Critical    High    Implement
PCI DSS, SSL/TLS upgrades
Router    High    All    30    Medium    High    Firmware
updates, Change default credentials
WiFi AP    Medium    All    40    Medium    Medium    Firmware
updates, Implement strong WiFi encryption
SAN Storage    High    All    60    High    High    Update SAN
software, Implement access control
```

The following figure illustrates the scoring algorithm output created by ChatGPT:



The risk scoring algorithm could assign numerical values to each rating category and sum these up for each system. For example:

- Criticality Rating: High = 3, Medium = 2, Low = 1
- Attack Surface Rating: Critical = 3, High = 2, Medium = 1, Low = 0
- Attack Vector Rating: High = 3, Medium = 2, Low = 1

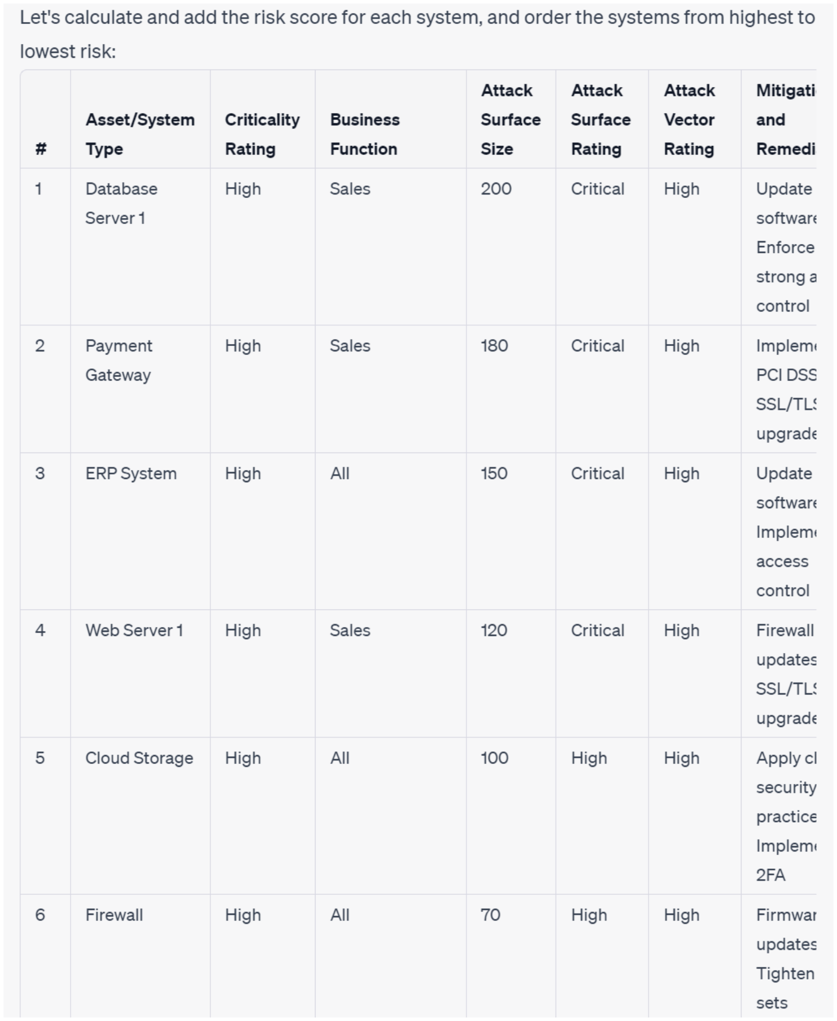
The risk score would then be the sum of these values for each system. Here's an example in python pseudocode:

Figure 4.3 – Example of a scoring algorithm output

The following code is the example scoring algorithm code output generated by ChatGPT:

```
def calculate_risk_score(criticality_rating,
attack_surface_rating, attack_vector_rating):
    ratings = {"High": 3, "Medium": 2, "Low": 1}
    return ratings[criticality_rating] +
        ratings[attack_surface_rating] +
            ratings[attack_vector_rating]
```

The following figure illustrates an example of a formatted output from ChatGPT, which lists the assets, prioritized by the risk score created by the algorithm:



Let's calculate and add the risk score for each system, and order the systems from highest to lowest risk:

#	Asset/System Type	Criticality Rating	Business Function	Attack Surface Size	Attack Surface Rating	Attack Vector Rating	Mitigati and Remedi
1	Database Server 1	High	Sales	200	Critical	High	Update software Enforce strong a control
2	Payment Gateway	High	Sales	180	Critical	High	Impleme PCI DSS SSL/TLS upgrade
3	ERP System	High	All	150	Critical	High	Update software Impleme access control
4	Web Server 1	High	Sales	120	Critical	High	Firewall updates SSL/TLS upgrade
5	Cloud Storage	High	All	100	High	High	Apply cl security practice Impleme 2FA
6	Firewall	High	All	70	High	High	Firmwar updates Tighten sets

Figure 4.4 – Example of a prioritization output

Tip

The data provided in the prompt is tab-delineated. You can provide any system data you want as long as it is separated or delineated and has header names and discernable values that represent the level of risk, severity, value, and so on of the systems and vulnerabilities, which ChatGPT can use to create the appropriate algorithm.

Hint

The sample data used for this recipe was generated with the following prompt:

```
"Generate a table of sample data I will be using for a hypothetical
risk assessment example. The table should be at least 20 rows
and contain the following columns:
Asset/System Type, Criticality Rating, Business Function,
Attack Surface Size (a value that is derived from number of
vulnerabilities found on the system), Attack Surface Rating (a
value that is derived by calculating the number of high and
critical severity ratings compared to the total attack surface),
Attack Vector Rating (a value that is derived by the number of
other systems that have access to this system, with internet
facing being the automatic highest number), list of mitigations
and remediations needed for this system (this would normally
be derived by the vulnerability scan recommendations based on
the findings but for this test/sample data, just make some
hypothetical data up.)"
```

How it works…

ChatGPT is based on a type of machine learning model known as a **transformer**, specifically a variant called **Generative Pretrained Transformer** (**GPT**). This model has been trained on a diverse range of internet text, has learned linguistic patterns and factual information, and has certain reasoning abilities from this vast corpus.

When presented with the task of creating a risk-scoring algorithm, ChatGPT doesn't draw on an inherent understanding of cybersecurity or risk management. Rather, it leverages the patterns it has learned during its training phase. During training, it is likely that it has encountered text related to **risk-scoring algorithms**, **risk prioritization**, and cybersecurity. By recognizing the structure and context of such information in the training data, it can generate relevant and coherent responses when prompted.

When creating a risk scoring algorithm, ChatGPT first understands the various factors presented in the data, such as `Criticality Rating`, `Business Function`, `Attack Surface Size`, `Attack Surface Rating`, `Attack Vector Rating`, and `Mitigations and Remediations`. It understands that these factors are important in determining the overall risk associated with each asset. ChatGPT then formulates an algorithm that takes these factors into account, assigning different weights and scores to each, based on their perceived importance in the overall risk assessment.

The generated algorithm is then applied to the data to score each risk, creating a new table sorted by these scores. This sorting process helps in risk prioritization – risks with higher scores are considered more critical and are listed at the top of the table.

The impressive aspect of ChatGPT is that while it doesn't truly *understand* cybersecurity or risk assessment in the human sense, it can mimic such understanding quite convincingly based on patterns it has learned. Its ability to generate creative and coherent text based on these patterns makes it a versatile tool for a wide range of tasks, including the generation of a risk-scoring algorithm in this recipe.

There's more...

This method is limited by ChatGPT's **token limit**. Due to this limit, only so much data can be pasted. However, later on in this book, we will provide recipes using more advanced techniques on how to get around this limitation.

Hint

Different models have different token limits. If you're an OpenAI Plus subscriber, you can choose between the GPT-3.5 and GPT-4 models. GPT-4 has twice the token limit size as GPT-3.5. Additionally, if you use the OpenAI Playground instead of the ChatGPT UI, you can use the new gpt-3.5-turbo-16k model, which has four times the token limit of GPT-3.5.

Building Risk Assessment Reports

Cybersecurity involves managing and mitigating risks, and an essential part of this process is creating detailed risk assessment reports. These reports not only document the identified risks, vulnerabilities, and threats but also articulate the steps taken to address them, facilitating clear communication with various stakeholders. Automating the creation of risk assessment reports can save significant time and ensure consistency across reports.

In this recipe, we'll create a Python script that uses OpenAI's ChatGPT to automatically generate a cyber risk assessment report. We'll be working with the data provided by the user, with a focus on the data we worked with in the *ChatGPT-assisted risk ranking and prioritization* recipe. However, the script and prompts have been designed to work with any relevant user-provided data. By the end of this recipe, you'll be able to generate detailed and coherent risk assessment reports using Python, ChatGPT, and your own data.

Getting ready

Before you start, ensure that you have the following:

- Python.
- The `openai` Python library installed. You can install it using pip: `pip install openai`.
- The `python-docx` library installed. You can install it using pip: `pip install python-docx`.
- The `tqdm` library installed. You can install it using pip: `pip install tqdm`.
- An API key from OpenAI.

How to do it...

Before we start, remember that you need to provide system data in the `systemdata.txt` file. This data can be anything as long as they are separated or delineated and contain discernable values that represent the level of risk, severity, value, and so on of the systems and vulnerabilities. This information will be used by ChatGPT to create the appropriate algorithm and to generate contextually accurate report sections:

1. Import the required libraries:

```
import openai
from openai import OpenAI
import os
from docx import Document
import threading
import time
from datetime import datetime
from tqdm import tqdm
```

These are the necessary libraries for the script to function correctly. `openai` is used to interact with the OpenAI API, `os` is used to access the environment variables, `Document` from `docx` is used to create a Word document, `threading` and `time` are used for multithreading and to keep track of elapsed time, `datetime` is used to generate a unique filename for each run, and `tqdm` is used to display a progress bar in the console.

2. Set up the OpenAI API key and generate the assessment name:

```
openai.api_key = os.getenv("OPENAI_API_KEY")

current_datetime = datetime.now()
    .strftime('%Y-%m-%d_%H-%M-%S')
assessment_name =
    f"Risk_Assessment_Plan_{current_datetime}"
```

The OpenAI API key is read from an environment variable, and the current date and time are used to create a unique filename for the risk assessment report.

3. Create the risk assessment report outline:

```
risk_assessment_outline = [
    "Executive Summary",
    "Introduction",
    # More sections...
]
```

This is the structure of the risk assessment report, which is used to guide the AI model in generating content for each section.

4. Define the function to generate section content:

```
def generate_section_content(section: str,
system_data: str) -> str:
    messages = [
        {
            "role": "system",
            "content": 'You are a cybersecurity
                professional...'
        },
        {
            "role": "user",
            "content": f'You are currently
                writing a cyber risk assessment
                report...{system_data}'
        },
    ]

    # Call the OpenAI API
client = OpenAI()
response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    Return
        response.choices[0].message.content.strip()
```

This function constructs a conversation prompt, sends it to the OpenAI API, and retrieves the model's response. It accepts the name of a section and system data as arguments and returns the generated content for the specified section.

5. Define the function to convert Markdown text to a Word document:

```
def markdown_to_docx(markdown_text: str, output_file: str):
    document = Document()
    # Parsing and conversion logic...
    document.save(output_file)
```

This function accepts Markdown text and a file path, creates a Word document based on the Markdown content, and saves the document to the specified file path.

6. Define the function to display elapsed time:

```
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f}
            seconds", end="")
        time.sleep(1)
```

   This function is used to display elapsed time in the console while waiting for the API call to complete. It's implemented as a separate thread to allow the main thread to continue executing the rest of the script.

7. Read the system data and start the elapsed time thread:

```
with open("systemdata.txt") as file:
    system_data = file.read()

api_call_completed = False
elapsed_time_thread =
    threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()
```

   The script reads system data from a text file and starts a new thread to display the elapsed time in the console.

8. Generate the report using the OpenAI API:

```
report = []
pbar = tqdm(total=len(risk_assessment_outline),
    desc="Generating sections")
for section in risk_assessment_outline:
    try:
        content = generate_section_content(section,
            system_data)
        report.append(f"## {section}\n{content}")
    except Exception as e:
        print(f"\nAn error occurred during the API
            call: {e}")
        api_call_completed = True
        exit()
    pbar.update(1)

api_call_completed = True
elapsed_time_thread.join()
pbar.close()
```

The script creates a progress bar, iterates over the sections in the risk assessment report outline, generates content for each section using the OpenAI API, and appends the content to the report. It then stops the elapsed time thread and closes the progress bar.

9. Save the report as a Word document:

```
docx_output_file = f"{assessment_name}_report.docx"

try:
    markdown_to_docx('\n'.join(report),
        docx_output_file)
    print("\nReport generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the report
        generation: {e}")
```

Finally, the script converts the generated report from Markdown to a Word document and saves the document. If an exception is thrown during this process, it's caught and a message is printed to the console.

The completed script should appear as follows:

```
import openai
from openai import OpenAI
import os
from docx import Document
import threading
import time
from datetime import datetime
from tqdm import tqdm

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

current_datetime = datetime.now()
    .strftime('%Y-%m-%d_%H-%M-%S')
assessment_name =
    f"Risk_Assessment_Plan_{current_datetime}"

# Cyber Risk Assessment Report Outline
risk_assessment_outline = [
    "Executive Summary",
    "Introduction",
    "Asset Discovery/Identification",
    "System Characterization/Classification",
    "Network Diagrams and Data Flow Review",
    "Risk Pre-Screening",
    "Security Policy & Procedures Review",
    "Cybersecurity Standards Selection and Gap
        Assessment/Audit",
    "Vulnerability Assessment",
    "Threat Assessment",
    "Attack Vector Assessment",
    "Risk Scenario Creation (using the Mitre ATT&CK
        Framework)",
    "Validate Findings with Penetration Testing/Red
        Teaming",
    "Risk Analysis (Aggregate Findings & Calculate Risk
        Scores)",
    "Prioritize Risks",
    "Assign Mitigation Methods and Tasks",
    "Conclusion and Recommendations",
    "Appendix",
]

# Function to generate a section content using the OpenAI
  API
def generate_section_content(section: str, system_data:
str) -> str:
    # Define the conversation messages
    messages = [
        {
            "role": "system",
            "content": 'You are a cybersecurity
                professional specializing in governance,
                risk, and compliance (GRC) with more than
                25 years of experience.'
        },
        {
            "role": "user",
            "content": f'You are currently writing a
                cyber risk assessment report. Write the
                context/details for the following section
                (and only this section): {section}, based
                on the context specific that section, the
                process that was followed, and the
                resulting system data provided below. In
                the absense of user provided context or
                information about the process followed,
                provide placeholder context that aligns
                with industry standard context for that
                section. Use as much detail and explanation
                as possible. Do not write
                anything that should go in another section
                of the policy.\n\n{system_data}'
        },
    ]

    # Call the OpenAI API
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    # Return the generated text
    response.choices[0].message.content.strip()

# Function to convert markdown text to a Word document
def markdown_to_docx(markdown_text: str, output_file: str):
    document = Document()

    # Iterate through the lines of the markdown text
    for line in markdown_text.split('\n'):
        # Add headings based on the markdown heading levels
        if line.startswith('# '):
            document.add_heading(line[2:], level=1)
        elif line.startswith('## '):
            document.add_heading(line[3:], level=2)
        elif line.startswith('### '):
            document.add_heading(line[4:], level=3)
        elif line.startswith('#### '):
            document.add_heading(line[5:], level=4)
        # Add paragraphs for other text
        else:
            document.add_paragraph(line)

    # Save the Word document
    document.save(output_file)

# Function to display elapsed time while waiting for the
```

```
  API call
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f}
            seconds", end="")
        time.sleep(1)

# Read system data from the file
with open("systemdata.txt") as file:
    system_data = file.read()

api_call_completed = False
elapsed_time_thread =
    threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()

# Generate the report using the OpenAI API
report = []
pbar = tqdm(total=len(risk_assessment_outline),
    desc="Generating sections")
for section in risk_assessment_outline:
    try:
        # Generate the section content
        content = generate_section_content(section,
            system_data)
        # Append the section content to the report
        report.append(f"## {section}\n{content}")
    except Exception as e:
        print(f"\nAn error occurred during the API call:
            {e}")
        exit()
    pbar.update(1)

api_call_completed = True
elapsed_time_thread.join()
pbar.close()

# Save the report as a Word document
docx_output_file = f"{assessment_name}_report.docx"

# Handle exceptions during the report generation
```

```
try:
    markdown_to_docx('\n'.join(report), docx_output_file)
    print("\nReport generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the report
        generation: {e}")
```

Now, let's take a look at how it works.

How it works…

The key function of this script is to automate the generation of a detailed risk assessment report based on system data and the assessment process. The script works by dividing the process into a series of defined sections, and in each section, uses the OpenAI API to generate specific, detailed content.

The system data loaded from the file provides context for the gpt-3.5-turbo model to generate the content of each section. We define an outline that breaks down the risk assessment report into various sections that each represent a stage in the risk assessment process. These sections match the steps outlined in the *Creating a Risk Assessment Process* recipe.

We used the following prompt to build a report template prompt in our script:

```
You are a cybersecurity professional and CISO with more than 25 years
of experience. Create a detailed cyber risk assessment report outline
that would be in line with the following risk assessment process
outline:
1. Define Business Objectives
2. Asset Discovery/Identification
3. System Characterization/Classification
4. Network Diagrams and Data Flow Review
5. Risk Pre-Screening
6. Security Policy & Procedures Review
7. Cybersecurity Standards Selection and Gap Assessment/Audit
8. Vulnerability Assessment
9. Threat Assessment
10. Attack Vector Assessment
11. Risk Scenario Creation (using the Mitre ATT&CK Framework)
12. Validate Findings with Penetration Testing/Red Teaming
13. Risk Analysis (Aggregate Findings & Calculate Risk Scores)
14. Prioritize Risks
15. Assign Mitigation Methods and Tasks"
```

This approach guides the model to generate content that matches each section of the report.

In each section, the script calls the `generate_section_content()` function. This function sends a chat message to the OpenAI API that includes the role of the model as a seasoned cybersecurity professional, the task at hand (writing the specified section), and the provided system data. The model's response, which is the content for the specified section, is returned by this function and added to the `report` list.

The `markdown_to_docx()` function converts the Markdown text in the `report` list into a Word document. It does this by iterating over each line in the Markdown text, checking whether it starts with a Markdown heading tag (such as `#`, `##`, etc.), and adding it to the document as a heading or paragraph accordingly.

Once all sections are generated and appended to the `report` list, the list is joined into a single string and converted into a Word document using the `markdown_to_docx()` function.

There's more…

The context for each section that describes certain aspects of the process is placeholder text that can and probably should be modified by the user. We've used this approach for simplicity, but in later recipes, we will demonstrate more advanced techniques on how to provide the actual risk assessment process as real context for the report.

We encourage you to experiment with different assessment process outlines and datasets. Understanding how to tweak the prompts and data to get the most effective results that suit your needs is a crucial part of leveraging AI models such as gpt-3.5-turbo and gpt-4.

Important note

Do remember that, similar to the previous recipe, this method is limited by the chosen model's token limit. The gpt-3.5-turbo model has a token limit of 4,096, which constrains how much data can be passed in from the system data file. However, we will be exploring advanced techniques later in this book to get around this limitation. With these techniques, you'll be able to handle larger datasets and generate more comprehensive reports.

Hint

As with most recipes in this book, the gpt-3.5-turbo model was used for the recipes in this chapter, so that the baseline is set with the most cost-effective model. The GPT-3.5 ChatGPT model was also used so the baseline is set with the most efficient model without limitations. However, you are encouraged to experiment with the use of different models such as gpt-3.5-turbo, gpt-4, and the newly released gpt-3.5-turbo-16k to find the results that best suit your needs.

5
Security Awareness and Training

In this chapter, we will delve into the fascinating realm of **cybersecurity training and education**, highlighting the instrumental role that OpenAI's **large language models** (**LLMs**) can play in enhancing and enriching this critical process. We'll embark on a journey to discover how ChatGPT can be employed as an interactive tool to facilitate various aspects of cybersecurity awareness, from the creation of comprehensive employee training material to the development of interactive cybersecurity assessments, and **even gamifying the learning process itself**.

We'll start by demonstrating how ChatGPT, coupled with Python and the OpenAI API, can be used to **automatically generate content for employee cybersecurity awareness training**. In an era where human error often precipitates security breaches, you'll learn to leverage these powerful tools to create engaging training materials tailored to your organization's specific needs.

As we progress, we'll explore how to create **interactive assessments** with ChatGPT, helping businesses and institutions test their employees understanding and retention of critical cybersecurity concepts. You'll be guided through a hands-on approach to customize these assessments, enabling you to construct a tool that aligns with your organization's existing training content. By the end of this section, you'll have the capability to **generate, export, and integrate these assessments into your learning management systems**.

Continuing our journey, we turn our attention towards **email phishing** - one of the most prevalent tactics employed by cyber criminals. You'll discover how to use ChatGPT to create a tool for interactive email phishing training, thereby fostering a safer cyber environment for your organization. The interactive nature of the training not only ensures a continuous, engaging, and efficient learning experience but also allows for easy integration with live courses or learning management systems.

Next, we'll see how ChatGPT can assist in **preparing for cybersecurity certification exams**. By creating a study guide tailored to certifications like CISSP and others, you'll harness the capabilities of ChatGPT to engage with potential exam questions, gather useful insights, and evaluate your readiness for the exam.

Finally, we explore the exciting and dynamic world of **gamification in cybersecurity** education. As the creator of *ThreatGEN® Red vs. Blue*, one of the world's first educational cybersecurity video games, I believe that the marriage of gaming and education offers a unique and engaging way to impart cybersecurity skills, and the way of the future. With ChatGPT acting as a **game master** in a **cybersecurity-themed role-playing game**, you'll discover how this AI tool can manage game progression, keep score, and provide detailed reports for improvement, adding a whole new dimension to the learning experience.

Through this chapter, you'll not only appreciate the diverse educational applications of ChatGPT but also gain the skills necessary to leverage its capabilities effectively in the realm of cybersecurity.

In this chapter, we will cover the following recipes:

- Developing Security Awareness Training Content
- Assessing Cybersecurity Awareness
- Interactive Email Phishing Training with ChatGPT
- ChatGPT-Guided Cybersecurity Certification Study
- Gamifying Cybersecurity Training

Technical requirement

For this chapter, you will need a **web browser** and a stable **internet connection** to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account setup and have obtained your API key. If not, revisit *Chapter 1* for details. Basic familiarity with the Python programming language and working with the command line is necessary, as you'll be using **Python 3.x**, which needs to be installed on your system, for working with the OpenAI GPT API and creating Python scripts. A **code editor** will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter.

The code files for this chapter can be found here: .

Developing Security Awareness Training Content

In the realm of cybersecurity, employee education is paramount. Human error remains one of the leading causes of security breaches, making it vital to ensure that all members of an organization understand their roles in maintaining cybersecurity. However, creating engaging and effective training materials can be a time-consuming process.

This recipe will guide you in using Python and the OpenAI API to automatically generate content for employee cybersecurity awareness training. The generated content can be utilized for both slide presentations and lecture notes, which you can seamlessly integrate into your chosen slide presentation application.

By leveraging the capabilities of the Python script and API prompt methods, you'll be able to generate a significant volume of content, much more than a single prompt in ChatGPT would typically produce.

The generated training materials in this recipe focus on the electric utility industry, a sector that frequently faces high-stakes cyber threats. However, the techniques used in this recipe are meant to be flexible, allowing you to specify any industry that suits your needs, and the appropriate content will be generated to match your chosen industry. The guidance and procedures developed will be an invaluable resource for educating employees about their roles in maintaining the organization's cybersecurity.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

1. `openai`: This library enables you to interact with the OpenAI API. Install it using the command `pip install openai`.
2. `os`: This is a built-in Python library, which allows you to interact with the operating system, especially for accessing environment variables.
3. `tqdm`: This library is utilized for showing progress bars during the policy generation process. Install it with `pip install tqdm`.

Once these requirements are in place, you are all set to dive into the script.

How to do it…

Important note

Before we begin, it should be noted that the **gpt-4** model is highly recommended for the prompts in this recipe. The **gpt-3.5-turbo** model sometimes provides inconsistent formatting in its output, even after much experimentation with the prompt.

In the following steps, we will guide you through creating a Python script that automates the process of using an initial prompt to generate a list of slides, generating detailed information for each slide, and finally, creating a document with all of the content suitable for copying and pasting directly into the slide presentation app of your choice.

1. **Import Necessary Libraries**. The script begins by importing the required Python libraries, which include **openai** (for OpenAI API calls), **os** (for environment variables), **threading** (for parallel threading), **time** (for time-based functions), **datetime** (for date and time operations), and **tqdm** (for progress bars).

```
import openai
from openai import OpenAI
import os
import threading
import time
from datetime import datetime
from tqdm import tqdm
```

2. **Set Up OpenAI API and Prepare File Output**. Here, we initialize the OpenAI API using your API key. We also prepare the output file where the generated slide content will be stored. The filename is based on the current date and time, ensuring it's unique.

```
# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

current_datetime = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
output_file = f"Cybersecurity_Awareness_Training_{current_
datetime}.txt"
```

3. **Define Helper Functions.** These functions, `content_to_text_file()` and `display_elapsed_time()`, are defined to handle writing slide content into a text file and displaying elapsed time while waiting for the API call, respectively.

```
def content_to_text_file(slide_content: str, file):
    try:
        file.write(f"{slide_content.strip()}\n\n---\n\n")
    except Exception as e:
        print(f"An error occurred while writing the slide
content: {e}")
        return False
    return True

def display_elapsed_time(event):
    start_time = time.time()
    while not event.is_set():
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f} seconds",
end="")
        time.sleep(1)
def display_elapsed_time(event):
    #... function content here...
```

4. **Start the Elapsed Time Tracking Thread.** An Event object is created and a separate thread is started, running the `display_elapsed_time()` function.

```
# Create an Event object
api_call_completed = threading.Event()

# Starting the thread for displaying elapsed time
elapsed_time_thread = threading.Thread(target=display_elapsed_
time, args=(api_call_completed,))
elapsed_time_thread.start()
```

5. **Prepare Initial Prompt.** We set up the initial prompts for the model. The system role describes the AI model's persona, and the user role provides the instruction for the model to generate a cybersecurity training outline.

```
messages=[
    {
        "role": "system",
        "content": "You are a cybersecurity professional with
more than 25 years of experience."
    },
    {
        "role": "user",
        "content": "Create a cybersecurity awareness training
slide list that will be used for a PowerPoint slide based
awareness training course, for company employees, for the
electric utility industry. This should be a single level list
and should not contain subsections or second-level bullets. Each
item should represent a single slide."
    }
]
```

6. **Generate Training Outlinea.** In this step, we make an API call to OpenAI's **gpt-3.5-turbo** model using the `openai.ChatCompletion.create()` function with the prepared prompts to generate a training outline. If any exceptions occur during this process, they are caught and printed to the console.

```
print(f"\nGenerating training outline...")
try:
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
except Exception as e:
    print("An error occurred while connecting to the OpenAI
API:", e)
    exit(1)
```

7. **Retrieve and Print the Training Outline.** After the model generates the training outline, it is extracted from the response and printed to the console for the user to review.

```
response.choices[0].message.content.strip()

print(outline + "\n")
```

8. **Split the Outline into Sections.** We split the outline into individual sections based on line breaks (`\n`). This prepares them for more detailed content generation in the next step.

```
sections = outline.split("\n")
```

9. **Generate Detailed Slide Content.** In this section, the script iterates through each section in the outline and generates detailed slide content for each. It opens the output text file, prepares a new prompt for the model, resets the elapsed time event, calls the model again, retrieves the generated slide content, and writes it to the output file.

```
try:
    with open(output_file, 'w') as file:
        for i, section in tqdm(enumerate(sections, start=1),
total=len(sections), leave=False):
            print(f"\nGenerating details for section {i}...")

            messages=[
                {
                    "role": "system",
                    "content": "You are a cybersecurity
professional with more than 25 years of experience."
                },
                {
                    "role": "user",
                    "content": f"You are currently working on a
PowerPoint presentation that will be used for a cybersecurity
awareness training course, for end users, for the electric
utility industry. The following outline is being used:\n\
n{outline}\n\nCreate a single slide for the following section
(and only this section) of the outline: {section}. The slides
are for the employee's viewing, not the instructor, so use the
appropriate voice and perspective. The employee will be using
these slides as the primary source of information and lecture
for the course. So, include the necessary lecture script in the
speaker notes section. Do not write anything that should go in
another section of the policy. Use the following format:\n\
n[Title]\n\n[Content]\n\n---\n\n[Lecture]"
                }
            ]

            api_call_completed.clear()

            try:
                response = client.chat.completions.create(
                    model="gpt-3.5-turbo",
                    messages=messages,
                    max_tokens=2048,
                    n=1,
                    stop=None,
                    temperature=0.7,
                )
            except Exception as e:
                print("An error occurred while connecting to the
OpenAI API:", e)
                api_call_completed.set()
                exit(1)

            api_call_completed.set()

            slide_content = response.choices[0].message.content.
strip()

            if not content_to_text_file(slide_content, file):
                print("Failed to generate slide content.
Skipping to the next section...")
                continue
```

10. **Handle Successful and Unsuccessful Runs.** If the output text file is generated successfully, a success message is printed to the console. If any exceptions occur during the process, they are caught and the error message is printed.

```
print(f"\nText file '{output_file}' generated successfully!")

except Exception as e:
    print(f"\nAn error occurred while generating the output text
file: {e}")
```

11. **Clean Up Threads** At the end of the script, we signal the `elapsed_time_thread` to stop and join it back to the main process. This ensures no threads are left running unnecessarily.

    ```
    api_call_completed.set()
    elapsed_time_thread.join()
    ```

Here is how the final script should look:

```
import openai
from openai import OpenAI
import os
import threading
import time
from datetime import datetime
from tqdm import tqdm

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

current_datetime = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
output_file = f"Cybersecurity_Awareness_Training_{current_datetime}.
txt"

def content_to_text_file(slide_content: str, file):
    try:
        file.write(f"{slide_content.strip()}\n\n---\n\n")
    except Exception as e:
        print(f"An error occurred while writing the slide content:
{e}")
        return False
    return True

# Function to display elapsed time while waiting for the API call
def display_elapsed_time(event):
    start_time = time.time()
    while not event.is_set():
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f} seconds", end="")
        time.sleep(1)

# Create an Event object
api_call_completed = threading.Event()

# Starting the thread for displaying elapsed time
elapsed_time_thread = threading.Thread(target=display_elapsed_time,
args=(api_call_completed,))
```

```
elapsed_time_thread.start()

# Prepare initial prompt
messages=[
    {
        "role": "system",
        "content": "You are a cybersecurity professional with more
than 25 years of experience."
    },
    {
        "role": "user",
        "content": "Create a cybersecurity awareness training slide
list that will be used for a PowerPoint slide based awareness training
course, for company employees, for the electric utility industry. This
should be a single level list and should not contain subsections or
second-level bullets. Each item should represent a single slide."
    }
]

print(f"\nGenerating training outline...")
try:
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )
except Exception as e:
    print("An error occurred while connecting to the OpenAI API:", e)
    exit(1)

# Get outline
outline = response.choices[0].message.content.strip()

print(outline + "\n")

# Split outline into sections
sections = outline.split("\n")

# Open the output text file
try:
    with open(output_file, 'w') as file:
        # For each section in the outline
        for i, section in tqdm(enumerate(sections, start=1),
total=len(sections), leave=False):
            print(f"\nGenerating details for section {i}...")

            # Prepare prompt for detailed info
            messages=[
                {
                    "role": "system",
                    "content": "You are a cybersecurity professional
with more than 25 years of experience."
                },
                {
                    "role": "user",
                    "content": f"You are currently working on a
PowerPoint presentation that will be used for a cybersecurity
awareness training course, for end users, for the electric utility
industry. The following outline is being used:\n\n{outline}\n\nCreate
a single slide for the following section (and only this section) of
the outline: {section}. The slides are for the employee's viewing,
not the instructor, so use the appropriate voice and perspective.
The employee will be using these slides as the primary source of
information and lecture for the course. So, include the necessary
lecture script in the speaker notes section. Do not write anything
that should go in another section of the policy. Use the following
format:\n\n[Title]\n\n[Content]\n\n---\n\n[Lecture]"
                }
            ]

            # Reset the Event before each API call
            api_call_completed.clear()

            try:
                response = client.chat.completions.create(
                    model="gpt-3.5-turbo",
                    messages=messages,
                    max_tokens=2048,
                    n=1,
                    stop=None,
                    temperature=0.7,
                )
            except Exception as e:
                print("An error occurred while connecting to the
OpenAI API:", e)
                exit(1)

            # Set the Event to signal that the API call is complete
            api_call_completed.set()

            # Get detailed info
            slide_content = response.choices[0].message.content.
strip()

            # Write the slide content to the output text file
            if not content_to_text_file(slide_content, file):
                print("Failed to generate slide content. Skipping to
the next section...")
                continue

    print(f"\nText file '{output_file}' generated successfully!")

except Exception as e:
    print(f"\nAn error occurred while generating the output text file:
{e}")

# At the end of the script, make sure to join the elapsed_time_thread
api_call_completed.set()
elapsed_time_thread.join()
```

The result is a comprehensive cybersecurity awareness training course in a text file, ready for conversion into a PowerPoint presentation.

How it works...

This script leverages the advanced capabilities of the OpenAI models to generate engaging, instructive, and well-structured content for a cybersecurity awareness training course. The process follows several stages:

- **API Initialization**: The script starts by initializing the OpenAI API. It uses the API key to connect with the OpenAI **gpt-3.5-turbo** model, which has been trained on a diverse range of internet text. The model is designed to generate human-like text, making it ideal for creating unique and comprehensive content for training materials.
- **Date-Time Stamping and File Naming**: The script creates a unique timestamp that it appends to the output file name. This ensures each run of the script creates a distinct text file, preventing any overwriting of previous outputs.
- **Function Definitions**: Two helper functions are defined: `content_to_text_file()` and `display_elapsed_time()`. The former is used for writing generated slide content to a text file, with error handling in place. The latter, working with Python's threading capabilities, provides a real-time display of elapsed time during API calls.

- **Outline Generation**: The script constructs a prompt that reflects the requirements of the course and sends this to the API. The API uses its contextual understanding to generate an outline matching these criteria.
- **Outline Segmentation**: After the outline is generated, the script separates it into individual sections. Each section will later be developed into a full-fledged slide.
- **Detailed Content Generation**: For each section in the outline, the script prepares a detailed prompt incorporating the entire outline and the particular section. This is sent to the API, which returns detailed slide content, split into slide content and lecture notes.
- **Writing to File**: Each generated slide content is written to the output file using the `content_to_text_file()` function. If a slide fails to generate, the script skips to the next section without halting the entire process.
- **Thread Management and Exception Handling**: The script includes robust thread management and exception handling to ensure smooth operation. If an error occurs while writing to the output file, the script reports the issue and gracefully shuts down the thread displaying elapsed time.

By employing the OpenAI API and the gpt-3.5-turbo model, this script efficiently generates a structured and comprehensive cybersecurity awareness training course. The course can then be converted into a PowerPoint presentation. The generated content is both engaging and instructive, making it a valuable resource for the target audience.

There's more...

The potential of this script is not limited to text output alone. With some modifications, you could integrate it with the Python library **python-pptx** to generate **Microsoft PowerPoint** presentations directly, thus streamlining the process even further.

At the time of writing, this method is in its development stage and is being actively explored for improvement and refinement. For the adventurous and the curious, you can access the modified script on GitHub at: `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`. This script promises an exciting step forward in automating the creation of cybersecurity training material.

To delve deeper into the workings and capabilities of the **python-pptx** library, which would allow you to generate and manipulate PowerPoint presentations in Python, you can visit its comprehensive documentation at: `https://python-pptx.readthedocs.io/en/latest/`.

As technology advances, the integration of AI and automation with content creation is an evolving landscape with immense potential. This script is just the starting point, and the possibilities for customization and expansion are endless!

Assessing Cybersecurity Awareness

With an increasing number of cyber threats around us, cybersecurity awareness has never been more critical. This recipe will walk you through creating an interactive cybersecurity awareness assessment using ChatGPT. The tool we are building can be a vital instrument for businesses and institutions looking to educate their employees about cybersecurity. The quiz could serve as a follow-up to a cybersecurity awareness training course, testing the employees understanding and retention of the content. Furthermore, the assessment can be customized to match your existing cybersecurity training content, making it highly adaptable to any organization's specific needs.

The most interesting part? At the end of the guide, you will be able to export the assessment questions and an answer key to a text document. This feature allows for easy integration with live courses or **Learning Management Systems (LMS)**. Whether you are a cybersecurity instructor, a business leader, or an enthusiast, this recipe will provide a practical and innovative way to engage with cybersecurity education.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

1. `openai`: This library enables you to interact with the OpenAI API. Install it using the command `pip install openai`.
2. `os`: This is a built-in Python library, which allows you to interact with the operating system, especially for accessing environment variables.
3. `tqdm`: This library is utilized for showing progress bars during the policy generation process. Install it with `pip install tqdm`.
4. A text file named `trainingcontent.txt`: This file should contain the categories you wish to base your assessment on. Each line should contain one category. This file should be in the same directory as your Python script.

How to do it…

Before we begin, let's note a few things. The assessment will consist of multiple-choice questions generated by ChatGPT. Each question will come with four options, and only one of these will be correct. The responses you provide will guide ChatGPT's interaction, helping it keep score, provide explanations, and give feedback on your performance. So let's get started.

1. **Log into your OpenAI account and access the ChatGPT interface.** Visit the website at `https://chat.openai.com` to get started.

2. **Generate a cybersecurity awareness training assessment.** Use the following prompt to instruct ChatGPT to start creating your cybersecurity awareness training assessment.

   ```
   You are a cybersecurity professional and instructor with more
   than 25 years of experience. Create a cybersecurity awareness
   training (for employees) assessment test via this chat
   conversation. Provide no other response other than to ask me a
   cybersecurity awareness related question and provide 4 multiple
   choice options with only one being the correct answer. Provide
   no further generation or response until I answer the question.
   If I answer correctly, just respond with "Correct" and a short
   description to further explain the answer, and then repeat the
   process. If I answer incorrectly, respond with "Incorrect", then
   the correct answer, then a short description to further explain
   the answer. Then repeat the process.

   Ask me only 10 questions in total throughout the process and
   remember my answer to them all. After the last question has been
   answered, and after your response, end the assessment and give
   me my total score, the areas/categories I did well in and where
   I need to improve.
   ```

3. **Generate a content specific assessment.** If you want a specific assessment for a cybersecurity awareness course, such as the one created in the *Developing Security Awareness Training Content* recipe, use this alternative prompt:

   ```
   You are a cybersecurity professional and instructor with more
   than 25 years of experience. Create a cybersecurity awareness
   training (for employees) assessment test via this chat
   conversation. Provide no other response other than to ask me a
   cybersecurity awareness related question and provide 4 multiple
   choice options with only one being the correct answer. Provide
   no further generation or response until I answer the question.
   If I answer correctly, just respond with "Correct" and a short
   description to further explain the answer, and then repeat the
   process. If I answer incorrectly, respond with "Incorrect", then
   the correct answer, then a short description to further explain
   the answer. Then repeat the process.

   Ask me only 10 questions in total throughout the process and
   remember my answer to them all. After the last question has been
   answered, and after your response, end the assessment and give
   me my total score, the areas/categories I did well in and where
   I need to improve.

   Base the assessment on the following categories:

   Introduction to Cybersecurity
   Importance of Cybersecurity in the Electric Utility Industry
   Understanding Cyber Threats: Definitions and Examples
   Common Cyber Threats in the Electric Utility Industry
   ```

```
The Consequences of Cyber Attacks on Electric Utilities
Identifying Suspicious Emails and Phishing Attempts
The Dangers of Malware and How to Avoid Them
Safe Internet Browsing Practices
The Importance of Regular Software Updates and Patches
Securing Mobile Devices and Remote Workstations
The Role of Passwords in Cybersecurity: Creating Strong
Passwords
Two-Factor Authentication and How It Protects You
Protecting Sensitive Information: Personal and Company Data
Understanding Firewalls and Encryption
Social Engineering: How to Recognize and Avoid
Handling and Reporting Suspected Cybersecurity Incidents
Role of Employees in Maintaining Cybersecurity
Best Practices for Cybersecurity in the Electric Utility
Industry
```

Tip

Experiment with the number of questions asked and the categories asked to get the results that work best for your needs.

How it works...

The success of this recipe lies in the intricate design of the prompts and the manner in which they guide ChatGPT's behavior to provide an interactive, Q&A-based assessment experience. Each instruction within the prompt corresponds to a task that ChatGPT is capable of executing. The OpenAI models have been trained on a diverse range of data and can generate relevant questions based on the input provided.

The initial portion of the prompt positions ChatGPT as an experienced cybersecurity professional and instructor, which sets the context for the kind of responses we expect. This is crucial in guiding the model to generate content related to cybersecurity awareness.

We further instruct the model to maintain the flow of a standard assessment: posing a question, waiting for a response, then giving feedback. We explicitly state that the AI should ask a question and provide four multiple choice options, giving it a clear structure to follow. The feedback, whether it's **Correct** or **InCorrect**, is designed to include a short explanation to supplement the learner's understanding.

One unique aspect of the prompt design is its built-in memory management. We instruct the model to remember all responses throughout the conversation. This way, we get a cumulative scoring mechanism, adding an element of progression and continuity to the interaction. This isn't perfect, as AI models have limited memory and cannot track context beyond a certain limit, but it's effective for the scope of this application.

Importantly, we restrict the model's responses to maintain the assessment context. The prompt explicitly states that the model should provide no other response apart from the question and feedback loop. This restriction is vital in ensuring that the model doesn't deviate from the intended conversational flow.

For custom assessments, we provide a list of specific topics to base the questions on, leveraging the model's ability to understand and generate questions from given subjects. This way, the model tailors the assessment to the specific needs of a cybersecurity awareness course.

In essence, the prompt's structure and creativity help harness ChatGPT's capabilities, transforming it into an interactive tool for cybersecurity awareness assessment.

Important note

While the models are good at understanding and generating human-like text, they don't *know* things in the way humans do. They can't remember specific details beyond what is available in the conversation context.

Different models have different strengths and weaknesses you might want to consider for this recipe. **GPT-4** has the ability to handle longer context (more assessment questions), but it's a bit slower and you can only submit 25 prompts over 3 hours (at the time of this writing). **GPT-3.5** is faster and doesn't have any prompt limitations. However, it might lose context over long assessments and provide inaccurate results at the end of the assessment.

In a nutshell, this recipe leverages the capabilities of OpenAI models to create a highly interactive and informative cybersecurity awareness assessment.

There's more…

If you were using a LMS, you might prefer a question set document rather than an interactive method like ChatGPT. In this case, Python scripting provides a convenient alternative, creating a static question set that you can then import into your LMS or use in an in-person training session.

Tip

Different models have different context memory windows. The more questions the script generates, the better the chance that the model will lose context along the way and provide inconsistent or out of context results. For more questions, try using the **gpt-4** model, which has twice the context window as **gpt-3.5-turbo**, or even the new **gpt-3.5-turbo-16k**, which has 4 times the context window.

Here are the steps to do it:

1. **Import necessary libraries.** For this script, we'll need to import **openai**, **os**, **threading**, **time**, **datetime**, and **tqdm**. These libraries will allow us to interact with the OpenAI API, manage files, and create multi-threading.

```
import openai
from openai import OpenAI
import os
import threading
import time
from datetime import datetime
from tqdm import tqdm
```

2. **Set up the OpenAI API.** You will need to provide your OpenAI API key, which you can store as an environment variable for security purposes.

```
openai.api_key = os.getenv("OPENAI_API_KEY")
```

3. **Set up the filename for the assessment.** We use the current date and time to create a unique name for each assessment.

```
current_datetime = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
assessment_name = f"Cybersecurity_Assessment_{current_datetime}.
txt"
```

4. **Define the function to generate questions.** This function creates a conversation with the AI model, using a similar approach as in the interactive session. It includes a function parameter for the categories.

```
def generate_question(categories: str) -> str:
    messages = [
        {"role": "system", "content": 'You are a cybersecurity
professional and instructor with more than 25 years of
experience.'},
        {"role": "user", "content": f'Create a cybersecurity
awareness training (for employees) assessment test. Provide
no other response other than to create a question set of 10
cybersecurity awareness questions. Provide 4 multiple choice
options with only one being the correct answer. After the
question and answer choices, provide the correct answer and
then provide a short contextual description. Provide no further
generation or response.\n\nBase the assessment on the following
categories:\n\n{categories}'},
    ]

    client = OpenAI()
response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    return response.choices[0].message.content.strip()
```

Important note

You can adjust the number of questions here to suit your needs. You can also modify the prompt to tell it you want at least x questions per category.

5. **Display elapsed time.** This function is used to provide a user-friendly display of elapsed time during the API call.

```
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f} seconds",
end="")
        time.sleep(1)
```

6. **Prepare and execute the API call.** We read the content categories from a file and initiate a thread to display the elapsed time. We then call the function to generate the questions.

```
try:
    with open("trainingcontent.txt") as file:
        content_categories = ', '.join([line.strip() for line in
file.readlines()])
except FileNotFoundError:
    content_categories = ''

api_call_completed = False
elapsed_time_thread = threading.Thread(target=display_elapsed_
time)
elapsed_time_thread.start()

try:
    questions = generate_question(content_categories)
except Exception as e:
    print(f"\nAn error occurred during the API call: {e}")
```

```
        exit()

    api_call_completed = True
    elapsed_time_thread.join()
```

7. **Save the generated questions.** Once the questions are generated, we write them to a file with the previously defined filename.

```
    try:
        with open(assessment_name, 'w') as file:
            file.write(questions)
        print("\nAssessment generated successfully!")
    except Exception as e:
        print(f"\nAn error occurred during the assessment 
    generation: {e}")
```

Here's how the complete script should look:

```
import openai
from openai import OpenAI
import os
import threading
import time
from datetime import datetime
from tqdm import tqdm

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

current_datetime = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
assessment_name = f"Cybersecurity_Assessment_{current_datetime}.txt"

def generate_question(categories: str) -> str:
    # Define the conversation messages
    messages = [
        {"role": "system", "content": 'You are a cybersecurity 
professional and instructor with more than 25 years of experience.'},
        {"role": "user", "content": f'Create a cybersecurity awareness 
training (for employees) assessment test. Provide no other response 
other than to create a question set of 10 cybersecurity awareness 
questions. Provide 4 multiple choice options with only one being 
the correct answer. After the question and answer choices, provide 
the correct answer and then provide a short contextual description. 
Provide no further generation or response.\n\nBase the assessment on 
the following categories:\n\n{categories}'},
    ]
    # Call the OpenAI API
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    # Return the generated text
    return response.choices[0].message.content.strip()

# Function to display elapsed time while waiting for the API call
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f} seconds", end="")
        time.sleep(1)

# Read content categories from the file
try:
    with open("trainingcontent.txt") as file:
        content_categories = ', '.join([line.strip() for line in file.
readlines()])
except FileNotFoundError:
    content_categories = ''

api_call_completed = False
elapsed_time_thread = threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()

# Generate the report using the OpenAI API
try:
    # Generate the question
    questions = generate_question(content_categories)
except Exception as e:
    print(f"\nAn error occurred during the API call: {e}")
    api_call_completed = True
    exit()

api_call_completed = True
```

```
elapsed_time_thread.join()

# Save the questions into a text file
try:
    with open(assessment_name, 'w') as file:
        file.write(questions)
    print("\nAssessment generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the assessment generation: 
{e}")
```

After these steps, you'll have a text file with a set of questions generated by the model that you can use for your cybersecurity awareness training!

Here's how it works:

This Python script is designed to generate a set of cybersecurity awareness training questions. It works by making use of the OpenAI **gpt-3.5-turbo** model through a series of API calls to generate the questions based on specific categories. The categories are read from a text file named `trainingcontent.txt`, where each line is considered a separate category.

The script first imports the necessary libraries, including **openai** to interact with the **gpt-3.5-turbo** model, **os** for operating system-dependent functionality such as reading environment variables (the API key, in this case), **threading** and **time** to create a separate thread that displays the elapsed time during the API call, **datetime** to get the current date and time for naming the output file, and **tqdm** to provide progress bars.

Once the API key is set, the script then constructs a filename for the output assessment file. It appends the current date and time to a base name to ensure the output file has a unique name each time the script is run.

Next, the `generate_question` function is defined, which sets up a conversation with the ChatGPT model. It starts by setting the system role message, establishing the perspective of the user (a cybersecurity professional), and then asks for the creation of a cybersecurity awareness training assessment test. It uses the categories parameter in the user's message to the model. This parameter is replaced later with the actual categories read from the file.

The `display_elapsed_time` function is designed to display the elapsed time since the API call started until it finishes. This function runs on a separate thread to keep updating the elapsed time on the console without blocking the main thread where the API call is made.

The content categories are read from the file `trainingcontent.txt`, and a new thread is created to display elapsed time. An API call is then made by calling the `generate_question` function and passing the content categories. If an exception occurs during the API call (for example, if there is a problem with the network connection), the script stops execution and reports the error.

Finally, once the API call is complete and the generated questions are received, they are written into the output file. If any exception occurs during the writing process (for example, if there is a problem with write permissions), the error is reported to the console.

Overall, this script provides a practical way to generate a set of questions for cybersecurity awareness training using the OpenAI **gpt-3.5-turbo** model. The structure of the prompt and the specific parameters used in the API call help ensure that the output is tailored to the specific needs of the training.

Interactive Email Phishing Training with ChatGPT

With the rise of cyber threats, organizations of all sizes are increasingly aware of the importance of training their staff on email phishing, a common and potentially dangerous tactic employed by cyber criminals. In this recipe, we'll be using ChatGPT to create a tool for interactive email phishing training.

This recipe guides you through the process of crafting a specialized prompt to turn ChatGPT into a simulation tool for phishing attack awareness. With this approach, you can use ChatGPT to train users to identify potential phishing emails, thereby increasing their awareness and helping to protect your organization from potential security threats.

What makes this truly powerful is its interactive nature. ChatGPT will present the user with a series of email scenarios. The user will then decide whether the email is a phishing attempt or a legitimate email, and can even ask for more details such as the URL to a link in the email or header information, for example. ChatGPT will provide feedback, ensuring a continuous, engaging, and efficient learning experience.

Additionally, we will also cover how to use Python in conjunction with these prompts to create exportable email simulation scenarios. This feature can be beneficial in situations where you might want to use the generated scenarios outside of ChatGPT, such as in a live course or in a LMS.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

1. `openai`: This library enables you to interact with the OpenAI API. Install it using the command `pip install openai`.
2. `os`: This is a built-in Python library, which allows you to interact with the operating system, especially for accessing environment variables.
3. `tqdm`: This library is utilized for showing progress bars during the policy generation process. Install it with `pip install tqdm`.

How to do it...

In this section, we will walk you through the process of creating an interactive email phishing training simulation using ChatGPT. The instructions are broken down into steps, starting from logging into your OpenAI account and ending with generating phishing training simulations.

1. **Access the ChatGPT interface.** Log in to your OpenAI account and go to the ChatGPT interface at `https://chat.openai.com`.
2. **Initialize the simulation by entering the specialized prompt.** The following prompt is carefully designed to instruct ChatGPT to act as a phishing training simulator. Enter the prompt into the text box and press Enter.

```
"You are a cybersecurity professional and expert in adversarial
social engineering tactics, techniques, and procedures, with
25 years of experience. Create an interactive email phishing
training simulation (for employees). Provide no other response
other than to ask the question, "Is the following email real
or a phishing attempt? (You may ask clarification questions
such as URL information, header information, etc.)" followed
by simulated email, using markdown language formatting. The
email you present can represent a legitimate email or a phishing
attempt, which can use one or more various techniques. Provide
no further generation or response until I answer the question.
If I answer correctly, just respond with "Correct" and a short
description to further explain the answer, and then restart
the process from the beginning. If I answer incorrectly,
respond with "Incorrect", then the correct answer, then a short
description to further explain the answer. Then repeat the
process from the beginning.

Present me with only 3 simulations in total throughout the
process and remember my answer to them all. At least one of
the simulations should simulate a real email. After the last
question has been answered, and after your response, end the
assessment and give me my total score, the areas I did well in
and where I need to improve."
```

Tip

Be sure to change the number of simulations ChatGPT provides, to suit your needs.

Now, ChatGPT will generate interactive email phishing scenarios based on your instructions. Respond to each scenario as if you were the employee undergoing the training. After the third scenario and your final response, ChatGPT will calculate and provide your total score, areas of strength, and areas for improvement.

How it works...

At the heart of this recipe lies the specialized prompt. This prompt is constructed to instruct ChatGPT to act as an interactive phishing training tool, delivering a series of email phishing scenarios. The prompt follows certain design principles which are essential to its effectiveness and interaction with the OpenAI models. Here, we'll dissect those principles:

1. **Defining the role:** The prompt starts by setting up the role of the AI model – a cybersecurity professional and expert in adversarial social engineering tactics, techniques, and procedures, with 25 years of experience. By defining the AI's persona, we direct the model to generate responses using the knowledge and expertise expected from such a role.
2. **Detailed instructions and simulation:** The instructions given in the prompt are meticulously detailed, and it is this precision that enables ChatGPT to create effective and realistic phishing simulations. The prompt asks the AI model to generate a phishing email scenario, followed by the question, "Is the following email real or a phishing attempt?". Notably, the AI model is given the liberty to provide additional clarifying questions, such as asking about URL information, header information, etc., giving it the freedom to generate more nuanced and complex scenarios.

   By asking the model to generate these emails using markdown language formatting, we ensure that the simulated emails have the structure and appearance of genuine emails, enhancing the realism of the simulation. The AI is also instructed to present emails that can represent either legitimate correspondence or a phishing attempt, ensuring a diverse range of scenarios for the user to evaluate.

   How can ChatGPT convincingly simulate phishing emails? Well, ChatGPT's strength comes from the wide variety of text it has been trained on, including (but not limited to) countless examples of email correspondences and probably some instances of phishing attempts or discussions around them. From this extensive training, the model has developed a robust understanding of the format, tone, and common phrases used in both legitimate and phishing emails. So, when prompted to simulate a phishing email, it can draw on this knowledge to generate a believable email that mirrors the features of a real-world phishing attempt.

   As the model doesn't generate responses until it receives an answer to its question, it guarantees an interactive user experience. Based on the user's response, the model provides relevant feedback (**Correct** or **Incorrect**), the correct answer if the user was wrong, and a brief explanation. This detailed, immediate feedback aids the learning process and helps to embed the knowledge gained from each simulated scenario.

   It's worth noting that, although the model has been trained to generate human-like text, it doesn't understand the content in the same way humans do. It doesn't have beliefs, opinions, or access to real-time, world-specific information or personal data unless explicitly provided in the conversation. Its responses are merely predictions based on its training data. The carefully designed prompt and structure are what guide the model to generate useful, contextually appropriate content for this particular task.

3. **Feedback mechanism:** The prompt instructs the AI to provide feedback based on the user's answer, further explaining the answer. This creates an iterative feedback loop that enhances the learning experience.
4. **Keeping track of progress:** The prompt instructs the AI to present three simulations in total and to remember the user's answer to all of them. This ensures continuity in the training and enables tracking of the user's progress.
5. **Scoring and areas of improvement:** After the final simulation and response, the prompt instructs the AI to end the assessment and provide a total score along with areas of strength and areas for improvement. This helps the user understand their proficiency and the areas they need to focus on for improvement.

ChatGPT's models are trained on a broad range of internet text. However, it's essential to note that it does not know specifics about which documents were part of its training set or have access to any private, confidential, or proprietary information. It generates responses to prompts by recognizing patterns and producing text that statistically aligns with the patterns it observed in its training data.

By structuring our prompt in a way that clearly defines the interactive assessment context and expected behavior, we're able to leverage this pattern recognition to create a highly specialized interactive tool. The ability of the OpenAI models to handle such a complex and interactive use case demonstrates their powerful capability and flexibility.

There's more...

If you're using a LMS or conducting a live class, you might prefer to have a list of scenarios and details rather than an interactive method like ChatGPT. In these settings, it's often more practical to provide learners with specific scenarios to ponder and discuss in a group setting. The list can also be used for assessments or training materials, offering a static reference point that learners can revisit as needed, or as content for a phishing simulation system.

By modifying the script from the previous recipe, you can instruct the ChatGPT model to produce a set of phishing email simulations complete with all necessary details. The resulting text can be saved into a file for easy distribution and usage in your training environment.

Since this script is so similar to the one from the previous recipe, we'll just cover the modifications instead of steppping through the entire script again.

Let's walk through the necessary modifications:

1. **Rename and modify the function:** The function `generate_question` is renamed to `generate_email_simulations`, and its argument list and body are updated to reflect its new purpose. It will now generate the phishing email simulations instead of cybersecurity awareness questions. This is done by updating the messages that are passed to the OpenAI API within this function.

```
def generate_email_simulations() -> str:
    # Define the conversation messages
    messages = [
        {"role": "system", "content": 'You are a cybersecurity
professional and expert in adversarial social engineering
tactics, techniques, and procedures, with 25 years of
experience.'},
        {"role": "user", "content": 'Create a list of fictitious
emails for an interactive email phishing training. The emails
can represent a legitimate email or a phishing attempt, using
one or more various techniques. After each email, provide the
answer, contextual descriptions, and details for any other
relevant information such as the URL for any links in the email,
header information. Generate all necessary information in the
email and supporting details. Present 3 simulations in total. At
least one of the simulations should simulate a real email.'},
    ]
    ...
```

Important note

You can adjust the number of scenarios here to suit your needs. In this example, we've requested 3 scenarios.

2. **Remove unnecessary code:** The script no longer reads content categories from an input file, as it's not required in your use case.
3. **Update variable and function names:** All variable and function names referring to "questions" or "assessment" have been renamed to refer to "email simulations" instead, to make the script more understandable in the context of its new purpose.
4. **Call the appropriate function:** The `generate_email_simulations` function is called instead of the `generate_question` function. This function initiates the process of generating the email simulations.

```
# Generate the email simulations
email_simulations = generate_email_simulations()
```

Tip

Like the previous method, more scenarios will require a model that supports a larger context window. However, the **gpt-4** model seems to provide better results in terms of accuracy, depth, and consistency with its generations for this recipe.

Here's how the complete script should look:

```
import openai
from openai import OpenAI
import os
import threading
import time
from datetime import datetime

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

current_datetime = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
assessment_name = f"Email_Simulations_{current_datetime}.txt"

def generate_email_simulations() -> str:
    # Define the conversation messages
    messages = [
        {"role": "system", "content": 'You are a cybersecurity 
professional and expert in adversarial social engineering tactics, 
techniques, and procedures, with 25 years of experience.'},
        {"role": "user", "content": 'Create a list of fictitious 
emails for an interactive email phishing training. The emails can 
represent a legitimate email or a phishing attempt, using one or more 
various techniques. After each email, provide the answer, contextual 
descriptions, and details for any other relevant information such as 
the URL for any links in the email, header information. Generate all 
necessary information in the email and supporting details. Present 3 
simulations in total. At least one of the simulations should simulate 
a real email.'},
    ]

    # Call the OpenAI API
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    # Return the generated text
    return response.choices[0].message.content.strip()

# Function to display elapsed time while waiting for the API call
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f} seconds", end="")
        time.sleep(1)

api_call_completed = False
elapsed_time_thread = threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()

# Generate the report using the OpenAI API
try:
    # Generate the email simulations
    email_simulations = generate_email_simulations()
except Exception as e:
    print(f"\nAn error occurred during the API call: {e}")
    api_call_completed = True
    exit()

api_call_completed = True
elapsed_time_thread.join()

# Save the email simulations into a text file
try:
    with open(assessment_name, 'w') as file:
        file.write(email_simulations)
    print("\nEmail simulations generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the email simulations
generation: {e}")
```

By running this modified script, the ChatGPT model is directed to generate a series of interactive email phishing training scenarios. The script then collects the generated scenarios, checks them for errors, and writes them to a text file. This gives you a ready-made training resource that you can distribute to your learners or incorporate into your LMS or live training sessions.

ChatGPT-Guided Cybersecurity Certification Study

This recipe will guide you through the process of using ChatGPT to create an interactive certification study guide, specifically designed for cybersecurity certifications like **CISSP**. The approach will leverage ChatGPT's conversational abilities to pose a series of questions mimicking the ones typically found on the specified certification exam. Furthermore, ChatGPT will provide you with additional context after each question, offering helpful insights and explanations. To round off the study session, ChatGPT will also evaluate your performance, highlighting areas for improvement and suggesting suitable study resources. This recipe could serve as a powerful study tool for anyone preparing for a cybersecurity certification exam.

Getting ready

Before diving in to this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

1. `openai`: This library enables you to interact with the OpenAI API. Install it using the command `pip install openai`.
2. `os`: This is a built-in Python library, which allows you to interact with the operating system, especially for accessing environment variables.
3. `tqdm`: This library is utilized for showing progress bars during the policy generation process. Install it with `pip install tqdm`.

How to do it…

This interactive certification study guide will be created directly on the OpenAI platform, specifically in the ChatGPT interface. The process is simple and straightforward.

1. **Access the ChatGPT interface.** Log into your OpenAI account and go to the ChatGPT interface at `https://chat.openai.com`.
2. **Initialize the session by entering the specialized prompt.** The following prompt is carefully designed to instruct ChatGPT to act as a phishing training simulator. Enter the prompt into the text box and press Enter.

```
You are a cybersecurity professional and training instructor
with more than 25 years of experience. Help me study for the
CISSP exam. Generate 5 questions, one at a time, just as they
will appear on the exam or practice exams. Present the question
and options and nothing else and wait for my answer. If I answer
correctly, say, "Correct" and move on to the next question.
If I answer incorrectly, say, "Incorrect", present me with the
```

```
correct answer, and any context for clarification, and then move
on to the next question. After all questions have been answered,
tally my results, present me with my score, tell me what areas I
need to improve on, and present me with appropriate resources to
help me study for the areas I need to improve in.
```

Important note

The certification exam mentioned in the prompt can be replaced with the one you're interested in. However, remember that ChatGPT's training data only extends up to **September 2021**, so it won't have information about certifications updated or introduced after that date.

Tip

We will be presenting another recipe later in this book on how to get ChatGPT and/or the OpenAI to access more recent information for more up-to-date exam practice.

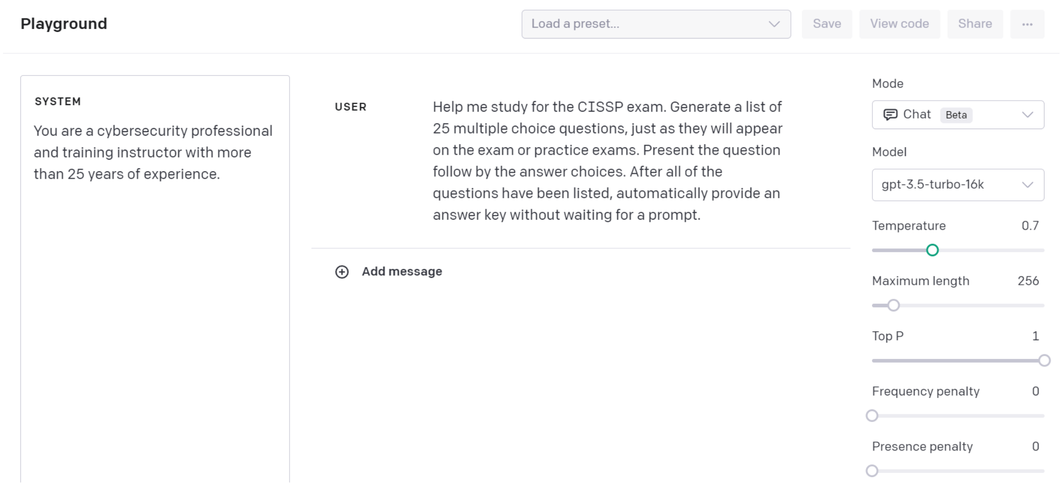
How it works…

This recipe leverages the AI's role-playing and interactive conversational capabilities to create an engaging study session. When given the role of a seasoned cybersecurity professional and instructor, ChatGPT generates a sequence of realistic certification exam questions, validates your answers, provides corrective feedback, and supplies additional context or explanation where needed. The prompt structure ensures that the AI maintains the focus on the task at hand, guiding the interaction to create an effective learning environment.

The approach relies on ChatGPT's ability to understand and generate human-like text based on the instructions provided. In the context of this recipe, the AI model employs its underlying language understanding to generate relevant cybersecurity certification exam questions and provide informative responses.

Important note

As has been mentioned throughout this book, the chosen model dictates the limitations you'll face. **GPT-4** offers a significantly larger context window (allowing for more questions before potentially straying) than **GPT-3.5**. If you have access to the **OpenAI Playground**, you can use the **gpt-3.5-turbo-16k** model, which has the largest context window to date.



Playground

Load a preset... Save View code Share ...

SYSTEM

You are a cybersecurity professional and training instructor with more than 25 years of experience.

USER

Help me study for the CISSP exam. Generate a list of 25 multiple choice questions, just as they will appear on the exam or practice exams. Present the question follow by the answer choices. After all of the questions have been listed, automatically provide an answer key without waiting for a prompt.

Add message

Mode: Chat Beta

Model: gpt-3.5-turbo-16k

Temperature: 0.7

Maximum length: 256

Top P: 1

Frequency penalty: 0

Presence penalty: 0

Figure 5.1 – Using the gpt-3.5-turbo-16k model in the OpenAI Playground

There's more…

If you're interested in generating a complete list of questions for a study group or class, you can adapt the script from the previous recipe (*Interactive Email Phishing Training with ChatGPT*). Here is the role and prompt to use:

Role:

```
You are a cybersecurity professional and training instructor with more
than 25 years of experience.
```

Prompt:

```
Help me study for the CISSP exam. Generate a list of 25 multiple
choice questions, just as they will appear on the exam or practice
exams. Present the question followed by the answer choices. After all
of the questions have been listed, automatically provide an answer key
without waiting for a prompt.
```

Remember to replace the certification name if needed, adjust the number of questions, choose the appropriate model, and modify the filename for the generated output (unless you're okay with the file being called "Email_Simulations_…").

Here's an example of what the modified script would like like:

```
import openai
from openai import OpenAI
import os
import threading
import time
from datetime import datetime

# Set up the OpenAI API
openai.api_key = os.getenv("OPENAI_API_KEY")

current_datetime = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
assessment_name = f"Exam_questions_{current_datetime}.txt"

def generate_email_simulations() -> str:
    # Define the conversation messages
    messages = [
        {"role": "system", "content": 'You are a cybersecurity
professional and training instructor with more than 25 years of
experience.'},
        {"role": "user", "content": 'Help me study for the CISSP exam.
Generate a list of 25 multiple choice questions, just as they will
appear on the exam or practice exams. Present the question follow
by the answer choices. After all of the questions have been listed,
automatically provide an answer key without waiting for a prompt.'},
    ]

    # Call the OpenAI API
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7,
    )

    # Return the generated text
    return response.choices[0].message.content.strip()

# Function to display elapsed time while waiting for the API call
def display_elapsed_time():
    start_time = time.time()
    while not api_call_completed:
        elapsed_time = time.time() - start_time
        print(f"\rElapsed time: {elapsed_time:.2f} seconds", end="")
        time.sleep(1)

api_call_completed = False
```

```
elapsed_time_thread = threading.Thread(target=display_elapsed_time)
elapsed_time_thread.start()

# Generate the report using the OpenAI API
try:
    # Generate the email simulations
    email_simulations = generate_email_simulations()
except Exception as e:
    print(f"\nAn error occurred during the API call: {e}")
    api_call_completed = True
    exit()

api_call_completed = True
elapsed_time_thread.join()

# Save the email simulations into a text file
try:
    with open(assessment_name, 'w') as file:
        file.write(email_simulations)
    print("\nEmail simulations generated successfully!")
except Exception as e:
    print(f"\nAn error occurred during the email simulations
generation: {e}")
```

Just like the script in the previous recipe, this script will generate a text document containing the response from the API. In this case, that's the list of certification exam questions and the answer key.

Gamifying Cybersecurity Training

Gamification, the application of game-design elements in non-game contexts, has transformed many areas of education and training, and cybersecurity is no exception. As the creator of one of the world's first educational cybersecurity video games, *ThreatGEN® Red vs. Blue*, I might be a bit biased. However, I firmly believe that gamification is the educational medium of the future.

The exciting world of gamification has increasingly become the go-to methodology for many forms of education and training. The essence of gamification is creating a game-like environment that keeps individuals engaged, thus enhancing the learning process. One of the most intriguing and promising applications of ChatGPT and OpenAI's LLMs is the ability to gamify cybersecurity education.

From Gen X and younger, most people have grown up in a culture of gaming. This trend, combined with the explosion of gamification and game-based learning over the last few years, has led to a significant shift in the way education and training are delivered. And when it comes to cybersecurity, the marriage of gaming and education provides an engaging, interactive, and fun way to learn complex concepts.

This recipe will show you how to turn ChatGPT into the **game master** (**GM**) for a cybersecurity-themed **role-playing game** (**RPG**). The game we'll be playing is "Find the Insider Threat", a "who done it" mystery. The objective of the game is to interview staff and investigate systems to find the insider threat, all in 50 turns or less. ChatGPT will manage the game, keeping score and tracking your turns. It will even provide a detailed report after the game, outlining your successes, failures, and areas for improvement.

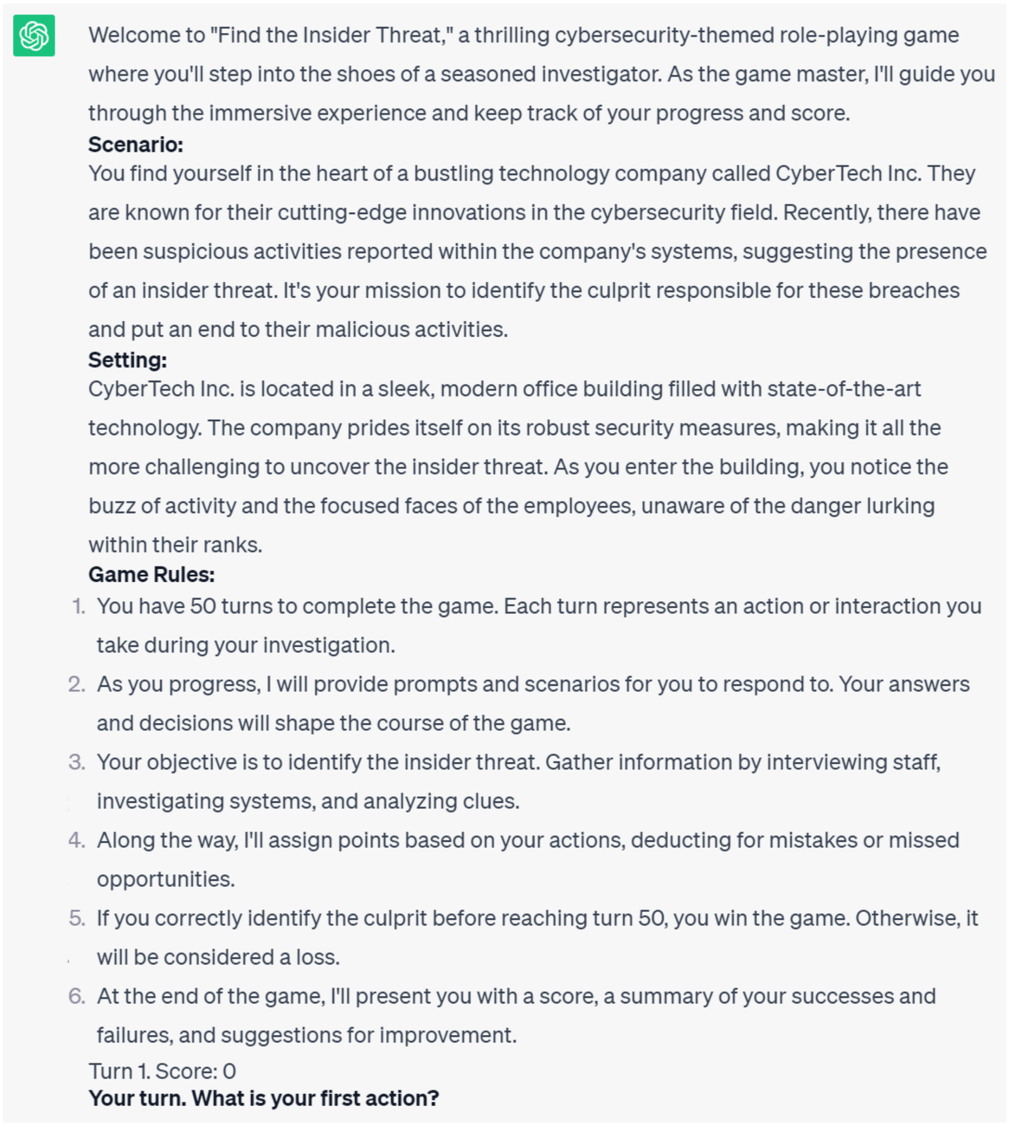
Getting ready

The prerequisites for this recipe are simple. You just need a web browser and an OpenAI account. If you haven't already created an account or need a refresher on how to use the ChatGPT interface, refer back to *Chapter 1* for a comprehensive guide.

How to do it...

1. **Access the ChatGPT interface.** Log into your OpenAI account and go to the ChatGPT interface at `https://chat.openai.com`.
2. **Initialize the game by entering the specialized prompt** The following prompt is carefully designed to instruct ChatGPT to act as a phishing training simulator. Enter the prompt into the text box and press Enter.

```
"You are a cybersecurity professional with more than 25 years
of experience and an expert in gamification and game-based
training. You will be the game master for a cybersecurity themed
role-playing game (RPG). The game is "Find the Insider Threat",
a "who did it" mystery. The object is to interview staff and
investigate systems to find the insider threat. I must do it in
50 turns or less. Keep score by adding and subtracting points
as you see fit, as I go. If I find the culprit (I win) or after
turn 50 the game is over (I lose). At that time, present me with
my score, the game summary, my successes, my failures, and where
I can improve (keeping in mind this is meant to be cybersecurity
educational). When the game starts, present me with the
scenario, setting, and game rules. Stay in character as the game
master, keep track of each turn (every prompt after you present
the game scenario, setting, and rules, is a game turn). At the
end of each of your responses after the game starts, you will
remind me of the turn and my current score (score starts at 0).
Use markdown language in your prompts, to make the presentation
more interesting and more readable.

If you understand, start the game."
```



Welcome to "Find the Insider Threat," a thrilling cybersecurity-themed role-playing game where you'll step into the shoes of a seasoned investigator. As the game master, I'll guide you through the immersive experience and keep track of your progress and score.

Scenario:

You find yourself in the heart of a bustling technology company called CyberTech Inc. They are known for their cutting-edge innovations in the cybersecurity field. Recently, there have been suspicious activities reported within the company's systems, suggesting the presence of an insider threat. It's your mission to identify the culprit responsible for these breaches and put an end to their malicious activities.

Setting:

CyberTech Inc. is located in a sleek, modern office building filled with state-of-the-art technology. The company prides itself on its robust security measures, making it all the more challenging to uncover the insider threat. As you enter the building, you notice the buzz of activity and the focused faces of the employees, unaware of the danger lurking within their ranks.

Game Rules:

1. You have 50 turns to complete the game. Each turn represents an action or interaction you take during your investigation.
2. As you progress, I will provide prompts and scenarios for you to respond to. Your answers and decisions will shape the course of the game.
3. Your objective is to identify the insider threat. Gather information by interviewing staff, investigating systems, and analyzing clues.
4. Along the way, I'll assign points based on your actions, deducting for mistakes or missed opportunities.
5. If you correctly identify the culprit before reaching turn 50, you win the game. Otherwise, it will be considered a loss.
6. At the end of the game, I'll present you with a score, a summary of your successes and failures, and suggestions for improvement.

Turn 1. Score: 0

Your turn. What is your first action?

Figure 5.2 – Example game initialization output

Important note

The model limitations will come into play significantly in this recipe due to the potential for long, contextual prompts. **GPT-4** will give you the longest context window, but you will be limited to 25 prompts over 3 hours, including the initial prompts. So, you might want to limit games using **GPT-4** to 20 turns. **GPT-3.5** has no prompt limits, but the context window is smaller. Therefore, ChatGPT might lose context after a certain point in the game and forget details from earlier in the game. It should remember the turn and score since that is restated every turn, but details from earlier prompts, especially at and just after initialization, will be lost. This includes the entire setting. ChatGPT does, however, attempt to maintain context the best it can by deriving context from what it does have access to. Sometimes, that can be just enough.

Tip

Play around (pun intended) with the turn limit and even the theme or game style to find a setting that works for your interests and needs.

How it works…

This recipe essentially transforms ChatGPT into a game master for a role-playing game. RPGs typically involve a narrative experience where players assume the roles of characters in a fictional setting. The game master (or GM) is the person who runs the game, creates the story and the setting, and adjudicates the rules.

By providing ChatGPT with a prompt that establishes it as the game master, it's directed to construct the narrative and guide you, the player, through the game. The prompt also instructs the model to track game progress, keep score, and provide a detailed report at the end of the game.

The effectiveness of this recipe relies heavily on ChatGPT's capability to generate coherent and contextually relevant responses. It needs to maintain the continuity of the game narrative while simultaneously tracking the score and turn count. This is achieved by ensuring that each of ChatGPT's responses includes a reminder of the turn and the current score.

However, it's worth mentioning, yet again, the limitations that exist regarding the model's capacity to remember context. The context window of **GPT-3.5** is smaller than **GPT-4**, which can impact the continuity of the game, especially if it extends over numerous turns.

There's more…

This recipe provides just a glimpse into the exciting and dynamic world of gamified cybersecurity training. By manipulating the prompts, the scope of the game, and the role of the AI, you can create entirely different scenarios that cater to different cybersecurity skills or areas of interest.

For instance, in our recipe, we used a "who done it" mystery to identify the insider threat. However, you could potentially adapt this approach to your specific interests or needs. If you're a more technically inclined person, you could focus the theme around the tasks for something more technical, such as performing a threat hunting exercise on a single system... RPG style! This unique blend of learning and entertainment provides a tailored educational experience, making the learning process much more engaging and fun.

Moreover, gamified cybersecurity training isn't limited to solo play. It's a fantastic tool for team building exercises, tradeshow events, or even a game night with friends. By fostering an interactive learning environment, you can elevate the educational experience, making it more memorable and effective.

6
Red Teaming and Penetration Testing

Penetration testing and red teaming are specialized approaches to cybersecurity assessment. Penetration testing, often referred to as *ethical hacking*, involves the simulation of cyber-attacks on a system, network, or application to uncover vulnerabilities that could be exploited by malicious actors. Red teaming, on the other hand, is a more comprehensive and adversarial engagement that simulates a full-scale attack to evaluate an organization's detection and response capabilities. Emulating adversarial tactics using such methods is crucial for evaluating the security posture of an organization.

By emulating the tactics and techniques of real-world adversaries, these authorized simulations reveal vulnerabilities and attack vectors before they can be exploited by malicious actors. In this chapter, we will explore recipes that leverage AI to enhance red teaming and penetration testing operations.

We will begin by using the **MITRE ATT&CK** framework, the OpenAI API, and Python to swiftly generate realistic red team scenarios. By combining curated adversarial knowledge with the expansive capabilities of **large language models** (**LLMs**), this technique allows us to create threat narratives that closely mirror real-world attacks.

Next, we will harness ChatGPT's natural language prowess to guide us through OSINT reconnaissance. From mining social media to analyzing job postings, these recipes illustrate how to extract actionable intelligence from public data sources in an automated fashion.

To accelerate the discovery of unintentionally exposed assets, we will use Python to automate Google Dorks generated by ChatGPT. Together, these technologies enable the methodical footprinting of an organization's digital footprint.

We wrap up with a unique recipe that infuses a **Kali Linux** terminal with the power of the OpenAI API. By translating natural language requests into OS commands, this AI-enabled terminal provides an intuitive way to navigate complex penetration testing tools and workflows.

By the end of this chapter, you will have an array of strategies powered by AI that augment red team and penetration testing engagements. When applied ethically and with permission, these techniques can uncover oversights, streamline testing, and ultimately, harden the security posture of an organization.

In this chapter, we will cover the following recipes:

- Creating red team scenarios using MITRE ATT&CK and the OpenAI API
- Social media and public data OSINT with ChatGPT
- Google Dork automation with ChatGPT and Python
- Analyzing job postings OSINT with ChatGPT
- GPT-powered Kali Linux terminals

Technical requirements

For this chapter, you will need a **web browser** and a stable **internet connection** to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account set up and have obtained your API key. If not, revisit *Chapter 1* for details. Basic familiarity with the Python programming language and working with the command line is necessary, as you'll be using **Python 3.x**, which needs to be installed on your system, to work with the OpenAI GPT API and create Python scripts. A **code editor** will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter. Finally, since many penetration testing use cases rely heavily on the Linux operating system, access to and familiarity with a Linux distribution (preferably Kali Linux) is recommended.

Kali Linux can be found here:

`https://www.kali.org/get-kali/#kali-platforms`

The code files for this chapter can be found here:

`https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`

Creating red team scenarios using MITRE ATT&CK and the OpenAI API

Red team exercises play a pivotal role in assessing an organization's preparedness against real-world cybersecurity threats. Crafting authentic and impactful red team scenarios is vital for these exercises, yet designing such scenarios can often be intricate. This recipe demonstrates a refined approach to scenario generation by synergizing the **Mitre ATT&CK** framework with the cognitive capabilities of ChatGPT via the OpenAI API. Not only will you be able to swiftly create scenarios but you'll also receive a ranked list of the most relevant techniques, complete with summarized descriptions and example TTP chains, ensuring your red team exercises are as realistic and effective as possible.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

- `openai`: This library enables you to interact with the OpenAI API. Install it using the `pip install openai` command.
- `os`: This is a built-in Python library that allows you to interact with the operating system, especially for accessing environment variables.
- `Mitreattack.stix20`: This library is used for searching Mitre ATT&CK datasets locally on your computer. Install it with `pip install mitreattack-python`.

Finally, you will need a MITRE ATT&CK dataset:

- For this recipe, we will be using `enterprise-attack.json`. You can get MITRE ATT&CK datasets at `https://github.com/mitre/cti`.
- The dataset used in this recipe, specifically, is at `https://github.com/mitre/cti/tree/master/enterprise-attack`.

Once these requirements are in place, you are all set to dive into the script.

How to do it...

Follow these steps:

1. **Set up the environment**: Before diving into the script, ensure you have the necessary libraries and the API key:

   ```
   import openai
   from openai import OpenAI
   import os
   from mitreattack.stix20 import MitreAttackData

   openai.api_key = os.getenv("OPENAI_API_KEY")
   ```

2. **Load the MITRE ATT&CK dataset**: Utilize the `MitreAttackData` class to load the dataset for easy access:

   ```
   mitre_attack_data = MitreAttackData("enterprise-attack.json")
   ```

3. **Extract keywords from the description**: This function integrates ChatGPT to extract relevant keywords from the provided description, which will later be used to search the MITRE ATT&CK dataset:

```
def extract_keywords_from_description(description):
    # Define the merged prompt
    prompt = (f"Given the cybersecurity scenario description:
'{description}', identify and list the key terms, "
              "techniques, or technologies relevant to MITRE
ATT&CK. Extract TTPs from the scenario. "
              "If the description is too basic, expand upon it
with additional details, applicable campaign, "
              "or attack types based on dataset knowledge. Then,
extract the TTPs from the revised description.")

    # Set up the messages for the OpenAI API
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity professional
with more than 25 years of experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]

    # Make the API call
    try:
        client = OpenAI()
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=messages,
            max_tokens=2048,
            n=1,
            stop=None,
            temperature=0.7
        )
        response_content = response.choices[0].message.content.
strip()

        keywords = response_content.split(', ')
        return keywords

    except Exception as e:
        print("An error occurred while connecting to the OpenAI 
API:", e)
        return []
```

4. **Search the MITRE ATT&CK dataset**: With the extracted keywords, the `search_dataset_for_matches` function searches the dataset for potential matches. Then, the `score_matches` function scores the findings:

```
def score_matches(matches, keywords):
    scores = []
    for match in matches:
        score = sum([keyword in match['name'] for keyword in 
keywords]) + \
                sum([keyword in match['description'] for keyword 
in keywords])
        scores.append((match, score))
    return scores

def search_dataset_for_matches(keywords):
    matches = []
    for item in mitre_attack_data.get_techniques():
        if any(keyword in item['name'] for keyword in keywords):
            matches.append(item)
        elif 'description' in item and any(keyword in 
item['description'] for keyword in keywords):
            matches.append(item)
    return matches
```

5. **Generate a comprehensive scenario using ChatGPT**: This function leverages the OpenAI API to generate a summarized description and an example TTP chain for each matched technique:

```
def generate_ttp_chain(match):
    # Create a prompt for GPT-3 to generate a TTP chain for the 
provided match
    prompt = (f"Given the MITRE ATT&CK technique 
'{match['name']}' and its description '{match['description']}', 
"
              "generate an example scenario and TTP chain 
demonstrating its use.")

    # Set up the messages for the OpenAI API
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity professional 
with expertise in MITRE ATT&CK techniques."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]

    # Make the API call
    try:
        client = OpenAI()
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=messages,
            max_tokens=2048,
            n=1,
            stop=None,
            temperature=0.7
        )
        response_content = response.choices[0].message.content.
strip()
        return response_content

    except Exception as e:
        print("An error occurred while generating the TTP
chain:", e)
        return "Unable to generate TTP chain."
```

6. **Put it all together**: Now, integrate all the functions to extract keywords, find matches in the dataset, and generate a comprehensive scenario with a TTP chain:

```
description = input("Enter your scenario description: ")
keywords = extract_keywords_from_description(description)
matches = search_dataset_for_matches(keywords)
scored_matches = score_matches(matches, keywords)

# Sort by score in descending order and take the top 3
top_matches = sorted(scored_matches, key=lambda x: x[1],
reverse=True)[:3]

print("Top 3 matches from the MITRE ATT&CK dataset:")
for match, score in top_matches:
    print("Name:", match['name'])
    print("Summary:", match['description'])
    ttp_chain = generate_ttp_chain(match)
    print("Example Scenario and TTP Chain:", ttp_chain)
    print("-" * 50)
```

By following the preceding steps, you'll have a robust tool at your disposal that can generate realistic red team scenarios using the MITRE ATT&CK framework, all enhanced by the capabilities of ChatGPT.

Here is how the completed script should look:

```
import openai
from openai import OpenAI
import os
from mitreattack.stix20 import MitreAttackData

openai.api_key = os.getenv("OPENAI_API_KEY")

# Load the MITRE ATT&CK dataset using MitreAttackData
mitre_attack_data = MitreAttackData("enterprise-attack.json")

def extract_keywords_from_description(description):
    # Define the merged prompt
    prompt = (f"Given the cybersecurity scenario description: 
'{description}', identify and list the key terms, "
              "techniques, or technologies relevant to MITRE ATT&CK. 
Extract TTPs from the scenario. "
              "If the description is too basic, expand upon it with 
additional details, applicable campaign, "
              "or attack types based on dataset knowledge. Then, 
extract the TTPs from the revised description.")

    # Set up the messages for the OpenAI API
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity professional with more 
than 25 years of experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]

    # Make the API call
    try:
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=messages,
            max_tokens=2048,
            n=1,
            stop=None,
            temperature=0.7
        )
        response_content = response.choices[0].message.content.strip()

        keywords = response_content.split(', ')
        return keywords

    except Exception as e:
        print("An error occurred while connecting to the OpenAI API:",
e)
        return []

def score_matches(matches, keywords):
    scores = []
    for match in matches:
        score = sum([keyword in match['name'] for keyword in
keywords]) + \
                sum([keyword in match['description'] for keyword in
keywords])
        scores.append((match, score))
    return scores

def search_dataset_for_matches(keywords):
    matches = []
    for item in mitre_attack_data.get_techniques():
        if any(keyword in item['name'] for keyword in keywords):
            matches.append(item)
        elif 'description' in item and any(keyword in
item['description'] for keyword in keywords):
            matches.append(item)
    return matches

def generate_ttp_chain(match):
    # Create a prompt for GPT-3 to generate a TTP chain for the
provided match
    prompt = (f"Given the MITRE ATT&CK technique '{match['name']}' and
its description '{match['description']}', "
              "generate an example scenario and TTP chain
demonstrating its use.")

    # Set up the messages for the OpenAI API
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity professional with
expertise in MITRE ATT&CK techniques."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]

    # Make the API call
    try:
        client = OpenAI()
        response = client.chat.completions.create
        (
            model="gpt-3.5-turbo",
            messages=messages,
            max_tokens=2048,
            n=1,
            stop=None,
            temperature=0.7
        )
        response_content = response.choices[0].message.content.strip()
        return response_content

    except Exception as e:
        print("An error occurred while generating the TTP chain:", e)
        return "Unable to generate TTP chain."

# Sample usage:
description = input("Enter your scenario description: ")
keywords = extract_keywords_from_description(description)
matches = search_dataset_for_matches(keywords)
scored_matches = score_matches(matches, keywords)

# Sort by score in descending order and take the top 3
top_matches = sorted(scored_matches, key=lambda x: x[1], reverse=True)
[:3]

print("Top 3 matches from the MITRE ATT&CK dataset:")
for match, score in top_matches:
    print("Name:", match['name'])
    print("Summary:", match['description'])
    ttp_chain = generate_ttp_chain(match)
    print("Example Scenario and TTP Chain:", ttp_chain)
    print("-" * 50)
```

In essence, this recipe works by combining structured cybersecurity data with the flexible and expansive knowledge of ChatGPT. The Python script serves as the bridge, directing the flow of information and ensuring that the user receives detailed, relevant, and actionable red team scenarios based on their initial input.

How it works...

This recipe merges the power of the MITRE ATT&CK framework with the natural language processing abilities of ChatGPT. By doing so, it provides a unique and efficient way to generate detailed red team scenarios based on a brief description. Let's delve into the intricacies of how this merger takes place:

1. **Python and MITRE ATT&CK integration**: At its core, the Python script utilizes the `mitreattack.stix20` library to interface with the MITRE ATT&CK dataset. This dataset provides a comprehensive list of **tactics, techniques, and procedures** (**TTPs**) that adversaries might employ. By using Python, we can efficiently query this dataset and retrieve relevant information based on specific keywords or criteria.

   The `MitreAttackData("enterprise-attack.json")` method call initializes an object that provides an interface to query the MITRE ATT&CK dataset. This ensures that our script has a structured and efficient way to access the data.

2. **ChatGPT integration for keyword extraction**: The first major task where GPT comes into play is in the `extract_keywords_from_description` function. This function sends a prompt to ChatGPT to extract relevant keywords from a given scenario description. The generated prompt is designed to guide the model in not just blindly extracting keywords but thinking and expanding upon the provided description. By doing so, it can consider broader aspects of the cybersecurity domain and extract more nuanced and relevant keywords.

3. **Searching the MITRE ATT&CK dataset**: Once keywords are extracted, they are used to search the MITRE ATT&CK dataset. This search isn't merely a straightforward string match. The script looks at both the name and the description of each technique in the dataset, checking whether any of the extracted keywords are present. This dual-check increases the likelihood of getting relevant results.

4. **ChatGPT integration for scenario generation**: With the matched techniques from the MITRE ATT&CK dataset in hand, the script once again leverages ChatGPT – this time, to generate comprehensive scenarios. The `generate_ttp_chain` function is responsible for this task. It sends a prompt to ChatGPT, instructing it to summarize the technique and provide an example TTP chain scenario for it. The reason for using ChatGPT here is crucial. While the MITRE ATT&CK dataset provides detailed descriptions of techniques, it doesn't necessarily provide them in a format that's easy for non-experts to understand. By using ChatGPT, we can convert these technical descriptions into more user-friendly summaries and scenarios, making them more accessible and actionable.

5. **Ranking and selection**: The script doesn't just return all matched techniques. It ranks them based on the length of their descriptions (as a proxy for relevance and detail) and then selects the top three. This ensures that the user isn't overwhelmed with too many results and instead receives a curated list of the most pertinent techniques.

There's more...

The current script prints the detailed red team scenarios directly to the console. However, in a real-world setting, you might want to store these scenarios for future reference, share them with team members, or even use them as a basis for reporting. One straightforward way to achieve this is by writing the output to a text file.

This is how we output to a text file:

1. **Modify the Python script**:

   We'll need to slightly modify the script to incorporate the functionality to write the results to a text file. Here's how you can achieve that.

   First, add a function to write the results to a file:

   ```
   def write_to_file(matches):
       with open("red_team_scenarios.txt", "w") as file:
           for match in matches:
               file.write("Name: " + match['name'] + "\n")
               file.write("Summary: " + match['summary'] + "\n")
               file.write("Example Scenario: " + match['scenario']
   + "\n")
               file.write("-" * 50 + "\n")
   ```

   Then, after the `print` statements in the main part of the script, call this function:

   ```
   write_to_file(top_matches)
   ```

2. **Run the script**: Once you've made these modifications, run the script again. After execution, you should find a file named `red_team_scenarios.txt` in the same directory as your script. This file will contain the top three matched scenarios, formatted for easy reading.

   There are three main benefits to doing this:

- **Portability**: A text file is universally accessible, making it easy to share or move between systems
- **Documentation**: By saving scenarios, you create a record of potential threat patterns to watch out for
- **Integration with other tools**: The output file can be ingested by other cybersecurity tools for further analysis or action

This enhancement allows you to not only view the red team scenarios interactively but also to maintain a persistent record of them, enhancing the utility and applicability of the script in diverse cybersecurity contexts.

Social media and public data OSINT with ChatGPT

Open source intelligence (**OSINT**) techniques allow us to gather information from publicly available sources to support cybersecurity operations such as penetration testing. This can include scouring social media sites, public records, job postings, and more. In this recipe, we will use ChatGPT's natural language capabilities to guide us through an entire OSINT project focused on gathering intel from social media and other public data sources.

ChatGPT can serve as an AI OSINT expert, walking us through effective online reconnaissance from start to finish. The conversational nature of the process helps us learn as we go, gaining new skills and ideas. By the end, you'll have a detailed plan and methodology you can replicate for real-world engagements. OSINT is a vital skill for ethical hackers, social engineers, and cyber defenders alike.

Getting ready

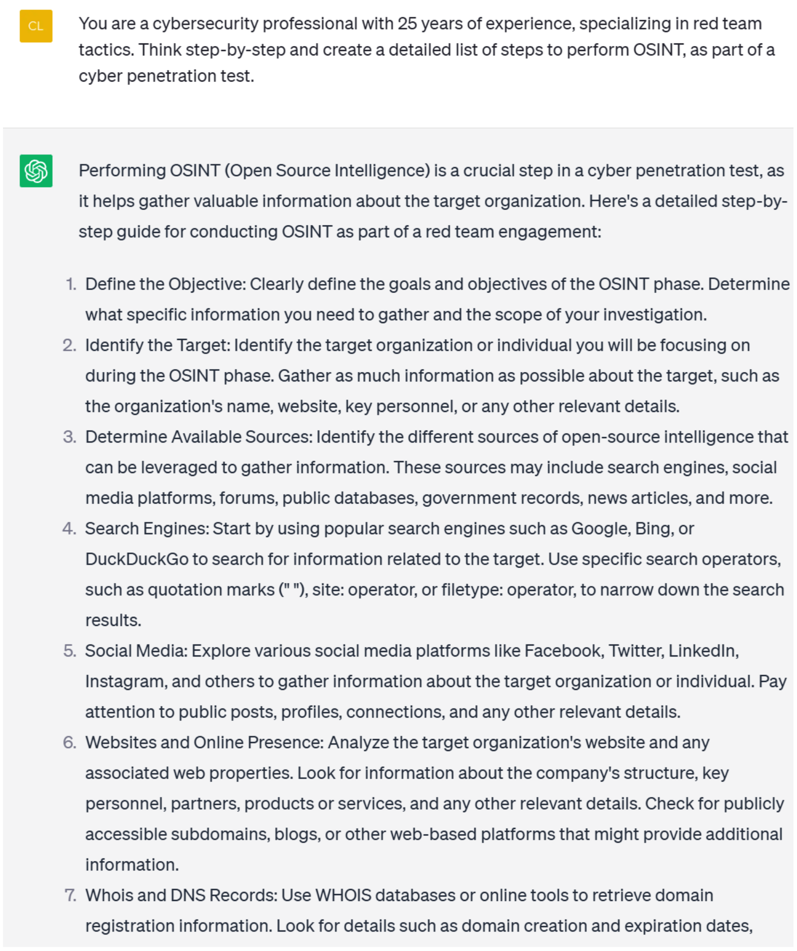
The prerequisites for this recipe are simple. You just need a web browser and an OpenAI account. If you haven't already created an account or need a refresher on how to use the ChatGPT interface, refer back to *Chapter 1* for a comprehensive guide.

How to do it…

Let's begin by having ChatGPT outline the high-level steps involved in an OSINT operation focused on social media and public data sources:

1. Establish the system role and create an OSINT process list:

```
You are a cybersecurity professional with 25 years of
experience, specializing in red team tactics. Think step-by-step
and create a detailed list of steps to perform OSINT, as part of
a cyber penetration test.
```


CL

You are a cybersecurity professional with 25 years of experience, specializing in red team tactics. Think step-by-step and create a detailed list of steps to perform OSINT, as part of a cyber penetration test.

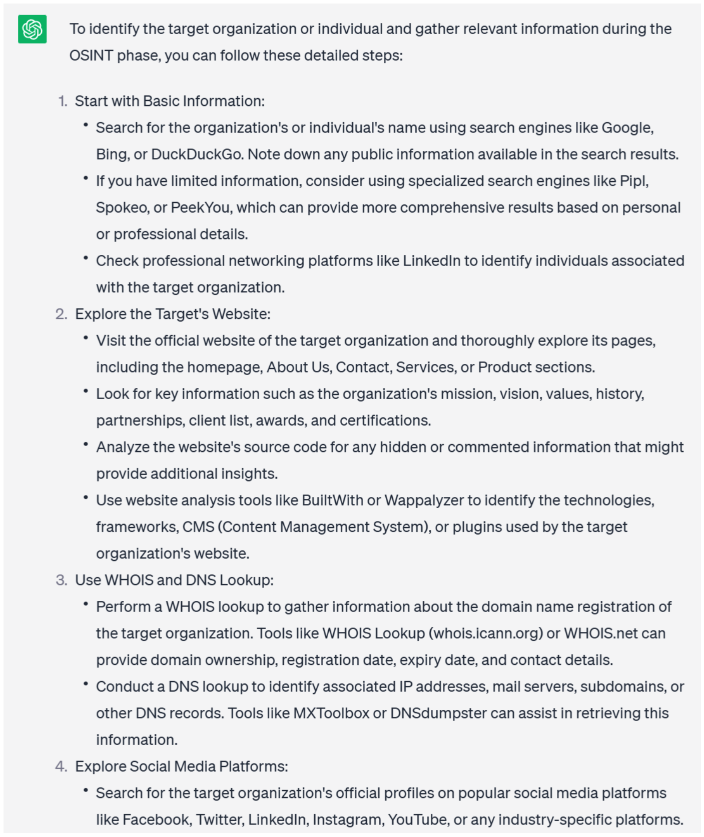
Performing OSINT (Open Source Intelligence) is a crucial step in a cyber penetration test, as it helps gather valuable information about the target organization. Here's a detailed step-by-step guide for conducting OSINT as part of a red team engagement:

1. Define the Objective: Clearly define the goals and objectives of the OSINT phase. Determine what specific information you need to gather and the scope of your investigation.
2. Identify the Target: Identify the target organization or individual you will be focusing on during the OSINT phase. Gather as much information as possible about the target, such as the organization's name, website, key personnel, or any other relevant details.
3. Determine Available Sources: Identify the different sources of open-source intelligence that can be leveraged to gather information. These sources may include search engines, social media platforms, forums, public databases, government records, news articles, and more.
4. Search Engines: Start by using popular search engines such as Google, Bing, or DuckDuckGo to search for information related to the target. Use specific search operators, such as quotation marks (" "), site: operator, or filetype: operator, to narrow down the search results.
5. Social Media: Explore various social media platforms like Facebook, Twitter, LinkedIn, Instagram, and others to gather information about the target organization or individual. Pay attention to public posts, profiles, connections, and any other relevant details.
6. Websites and Online Presence: Analyze the target organization's website and any associated web properties. Look for information about the company's structure, key personnel, partners, products or services, and any other relevant details. Check for publicly accessible subdomains, blogs, or other web-based platforms that might provide additional information.
7. Whois and DNS Records: Use WHOIS databases or online tools to retrieve domain registration information. Look for details such as domain creation and expiration dates,

Figure 6.1 – Example OSINT process steps output from ChatGPT

2. Next, we will take the first step identified by ChatGPT and expand on it:

```
"You are a cybersecurity professional with 25 years of
experience, specializing in red team tactics. Think step-by-step
and provide more detail on how I achieve the following. Include
any applicable tools, apps, and resources.
Identify the Target: Identify the target organization or
individual you will be focusing on during the OSINT phase.
Gather as much information as possible about the target, such as
the organization's name, website, key personnel, or any other
relevant details."
```


To identify the target organization or individual and gather relevant information during the OSINT phase, you can follow these detailed steps:

1. Start with Basic Information:
   - Search for the organization's or individual's name using search engines like Google, Bing, or DuckDuckGo. Note down any public information available in the search results.
   - If you have limited information, consider using specialized search engines like Pipl, Spokeo, or PeekYou, which can provide more comprehensive results based on personal or professional details.
   - Check professional networking platforms like LinkedIn to identify individuals associated with the target organization.
2. Explore the Target's Website:
   - Visit the official website of the target organization and thoroughly explore its pages, including the homepage, About Us, Contact, Services, or Product sections.
   - Look for key information such as the organization's mission, vision, values, history, partnerships, client list, awards, and certifications.
   - Analyze the website's source code for any hidden or commented information that might provide additional insights.
   - Use website analysis tools like BuiltWith or Wappalyzer to identify the technologies, frameworks, CMS (Content Management System), or plugins used by the target organization's website.
3. Use WHOIS and DNS Lookup:
   - Perform a WHOIS lookup to gather information about the domain name registration of the target organization. Tools like WHOIS Lookup (whois.icann.org) or WHOIS.net can provide domain ownership, registration date, expiry date, and contact details.
   - Conduct a DNS lookup to identify associated IP addresses, mail servers, subdomains, or other DNS records. Tools like MXToolbox or DNSdumpster can assist in retrieving this information.
4. Explore Social Media Platforms:
   - Search for the target organization's official profiles on popular social media platforms like Facebook, Twitter, LinkedIn, Instagram, YouTube, or any industry-specific platforms.

Figure 6.2 – Example OSINT details output from ChatGPT

3. Repeat the second prompt for each remaining step in the OSINT outline provided originally. This will expand each high-level step into a detailed process with tools and tactical advice from ChatGPT.

Once all steps have been expanded on, you will have a comprehensive methodology for executing a social media and public data-focused OSINT operation.

How it works…

The key to this technique is establishing ChatGPT as an experienced OSINT expert upfront. This frames the conversational prompts in a way that guides the model to provide detailed, practical responses on conducting online reconnaissance.

Asking ChatGPT to “think step-by-step” further tunes the output, resulting in orderly, logical processes. We first have it outline the overall workflow, providing high-level steps.

Then, by taking each step and providing it back to ChatGPT as a prompt asking for more details, we essentially get into the specifics of how to execute each phase.

This leverages ChatGPT's knowledge bank on OSINT tradecraft and elicits tailored recommendations powered by its natural language processing capabilities. The result is an expert-guided OSINT methodology customized to our goal.

There's more...

The beauty of this technique is that the "recursion" can be taken even further. If any single step's explanation from ChatGPT contains additional high-level tasks, those can be further expanded by repeating the process.

For example, ChatGPT may mention "Use Google Dorks to find public records." This could be provided back to ChatGPT as another prompt asking for more details on which operators and strategies to use.

By recursively "zooming in" on details in this way, you can extract an immense amount of practical advice from ChatGPT to build a comprehensive guide. The model can also suggest tools, techniques, and ideas you may have never considered before!

Google Dork automation with ChatGPT and Python

Google Dorks are a powerful tool in the arsenal of penetration testers, ethical hackers, and even malicious actors. These specially crafted search queries leverage advanced Google search operators to uncover information or vulnerabilities that are unintentionally exposed on the web. From finding open directories to exposed configuration files, Google Dorks can reveal a treasure trove of information, often inadvertently published.

However, crafting effective Google Dorks requires expertise, and manually searching for each dork can be time-consuming. This is where the combination of ChatGPT and Python shines. By utilizing the linguistic capabilities of ChatGPT, we can automate the generation of Google Dorks tailored to specific requirements. Python then takes over, using these dorks to initiate searches and organize the results for further analysis.

In this recipe, we leverage ChatGPT to generate a series of Google Dorks that are designed to unearth valuable data during penetration tests. We then employ Python to apply these dorks systematically, producing a consolidated view of potential vulnerabilities or exposed information regarding a target. This approach not only amplifies the efficiency of the pentesting process but also ensures a comprehensive sweep of the digital footprint of the target. Whether you're a seasoned penetration tester looking to streamline your reconnaissance phase or a cybersecurity enthusiast keen on exploring Google Dorks, this recipe provides a practical, automated approach to harnessing the power of Google's search engine for security assessments.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**, and the following libraries:

- `openai`: This library enables you to interact with the OpenAI API. Install it using the `pip install openai` command.
- `requests`: This library is essential for making HTTP requests. Install it using `pip install requests`.
- `time`: This is a built-in Python library used for various time-related tasks.

Additionally, you'll need to set up a **Google API key** and a **custom search engine ID**, which can be done at `https://console.cloud.google.com/` and `https://cse.google.com/cse/all`.

With these requirements in place, you're prepared to delve into the script.

How to do it…

Google Dorks are incredibly potent when it comes to uncovering exposed data or vulnerabilities on the web. While they can be run manually, automating this process can significantly boost efficiency and comprehensiveness. In this section, we will guide you through the steps of using Python to automate the application of Google Dorks, fetching search results, and saving them for further analysis.

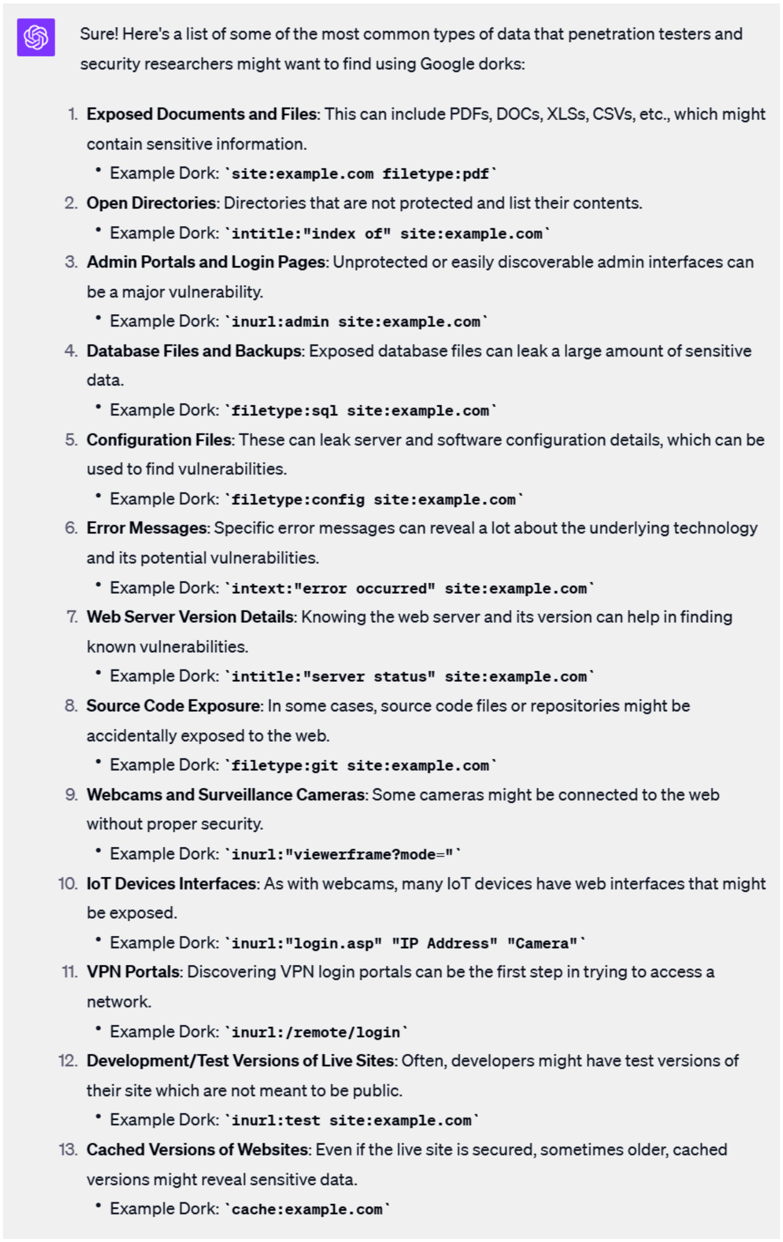
First, let's generate a list of Google Dorks:

1. **Generate a series of Google Dorks**: To do this, give ChatGPT a clear objective. Use the following prompt with ChatGPT:

   ```
   "You are a cybersecurity professional specializing in red team
   tactics. I am a cybersecurity professional and I have a scenario
   where I need to find exposed documents on a my own domain.
   Please provide a list of example Google dorks that I can use to
   discover such vulnerabilities as part of an authorized exercise
   on my own authorized domain."
   ```

   By giving ChatGPT the scenario and purpose, ChatGPT is less likely to reject the prompt, thinking it is being asked to provide something unethical.

This is an example output:



Sure! Here's a list of some of the most common types of data that penetration testers and security researchers might want to find using Google dorks:

1. **Exposed Documents and Files**: This can include PDFs, DOCs, XLSs, CSVs, etc., which might contain sensitive information.
   - Example Dork: `` `site:example.com filetype:pdf` ``
2. **Open Directories**: Directories that are not protected and list their contents.
   - Example Dork: `` `intitle:"index of" site:example.com` ``
3. **Admin Portals and Login Pages**: Unprotected or easily discoverable admin interfaces can be a major vulnerability.
   - Example Dork: `` `inurl:admin site:example.com` ``
4. **Database Files and Backups**: Exposed database files can leak a large amount of sensitive data.
   - Example Dork: `` `filetype:sql site:example.com` ``
5. **Configuration Files**: These can leak server and software configuration details, which can be used to find vulnerabilities.
   - Example Dork: `` `filetype:config site:example.com` ``
6. **Error Messages**: Specific error messages can reveal a lot about the underlying technology and its potential vulnerabilities.
   - Example Dork: `` `intext:"error occurred" site:example.com` ``
7. **Web Server Version Details**: Knowing the web server and its version can help in finding known vulnerabilities.
   - Example Dork: `` `intitle:"server status" site:example.com` ``
8. **Source Code Exposure**: In some cases, source code files or repositories might be accidentally exposed to the web.
   - Example Dork: `` `filetype:git site:example.com` ``
9. **Webcams and Surveillance Cameras**: Some cameras might be connected to the web without proper security.
   - Example Dork: `` `inurl:"viewerframe?mode="` ``
10. **IoT Devices Interfaces**: As with webcams, many IoT devices have web interfaces that might be exposed.
    - Example Dork: `` `inurl:"login.asp" "IP Address" "Camera"` ``
11. **VPN Portals**: Discovering VPN login portals can be the first step in trying to access a network.
    - Example Dork: `` `inurl:/remote/login` ``
12. **Development/Test Versions of Live Sites**: Often, developers might have test versions of their site which are not meant to be public.
    - Example Dork: `` `inurl:test site:example.com` ``
13. **Cached Versions of Websites**: Even if the live site is secured, sometimes older, cached versions might reveal sensitive data.
    - Example Dork: `` `cache:example.com` ``

Figure 6.3 – Example ChatGPT output for a list of Google Dorks

Next, let's generate the Python script to automate the Google Dork execution.

2. **Import the necessary libraries**: In this case, we will need `requests` and `time`:

```
import requests
import time
```

3. **Set up the prerequisites**: To utilize Google's Custom Search JSON API, you need to set it up and get the necessary credentials:

```
API_KEY = 'YOUR_GOOGLE_API_KEY'
CSE_ID = 'YOUR_CUSTOM_SEARCH_ENGINE_ID'
SEARCH_URL = "https://www.googleapis.com/customsearch/
v1?q={query}&key={api_key}&cx={cse_id}"
```

Replace `'YOUR_GOOGLE_API_KEY'` with your API key and `'YOUR_CUSTOM_SEARCH_ENGINE_ID'` with your custom search engine ID. These are vital for your script to communicate with Google's API.

4. **List the Google Dorks**: Craft or gather the list of Google Dorks you want to run. For our example, we've provided a sample list targeting `'example.com'`:

```
dorks = [
    'site:example.com filetype:pdf',
    'intitle:"index of" site:example.com',
    'inurl:admin site:example.com',
    'filetype:sql site:example.com',
    # ... add other dorks here ...
]
```

You can extend this list with any additional Dorks relevant to your pentesting objectives.

5. **Fetch the search results**: Create a function to fetch Google search results using the provided Dork:

```
def get_search_results(query):
    """Fetch the Google search results."""
    response = requests.get(SEARCH_URL.format(query=query, api_
key=API_KEY, cse_id=CSE_ID))
    if response.status_code == 200:
        return response.json()
    else:
        print("Error:", response.status_code)
        return {}
```

This function sends a request to **Google's Custom Search API** with the Dork as the query and returns the search results.

6. **Iterate through Dorks and fetch and save results**: This is the core of your automation. Here, we loop through each Google Dork, fetch its results, and save them in a text file:

```
def main():
    with open("dork_results.txt", "a") as outfile:
        for dork in dorks:
            print(f"Running dork: {dork}")
            results = get_search_results(dork)

            if 'items' in results:
                for item in results['items']:
                    print(item['title'])
                    print(item['link'])
                    outfile.write(item['title'] + "\n")
                    outfile.write(item['link'] + "\n")
                    outfile.write("-" * 50 + "\n")
            else:
                print("No results found or reached API limit!")

            # To not hit the rate limit, introduce a delay
between requests
            time.sleep(20)
```

   This simple piece of code ensures that when you run the script, the `main` function, which contains our core logic, gets executed.

Important note

Remember, Google's API might have rate limits. We've introduced a delay in our loop to prevent hitting these limits too quickly. Adjustments might be required based on your API's specific rate limits.

Here is how the completed script should look:

```
import requests
import time

# Google Custom Search JSON API configuration
API_KEY = 'YOUR_GOOGLE_API_KEY'
CSE_ID = 'YOUR_CUSTOM_SEARCH_ENGINE_ID'
SEARCH_URL = "https://www.googleapis.com/customsearch/
v1?q={query}&key={api_key}&cx={cse_id}"

# List of Google dorks
dorks = [
    'site:example.com filetype:pdf',
    'intitle:"index of" site:example.com',
    'inurl:admin site:example.com',
    'filetype:sql site:example.com',
    # ... add other dorks here ...
]

def get_search_results(query):
    """Fetch the Google search results."""
    response = requests.get(SEARCH_URL.format(query=query, api_
key=API_KEY, cse_id=CSE_ID))
    if response.status_code == 200:
        return response.json()
    else:
        print("Error:", response.status_code)
        return {}

def main():
    with open("dork_results.txt", "a") as outfile:
        for dork in dorks:
            print(f"Running dork: {dork}")
            results = get_search_results(dork)

            if 'items' in results:
                for item in results['items']:
                    print(item['title'])
                    print(item['link'])
                    outfile.write(item['title'] + "\n")
                    outfile.write(item['link'] + "\n")
                    outfile.write("-" * 50 + "\n")
            else:
                print("No results found or reached API limit!")

            # To not hit the rate limit, introduce a delay between
requests
            time.sleep(20)

if __name__ == '__main__':
    main()
```

This script harnesses the power of both Python (for automation) and ChatGPT (for the initial expertise to create the list) to create an efficient and comprehensive tool for Google Dorking, a valuable method in the arsenal of penetration testers.

How it works...

Understanding the mechanics behind this script will empower you to adapt and optimize it according to your requirements. Let's delve into the intricacies of how this automated Google Dorking script functions:

Python scripting:

1. **API and URL configuration**:

```
API_KEY = 'YOUR_GOOGLE_API_KEY'
CSE_ID = 'YOUR_CUSTOM_SEARCH_ENGINE_ID'
SEARCH_URL = https://www.googleapis.com/customsearch/
v1?q={query}&key={api_key}&cx={cse_id}
```

   The script starts by defining constants for the Google API key, custom search engine ID, and URL endpoint for search requests. These constants are vital for making authenticated API calls to Google and retrieving search results.

2. **Fetching search results**: The `get_search_results` function uses the `requests.get()` method to send a `GET` request to the Google Custom Search JSON API. By formatting the URL with the query (Google Dork), API key, and custom search engine ID, the function retrieves search results for the specified Dork. The results are then parsed as JSON.
3. **Iterating and storing**: The `main` function is where the script iterates over each Google Dork in the list. For each Dork, it fetches the search results using the function mentioned previously and writes the title and link of each result to both the console and a `dork_results.txt` text file. This ensures that you have a persistent record of your findings.
4. **Rate limiting**: To avoid hitting Google's API rate limits, the script includes a `time.sleep(20)` statement, which introduces a 20-second delay between successive API calls. This is crucial, as sending too many requests in a short span can lead to temporary IP bans or API restrictions.

GPT prompts:

1. **Crafting the prompt**: The initial step involves creating a prompt that instructs the GPT model to generate a list of Google Dorks. The prompt is specifically designed to provide the model with a clear and concise directive, along with a purpose and scenario, so that ChatGPT doesn't reject the prompt (due to safety measures preventing unethical activity).

There's more...

While the core recipe provides a foundational approach to leveraging Google Dorks for penetration testing, truly mastering this domain requires diving into a deeper layer of complexities and nuances. The additional enhancements and suggestions provided in this section might necessitate a more advanced understanding of both penetration testing and programming. Venturing beyond the scope of this basic recipe can open up a wealth of possibilities for more in-depth vulnerability discovery and analysis. If you're looking to elevate your penetration testing capabilities, extending this recipe with these add-ons can offer more comprehensive insights, more refined results, and a higher degree of automation. However, always approach with caution, ensuring you maintain ethical practices and have the necessary permissions when probing systems and networks:

1. **Refinement of Dorks**: While the initial prompt provided a basic list of Dorks, it's always a good idea to customize and refine these queries based on the specific target or domain you're working with. For instance, if you're specifically interested in SQL vulnerabilities, you might want to expand your list with more SQL-specific Dorks.
2. **Integration with other search engines**: Google isn't the only game in town. Consider expanding the script to work with other search engines such as Bing or DuckDuckGo. Each search engine might index websites differently, giving you a broader range of potential vulnerabilities.
3. **Automated analysis**: Once you have the results, you might want to implement a post-processing step. This could involve checking the legitimacy of vulnerabilities, sorting them based on potential impact, or even integrating with tools that can automate the exploitation of found vulnerabilities.
4. **Notifications**: Depending on the scope of your penetration test, you might be running many Dorks, and analyzing them all can be time-consuming. Consider adding a feature that sends notifications (maybe through email or a messenger bot) when a particularly high-value vulnerability is detected.
5. **Visual dashboard**: Presenting the results in a more visual format, such as a dashboard, can be beneficial, especially if you're reporting to stakeholders. There are Python libraries such as Dash or even integration with tools such as Grafana that can help present your findings in a more digestible manner.
6. **Rate limiting and proxies**: If you're making a lot of requests, not only might you hit API rate limits but you might also end up getting IP banned. Consider integrating proxy rotation in the script to distribute the requests across different IP addresses.
7. **Ethical considerations**: Always remember to use Google Dorks responsibly and ethically. Never use them to exploit vulnerabilities on systems you do not have permission to test. Additionally, be aware of the terms of service for both Google and the Google Cloud API. Over-reliance or misuse can lead to API key suspension or other penalties.

Analyzing job postings OSINT with ChatGPT

OSINT refers to the practice of collecting and analyzing information that is publicly available. In the realm of cybersecurity, OSINT serves as a valuable tool, offering insights into potential vulnerabilities, threats, and targets within an organization. Among the myriad sources of OSINT, company job listings stand out as a particularly rich trove of data. At first glance, job listings are innocuous, aiming to attract potential candidates by detailing the responsibilities, qualifications, and benefits associated with a position. However, these descriptions often inadvertently disclose far more than intended.

For instance, a job listing seeking an expert in a specific version of software might reveal the exact technologies a company uses, potentially highlighting known vulnerabilities in that software. Similarly, listings that mention proprietary technologies or in-house tools can give hints about a company's unique technological landscape. Job ads might also detail team structures, revealing hierarchies and key roles, which can be exploited for social engineering attacks. Furthermore, geographic locations, departmental interactions, and even the tone of the job listing can provide astute observers with insights into a company's culture, size, and operational focus.

Understanding these nuances, this recipe guides you on how to harness the capabilities of ChatGPT to meticulously analyze job listings. By doing so, you can extract valuable OSINT data, which can then be structured and presented in a comprehensive report format.

Getting ready

The prerequisites for this recipe are simple. You just need a web browser and an OpenAI account. If you haven't already created an account or need a refresher on how to use the ChatGPT interface, refer back to *Chapter 1* for a comprehensive guide.

How to do it...

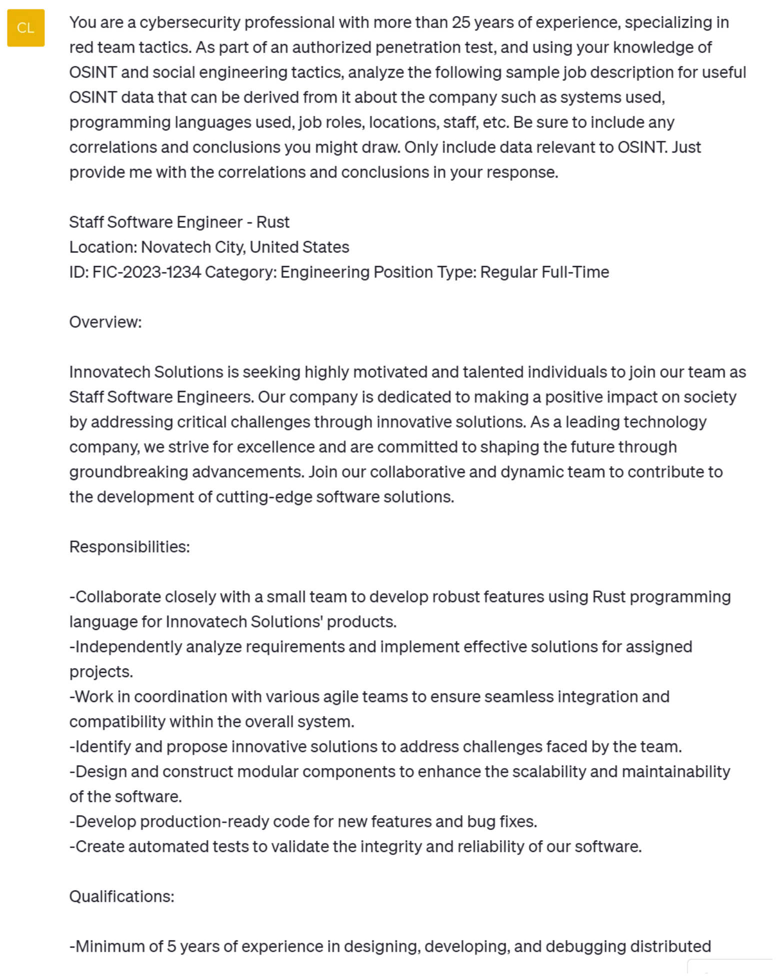
Before diving into the step-by-step instructions, it's essential to understand that the quality and depth of OSINT data derived will vary based on the richness of the job description. Remember, while this method provides valuable insights, always ensure you are authorized to perform any intelligence gathering or penetration testing.

First, we need to analyze the job description:

1. Prepare the prompt for the initial OSINT analysis:

```
You are a cybersecurity professional with more than 25 years
of experience, specializing in red team tactics. As part of an
authorized penetration test, and using your knowledge of OSINT
and social engineering tactics, analyze the following sample
job description for useful OSINT data. Be sure to include any
correlations and conclusions you might draw.
```

2. Provide the job description data. Append the job description to the prompt, ensuring clear separation:



CL

You are a cybersecurity professional with more than 25 years of experience, specializing in red team tactics. As part of an authorized penetration test, and using your knowledge of OSINT and social engineering tactics, analyze the following sample job description for useful OSINT data that can be derived from it about the company such as systems used, programming languages used, job roles, locations, staff, etc. Be sure to include any correlations and conclusions you might draw. Only include data relevant to OSINT. Just provide me with the correlations and conclusions in your response.

Staff Software Engineer - Rust
Location: Novatech City, United States
ID: FIC-2023-1234 Category: Engineering Position Type: Regular Full-Time

Overview:

Innovatech Solutions is seeking highly motivated and talented individuals to join our team as Staff Software Engineers. Our company is dedicated to making a positive impact on society by addressing critical challenges through innovative solutions. As a leading technology company, we strive for excellence and are committed to shaping the future through groundbreaking advancements. Join our collaborative and dynamic team to contribute to the development of cutting-edge software solutions.

Responsibilities:

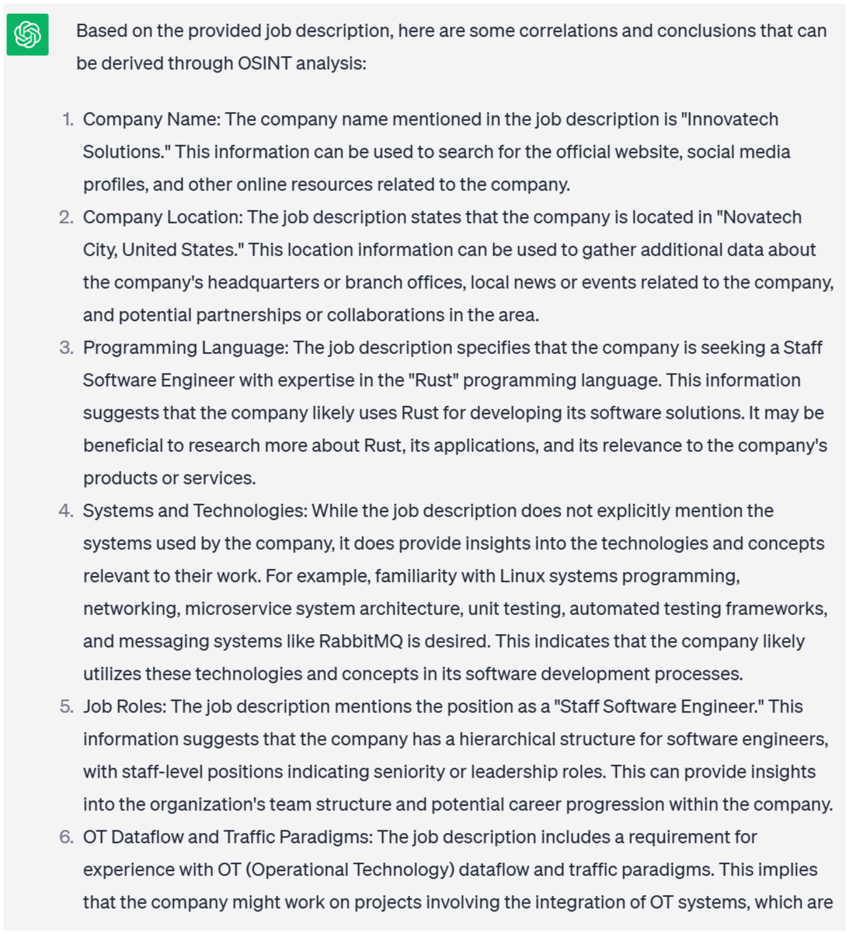
-Collaborate closely with a small team to develop robust features using Rust programming language for Innovatech Solutions' products.
-Independently analyze requirements and implement effective solutions for assigned projects.
-Work in coordination with various agile teams to ensure seamless integration and compatibility within the overall system.
-Identify and propose innovative solutions to address challenges faced by the team.
-Design and construct modular components to enhance the scalability and maintainability of the software.
-Develop production-ready code for new features and bug fixes.
-Create automated tests to validate the integrity and reliability of our software.

Qualifications:

-Minimum of 5 years of experience in designing, developing, and debugging distributed

Figure 6.4 – Example prompt with the job posting appended to it

3. Analyze the results. Send the combined prompt and data to ChatGPT and review the derived OSINT data:



Based on the provided job description, here are some correlations and conclusions that can be derived through OSINT analysis:

1. Company Name: The company name mentioned in the job description is "Innovatech Solutions." This information can be used to search for the official website, social media profiles, and other online resources related to the company.
2. Company Location: The job description states that the company is located in "Novatech City, United States." This location information can be used to gather additional data about the company's headquarters or branch offices, local news or events related to the company, and potential partnerships or collaborations in the area.
3. Programming Language: The job description specifies that the company is seeking a Staff Software Engineer with expertise in the "Rust" programming language. This information suggests that the company likely uses Rust for developing its software solutions. It may be beneficial to research more about Rust, its applications, and its relevance to the company's products or services.
4. Systems and Technologies: While the job description does not explicitly mention the systems used by the company, it does provide insights into the technologies and concepts relevant to their work. For example, familiarity with Linux systems programming, networking, microservice system architecture, unit testing, automated testing frameworks, and messaging systems like RabbitMQ is desired. This indicates that the company likely utilizes these technologies and concepts in its software development processes.
5. Job Roles: The job description mentions the position as a "Staff Software Engineer." This information suggests that the company has a hierarchical structure for software engineers, with staff-level positions indicating seniority or leadership roles. This can provide insights into the organization's team structure and potential career progression within the company.
6. OT Dataflow and Traffic Paradigms: The job description includes a requirement for experience with OT (Operational Technology) dataflow and traffic paradigms. This implies that the company might work on projects involving the integration of OT systems, which are

Figure 6.5 – Example ChatGPT output analysis

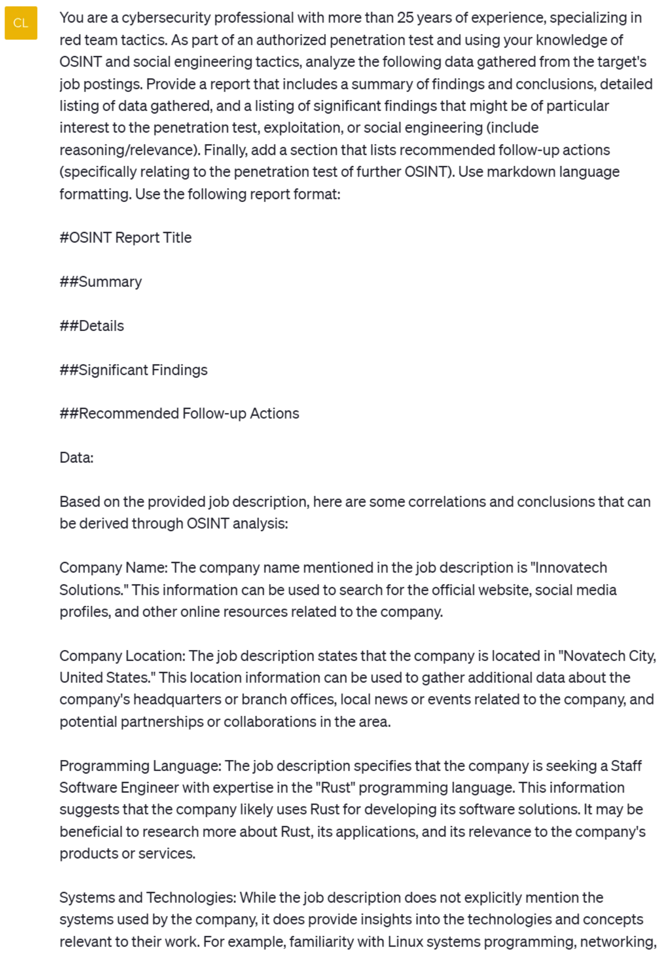
Now we have analyzed the results, we can generate a structured OSINT report.

4. Prepare the next prompt for report generation:

```
You are a cybersecurity professional with more than 25 years
of experience, specializing in red team tactics. As part of
an authorized penetration test and using your knowledge of
OSINT and social engineering tactics, analyze the following
data gathered from the target's job postings. Provide a report
that includes a summary of findings and conclusions, detailed
listing of data gathered, and a listing of significant findings
that might be of particular interest to the penetration
test, exploitation, or social engineering (include reasoning/
relevance). Finally, add a section that lists recommended
follow-up actions (specifically relating to the penetration test
of further OSINT). Use markdown language formatting. Use the
```

```
following report format:

#OSINT Report Title

##Summary

##Details

##Significant Findings

##Recommended Follow-up Actions
```

5. Provide the OSINT analysis data. Append the summarized OSINT findings from the previous step to the prompt:


CL

You are a cybersecurity professional with more than 25 years of experience, specializing in red team tactics. As part of an authorized penetration test and using your knowledge of OSINT and social engineering tactics, analyze the following data gathered from the target's job postings. Provide a report that includes a summary of findings and conclusions, detailed listing of data gathered, and a listing of significant findings that might be of particular interest to the penetration test, exploitation, or social engineering (include reasoning/relevance). Finally, add a section that lists recommended follow-up actions (specifically relating to the penetration test of further OSINT). Use markdown language formatting. Use the following report format:

#OSINT Report Title

##Summary

##Details

##Significant Findings

##Recommended Follow-up Actions

Data:

Based on the provided job description, here are some correlations and conclusions that can be derived through OSINT analysis:

Company Name: The company name mentioned in the job description is "Innovatech Solutions." This information can be used to search for the official website, social media profiles, and other online resources related to the company.

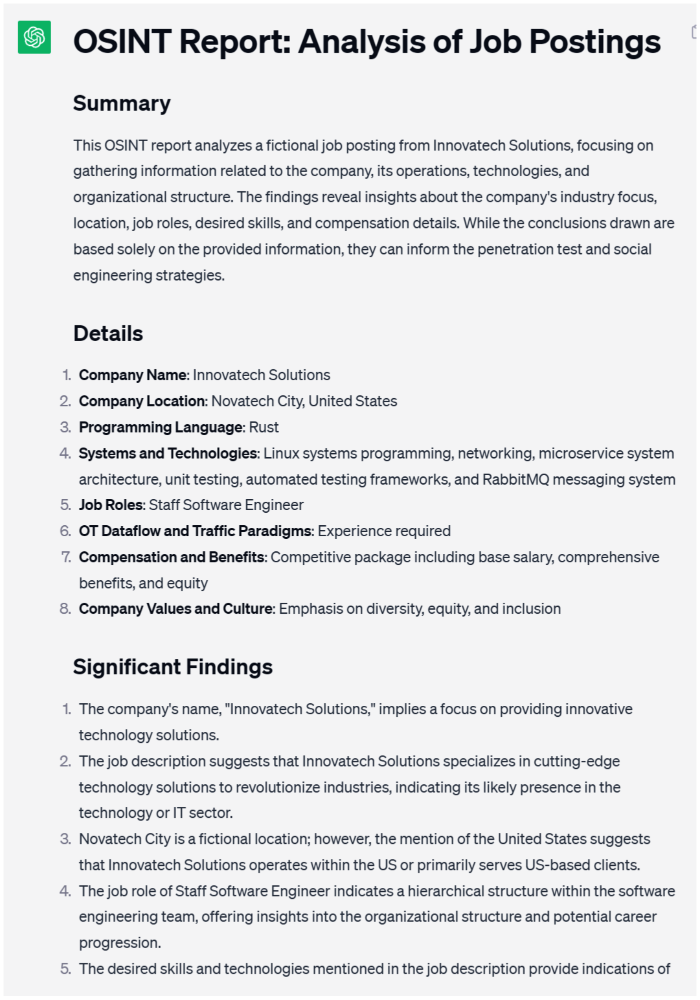
Company Location: The job description states that the company is located in "Novatech City, United States." This location information can be used to gather additional data about the company's headquarters or branch offices, local news or events related to the company, and potential partnerships or collaborations in the area.

Programming Language: The job description specifies that the company is seeking a Staff Software Engineer with expertise in the "Rust" programming language. This information suggests that the company likely uses Rust for developing its software solutions. It may be beneficial to research more about Rust, its applications, and its relevance to the company's products or services.

Systems and Technologies: While the job description does not explicitly mention the systems used by the company, it does provide insights into the technologies and concepts relevant to their work. For example, familiarity with Linux systems programming, networking,

Figure 6.6 – Example prompt with the first job data analysis appended to it

6. Now, we can generate the report. Submit the combined prompt and OSINT analysis data to ChatGPT to receive a structured report in Markdown format:



OSINT Report: Analysis of Job Postings

Summary

This OSINT report analyzes a fictional job posting from Innovatech Solutions, focusing on gathering information related to the company, its operations, technologies, and organizational structure. The findings reveal insights about the company's industry focus, location, job roles, desired skills, and compensation details. While the conclusions drawn are based solely on the provided information, they can inform the penetration test and social engineering strategies.

Details

1. **Company Name**: Innovatech Solutions
2. **Company Location**: Novatech City, United States
3. **Programming Language**: Rust
4. **Systems and Technologies**: Linux systems programming, networking, microservice system architecture, unit testing, automated testing frameworks, and RabbitMQ messaging system
5. **Job Roles**: Staff Software Engineer
6. **OT Dataflow and Traffic Paradigms**: Experience required
7. **Compensation and Benefits**: Competitive package including base salary, comprehensive benefits, and equity
8. **Company Values and Culture**: Emphasis on diversity, equity, and inclusion

Significant Findings

1. The company's name, "Innovatech Solutions," implies a focus on providing innovative technology solutions.
2. The job description suggests that Innovatech Solutions specializes in cutting-edge technology solutions to revolutionize industries, indicating its likely presence in the technology or IT sector.
3. Novatech City is a fictional location; however, the mention of the United States suggests that Innovatech Solutions operates within the US or primarily serves US-based clients.
4. The job role of Staff Software Engineer indicates a hierarchical structure within the software engineering team, offering insights into the organizational structure and potential career progression.
5. The desired skills and technologies mentioned in the job description provide indications of

Figure 6.7 – Example ChatGPT output with structured formatting

How it works…

The process is divided into two primary steps – OSINT extraction from job listings and structured report generation:

1. **Job description analysis**: The first prompt guides ChatGPT to focus on extracting OSINT data from job listings. The key here is the role assignment, which ensures the model adopts the perspective of a seasoned cybersecurity professional, allowing for a more insightful analysis.

2. **Report generation**: The second prompt takes the OSINT findings and structures them into a detailed report. Again, the role assignment is crucial. It ensures that ChatGPT understands the context and delivers the report in a manner suitable for a cybersecurity professional. The use of Markdown formatting ensures the report is structured, clear, and easy to read.

In both steps, the prompts are engineered to give ChatGPT the right context. By explicitly instructing the model on the desired outcome and the role it should adopt, we ensure the results are tailored to the needs of cybersecurity OSINT analysis.

In conclusion, this recipe illustrates how ChatGPT can be an invaluable tool for cybersecurity professionals, simplifying the process of OSINT extraction and report generation from job listings.

There's more...

OSINT analysis of job listings is just the tip of the iceberg when it comes to understanding a company's digital footprint. Here are some additional ways to further enhance and expand on this recipe:

1. **Multiple data sources**: While job listings can provide a wealth of information, considering other public-facing documents such as press releases, annual reports, and official blogs can yield even more OSINT data. Aggregating and cross-referencing data from multiple sources can lead to more comprehensive insights.
2. **Automate data gathering**: Instead of manually gathering job listings, consider building a web scraper or using APIs (if available) to automatically fetch new job listings from targeted companies. This allows for continuous monitoring and timely analysis.

Important note

We didn't include automated web scraping here due to the current controversy regarding LLMs and web scraping. These techniques are fine during an authorized penetration test as long as you have permission to do so.

3. **Temporal analysis**: Analyzing job listings over time can provide insights into a company's growth areas, shifts in technology stacks, or expansion into new domains. For instance, a sudden increase in hiring cloud security professionals might indicate a move to cloud platforms.
4. **Integration with other OSINT tools**: There are many OSINT tools and platforms available that can complement the insights gained from job listings. Integrating this method with other tools can provide a more holistic view of a target.
5. **Ethical considerations**: Always ensure that any OSINT gathering activity is done ethically and legally. Remember that while the information might be publicly available, how it's used can have legal and ethical implications.

In conclusion, while analyzing job listings is a potent method in the OSINT toolkit, combining it with other techniques and data sources can significantly enhance its value. As always, the key is to be thorough and ethical and stay updated on the latest trends and tools in the OSINT domain.

GPT-powered Kali Linux terminals

Navigating and mastering the command line of any **Linux** distribution, especially security-focused ones such as Kali Linux, can be a daunting task. For beginners, there's a steep learning curve as they have to memorize various commands, switches, and syntax to accomplish even basic tasks. For experienced professionals, while they may be familiar with many commands, constructing complex command strings on the fly can sometimes be time-consuming. Enter the power of **natural language processing** (**NLP**) and the capabilities of OpenAI's GPT models.

In this recipe, we present an innovative approach to interacting with your Linux terminal: an NLP-powered terminal interface. This script harnesses the capabilities of OpenAI's GPT model to allow users to input requests in natural language. In return, the model deciphers the intent and translates it into the appropriate command for the Linux operating system. For instance, instead of remembering the intricate syntax for certain operations, a user could simply input `Show me all the files modified in the last 24 hours`, and the model would generate and execute the appropriate `find` command.

This approach provides numerous benefits:

- **User-friendly**: Beginners can start performing complex operations without the need for deep command-line knowledge. It lowers the barrier to entry and accelerates the learning curve.
- **Efficiency**: Even for experienced users, this can speed up workflows. Instead of recalling specific flags or syntax, a simple sentence can generate the needed command.
- **Flexibility**: It's not just limited to OS commands. This approach can be extended to applications within the OS, from networking tools to cybersecurity utilities in distributions such as Kali Linux.
- **Logging**: Every command generated by the model is logged, providing an audit trail and a way to learn the actual commands over time.

By the end of this recipe, you'll have a terminal interface that feels more like a conversation with a Linux expert, guiding you and executing tasks on your behalf, powered by the advanced NLP capabilities of GPT models.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

- `openai`: This library enables you to interact with the OpenAI API. Install it using the `pip install openai` command.
- `os`: This is a built-in Python library that allows you to interact with the operating system, especially for accessing environment variables.
- `subprocess`: This library is a built-in Python library that allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

Once these requirements are in place, you are all set to dive into the script.

How to do it…

To construct a GPT-powered terminal, we'll leverage the OpenAI API to interpret natural language input and generate the corresponding Linux command. This fusion of advanced NLP with the OS's capabilities offers a unique and enhanced user experience, especially for those who may not be familiar with intricate Linux commands. Follow this step-by-step guide to integrate this functionality into your Linux system:

1. **Setting up your environment**: Before diving into the code, ensure you have Python installed and the necessary libraries available. If not, you can easily install them using `pip`:

   ```
   import openai
   from openai import OpenAI
   import os
   import subprocess
   ```

2. **Storing the OpenAI API key**: To interact with the OpenAI API, you'll need your API key. For security reasons, it's a good practice not to hardcode this key directly in the script. Instead, we're storing it in a file called `openai-key.txt`:

   ```
   def open_file(filepath): #Open and read a file
       with open(filepath, 'r', encoding='UTF-8') as infile:
           return infile.read()
   ```

   This function reads the content of a file. In our case, it retrieves the API key from `openai-key.txt`.

3. **Sending requests to the OpenAI API**:

   Create a function that sets up the request to the OpenAI API and retrieves the output:

   ```
   def gpt_3(prompt):
       try:
           client = OpenAI()
           response = client.chat.completions.create(
   ```

```
            model="gpt-3.5-turbo",
            prompt=prompt,
            temperature=0.1,
            max_tokens=600,
        )
        text = response.choices[0].message.content.strip()
        return text
    except openai.error.APIError as e:
        print(f"\nError communicating with the API.")
        print(f"\nError: {e}")
        print("\nRetrying...")
        return gpt_3(prompt)
```

This function sends a prompt to the OpenAI GPT model and fetches the corresponding output.

4. **Running the command**: Use the Python `subprocess` library to execute the command generated by the OpenAI API on your Linux system:

```
process = subprocess.Popen(command, shell=True,
stdout=subprocess.PIPE, bufsize=1, universal_newlines=True)
```

This piece of code initializes a new subprocess, runs the command, and provides real-time feedback to the user.

5. **Continuous interaction loop**: To keep the NLP terminal running and accepting continuous user input, implement a `while` loop:

```
while True:
    request = input("\nEnter request: ")
    if not request:
        break
    if request == "quit":
        break
    prompt = open_file("prompt4.txt").replace('{INPUT}',
request)
    command = gpt_3(prompt)
    process = subprocess.Popen(command, shell=True,
stdout=subprocess.PIPE, bufsize=1, universal_newlines=True)
    print("\n" + command + "\n")
    with process:
        for line in process.stdout:
            print(line, end='', flush=True)

    exit_code = process.wait()
```

This loop ensures the script continuously listens for user input, processes it, and executes the corresponding commands until the user decides to quit.

6. **Logging the commands**: For future reference and auditing purposes, log every generated command:

   ```
   append_file("command-log.txt", "Request: " + request + "\
   nCommand: " + command + "\n\n")
   ```

   This code appends each user request and the corresponding generated command to a file named `command-log.txt`.

7. **Create the prompt file**: Enter the following text in a text file named `prompt4.txt`:

   ```
   Provide me with the Windows CLI command necessary to complete
   the following request:

   {INPUT}

   Assume I have all necessary apps, tools, and commands necessary
   to complete the request. Provide me with the command only and do
   not generate anything further. Do not provide any explanation.
   Provide the simplest form of the command possible unless I
   ask for special options, considerations, output, etc.. If the
   request does require a compound command, provide all necessary
   operators, options, pipes, etc.. as a single one-line command.
   Do not provide me more than one variation or more than one line.
   ```

Here's how the completed script should look:

```
import openai
from openai import OpenAI
import os
import subprocess

def open_file(filepath): #Open and read a file
    with open(filepath, 'r', encoding='UTF-8') as infile:
        return infile.read()

def save_file(filepath, content): #Create a new file or overwrite an
existing one.
    with open(filepath, 'w', encoding='UTF-8') as outfile:
        outfile.write(content)

def append_file(filepath, content): #Create a new file or append an
existing one.
    with open(filepath, 'a', encoding='UTF-8') as outfile:
        outfile.write(content)

#openai.api_key = os.getenv("OPENAI_API_KEY") #Use this if you prefer
to use the key in an environment variable.
```

```
openai.api_key = open_file('openai-key.txt') #Grabs your OpenAI key
from a file

def gpt_3(prompt): #Sets up and runs the request to the OpenAI API
    try:
        client = OpenAI()
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            prompt=prompt,
            temperature=0.1,
            max_tokens=600,
        )
        text = response['choices'].message.content.strip()
        return text
    except openai.error.APIError as e: #Returns and error and retries
if there is an issue communicating with the API
        print(f"\nError communicating with the API.")
        print(f"\nError: {e}") #More detailed error output
        print("\nRetrying...")
        return gpt_3(prompt)

while True: #Keeps the script running until we issue the "quit"
command at the request prompt
    request = input("\nEnter request: ")
    if not request:
        break
    if request == "quit":
        break
    prompt = open_file("prompt4.txt").replace('{INPUT}', request)
#Merges our request input with the pre-written prompt file
    command = gpt_3(prompt)
    process = subprocess.Popen(command, shell=True, stdout=subprocess.
PIPE, bufsize=1, universal_newlines=True) #Prepares the API response
to run in an OS as a command
    print("\n" + command + "\n")
    with process: #Runs the command in the OS and gives real-time
feedback
        for line in process.stdout:
            print(line, end='', flush=True)

    exit_code = process.wait()
    append_file("command-log.txt", "Request: " + request + "\nCommand:
" + command + "\n\n") #Write the request and GPT generated command to
a log
```

This script provides a fully operational GPT-powered, NLP-driven terminal interface, providing a powerful and user-friendly way to interact with your Linux system.

How it works…

At its core, this script bridges the gap between NLP and the Linux operating system. Let's break down the components to understand the intricacies of this integration:

1. **The OpenAI API connection**: The first major component is the connection to the OpenAI API,Remove this from the sentence. GPT-3.5 and GPT-4 models are autoregressive language model that use deep learning to produce human-like text. Their extensive training on diverse datasets means they can comprehend a wide range of prompts and generate accurate and coherent responses.

   When you make a query in natural language, such as `List all files in the current directory`, the script sends this query to the GPT-3 model. The model then processes it and responds with a corresponding Linux command – in this case, `ls`.

2. **Python integration with the OS**: Python's `subprocess` library is the linchpin that allows the script to execute commands on the operating system. This library provides an interface to spawn and interact with subprocesses, mimicking the command-line behavior within the script.

   The command returned by GPT-3 is executed using `subprocess.Popen()`. The advantage of using `Popen` over other methods is its flexibility. It spawns a new process, lets you interact with its input/output/error pipes, and obtains its return code.

3. **User interaction loop**: The script uses a `while` loop to keep the terminal running continuously, allowing the user to input multiple requests without having to restart the script. This emulates the behavior of a typical terminal where a user can run successive commands.

4. **Logging mechanism**: Maintaining a log of all executed commands is crucial for multiple reasons. For one, it aids in troubleshooting; if a command behaves unexpectedly, you can trace back to see what was executed. Furthermore, from a security perspective, having an audit trail of commands can be invaluable.

5. **Security measures**: Storing sensitive information such as API keys in plain text within scripts is a potential security risk. This script circumvents this by reading the API key from a separate file, ensuring that even if the script is shared or exposed, the API key remains protected. Always ensure that the file containing the API key has appropriate file permissions to limit unauthorized access.

6. **GPT-3 prompt design**: The design of the prompt is crucial. A well-crafted prompt will guide the model to provide more accurate results. In this script, a predefined prompt is merged with the user's input to generate a more comprehensive query for GPT-3. This ensures that the model has the right context to interpret the request and return an appropriate command.

In conclusion, this script embodies a seamless melding of advanced NLP capabilities with the power of the Linux operating system. By translating natural language into complex commands, it offers both beginners and experienced users an enhanced, intuitive, and efficient interface to interact with their systems.

There's more...

This script is just the tip of the iceberg when it comes to leveraging the power of NLP with operating systems. Here are some enhancements and extensions you can consider:

1. **Support for multiple operating systems**: Currently, the script is tailored for Linux commands, but with tweaks to the GPT-3 prompt, it can be adapted to work with Windows, macOS, or any other OS. By detecting the operating system type using Python (`os.name` or `platform.system()`), you can dynamically adjust the GPT-3 prompt to request OS-specific commands.
2. **Command validation**: Before executing any command, implement a security layer that validates the command against a list of safe commands. This can prevent potentially harmful commands from being executed inadvertently.
3. **Interactive command execution**: Some commands, especially installation or system configuration ones, might require user interaction (e.g., confirmation or choices). Enhancing the script to handle such interactive commands would be invaluable.
4. **Integration with other APIs**: Beyond just the OpenAI API, consider integrating other APIs to fetch real-time data. For instance, if you're working in cybersecurity, you could integrate threat intelligence feeds. This way, a query such as `Get me the latest threats related to ransomware` could fetch real-time data from these feeds.
5. **Enhanced logging**: The current logging mechanism records commands. Consider extending this to log the date, the time, and perhaps the user, especially if deployed on a multi-user system. This provides better auditing capabilities.
6. **User profiles**: If you're aiming to make this tool usable by multiple people, consider user profiles. Each profile could have its custom set of commands, permissions, and perhaps even its API keys.
7. **Command history**: Just like traditional terminals maintain a command history, having a mechanism where users can recall and rerun previous commands could enhance the user experience.

Remember, while the potential applications are vast, always be mindful of security. Ensure you're not exposing sensitive information, and always validate and sanitize inputs and outputs.

Hint

Using ChatGPT and the OpenAI API for red team support can be tricky due to OpenAI's evolving safety mechanisms. While we don't condone unethical or malicious behavior, sometimes it is necessary to mimic the tactics and actions of an attacker in order to truly assess our own vulnerabilities and associated level of risk. These methods should obviously be performed on authorized systems only. That said, when we can get OpenAI's models to support us in the efforts, it is of great value and can save a lot of time.

I have found that the OpenAI API tends to be more lenient than the ChatGPT web UI when it comes to red team adversarial tactics. It also helps in many cases to include in your prompts that you are authorized to perform the actions you are asking ChatGPT to assist with and tell ChatGPT that you are looking for examples (just as we have done in the previous recipe).

We will be providing even more strategies for getting the results you need from ChatGPT using red team methods later on in this book, as we cover more advanced techniques.

7

Threat Monitoring and Detection

In the dynamic and ever-evolving realm of cybersecurity, the role of timely detection, analysis, and response to threats is of paramount importance. Modern challenges require innovative solutions that leverage the power of technology, artificial intelligence, and human expertise. This chapter offers a deep dive into the world of proactive cybersecurity, exploring various methods and tools to stay ahead of potential threats.

At the forefront of our exploration is the concept of Threat Intelligence Analysis. As cyber threats continue to grow in complexity and volume, the need for effective and efficient threat intelligence becomes indispensable. This chapter introduces you to the potential of ChatGPT in analyzing raw threat data, extracting crucial indicators of compromise, and generating detailed narratives for each identified threat. While traditional platforms offer invaluable insights, the integration of ChatGPT presents a unique opportunity for swift initial analyses, providing immediate insights and augmenting the capabilities of existing systems.

Diving deeper, the chapter sheds light on the significance of Real-Time Log Analysis. With an ever-growing number of devices, applications, and systems generating logs, the ability to analyze this data in real-time becomes a critical asset. By utilizing the OpenAI API as an intelligent filter, we can highlight potential security incidents, offering invaluable context and enabling incident responders to act with precision and speed.

A specific focus is also given to the stealthy and persistent nature of **Advanced Persistent Threats** (**APTs**). These threats, often lurking in the shadows, pose significant challenges due to their evasive tactics. By leveraging ChatGPT's analytical prowess combined with native Windows utilities, this chapter offers a novel approach to detect such sophisticated threats, serving as a primer for those looking to integrate AI-driven insights into their threat hunting toolkit.

Recognizing the unique nature of each organization's cybersecurity landscape, the chapter delves into the art and science of Building Custom Threat Detection Rules. Generic rules often fail to capture the intricacies of specific threat landscapes, and this section serves as a guide to tailor-making rules that resonate with an organization's unique cybersecurity needs.

Lastly, the chapter navigates the waters of Network Traffic Analysis, emphasizing the importance of monitoring and analyzing network data. Through hands-on examples and scenarios, you'll learn to leverage the OpenAI API and Python's SCAPY library, offering a fresh perspective on how to detect anomalies and bolster network security.

In essence, this chapter stands as a testament to the fusion of traditional cybersecurity practices with modern AI-driven tools. Whether you're just starting your journey in cybersecurity or are a seasoned expert, this chapter promises a blend of theory, hands-on exercises, and insights that will enrich your cybersecurity toolkit.

In this chapter, we will cover the following recipes:

- Threat Intelligence Analysis
- Real-Time Log Analysis
- Detecting APTs using ChatGPT for Windows Systems
- Building Custom Threat Detection Rules
- Network Traffic Analysis and Anomaly Detection with PCAP Analyzer

Technical requirements

For this chapter, you will need a *web browser* and a stable *internet connection* to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account setup and have obtained your API key. If not, revisit *Chapter 1* for details. Basic familiarity with the Python programming language and working with the command line is necessary, as you'll be using **Python 3.x**, which needs to be installed on your system, for working with the OpenAI GPT API and creating Python scripts. A **code editor** will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter. Since we will be discussing APTs specifically for Windows systems, access to a Windows environment (preferably Windows Server) is essential.

Familiarity with the following subjects can be helpful:

- **Threat Intelligence Platforms**: Familiarity with common threat intelligence feeds and **indicators of compromise** (**IoCs**) would be advantageous.
- **Log Analysis Tools**: A tool or platform for real-time log analysis, such as ELK Stack (Elasticsearch, Logstash, Kibana) or Splunk.
- **Rule Creation**: Basic understanding of how threat detection rules are structured and the logic behind them. Familiarity with platforms like YARA can be beneficial.
- **Network Monitoring Tools**: Tools like Wireshark or Suricata for analyzing network traffic and detecting anomalies.

The code files for this chapter can be found here: `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

Threat Intelligence Analysis

In the dynamic field of cybersecurity, the importance of staying ahead of threats cannot be overstated. One of the pillars of this proactive approach is effective threat intelligence analysis. This recipe offers a hands-on guide on how to use ChatGPT for analyzing raw threat intelligence data. By the end of this exercise, you will have a working script capable of gathering unstructured threat intelligence data from a variety of sources, utilizing ChatGPT to identify and categorize potential threats, extracting indicators of compromise like IP addresses, URLs, and hashes, and finally, generating a contextual narrative for each identified threat. While ChatGPT isn't designed to replace specialized threat intelligence platforms, it can serve as an invaluable tool for quick initial analyses and insights.

This recipe aims to equip you with a set of skills crucial for any modern cybersecurity professional. You will learn how to set up your working environment for interacting with OpenAI's GPT models. You will also discover how to construct queries that prompt ChatGPT to sift through raw data to identify potential threats. Moreover, the recipe will teach you how to use ChatGPT to extract indicators of compromise from unstructured threat data. Lastly, you'll gain insights into understanding the context or narrative behind the threats you uncover, thereby enriching your threat analysis capabilities.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

1. `openai`: This library enables you to interact with the OpenAI API. Install it using the command `pip install openai`.
2. `Raw Threat Data`: Prepare a text file containing the raw threat intelligence data you wish to analyze. This can be gathered from various forums, security bulletins, or threat intelligence feeds.

By completing these steps, you'll be well-prepared to run the script and analyze raw threat intelligence data.

How to do it…

In this section, we'll walk through the steps to analyze raw threat intelligence data using ChatGPT. Since the primary focus of this recipe is to use ChatGPT prompts, the steps are geared towards querying the model effectively.

1. **Gather raw threat data**. Start by collecting unstructured threat intelligence data. This data can be sourced from various places, such as forums, blogs, and security bulletins/alerts. Store this data in a text file for easy access.

2. **Query ChatGPT for threat identification**. Open your favorite text editor or IDE and initiate a ChatGPT session. Enter the following prompt to identify potential threats in the raw data:

   ```
   Analyze the following threat data and identify potential
   threats: [Your Raw Threat Data Here]
   ```

   ChatGPT will analyze the data and provide a list of potential threats it has identified.

3. **Extract Indicators of Compromise (IoCs)**. Now, use the second prompt to have ChatGPT highlight specific indicators of compromise. Enter the following:

   ```
   Extract all indicators of compromise (IoCs) from the following
   threat data: [Your Raw Threat Data Here]
   ```

   ChatGPT will sift through the data and list out the IoCs such as IP addresses, URLs, and hashes.

4. **Begin contextual analysis**. To understand the context or narrative behind each identified threat, use the third prompt:

   ```
   Provide a detailed context or narrative behind the identified
   threats in this data: [Your Raw Threat Data Here]
   ```

   ChatGPT will provide you with a detailed analysis, explaining the origin, objectives, and potential impact of each threat.

5. **Store and Share**. Once you have all this information, store it in a centralized database and distribute the findings to relevant stakeholders for further action.

How it works…

In this recipe, we leveraged ChatGPT's natural language processing capabilities for threat intelligence analysis. Let's break down how each part works:

- **Gathering Raw Threat Data**. The first step involves collecting unstructured data from various sources. While ChatGPT isn't designed to scrape or gather data, you can manually compile this information from multiple sources into a text file. The objective is to get a comprehensive set of data that may contain hidden threats.
- **Querying ChatGPT for Threat Identification**. ChatGPT processes the raw data using natural language understanding to identify potential threats. Although not a replacement for specialized threat intelligence software, ChatGPT can give quick insights that are useful for initial assessments.
- **Extracting IoCs**. IoCs are elements in the data that signify malicious activity. These can range from IP addresses to file hashes. ChatGPT uses its text analysis capabilities to identify and list these IoCs, aiding in quicker decision-making for security professionals.
- **Contextual Analysis**. Understanding the context behind a threat is critical for assessing its severity and potential impact. ChatGPT provides a narrative or contextual analysis based on the data it has processed. This can give you valuable insights into the origin and objectives of the threat actors involved.

- **Storing and Sharing**. The final step involves storing the analyzed data and sharing it with relevant stakeholders. While ChatGPT doesn't handle database interactions or data distribution, its outputs can easily be integrated into existing workflows for these tasks.

By combining these steps, you harness the power of ChatGPT to add an extra layer of analysis to your threat intelligence efforts, all in a matter of minutes.

There's more...

While our primary focus has been on using ChatGPT through prompts, you can also automate this process by using the OpenAI API in Python. This way, you can integrate ChatGPT's analysis into your existing cybersecurity workflows. In this extended section, we'll guide you through the Python code to automate the ChatGPT threat analysis process.

1. **Import the OpenAI Library**. First, import the OpenAI library to interact with the OpenAI API.

```
import openai
from openai import OpenAI
```

2. **Initialize the OpenAI API Client**. Set your OpenAI API key to initialize the client. Use the environment variable method as demonstrated in previous recipes.

```
openai.api_key = os.getenv("OPENAI_API_KEY")
```

3. **Define the ChatGPT Query Function**. Create a function, `call_gpt`, to handle sending prompts to ChatGPT and receiving its responses.

```
def call_gpt(prompt):
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity SOC
       analyst with more than 25 years of experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]

    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7
    )
    return response.choices[0].message.content
```

4. **Create the Threat Analysis Function**. Now create a function, `analyze_threat_data`, which takes a file path as an argument and uses `call_gpt` to analyze the threat data.

```
def analyze_threat_data(file_path):
    # Read the raw threat data from the provided file
    with open(file_path, 'r') as file:
        raw_data = file.read()
```

5. **Complete the Threat Analysis Function**. Complete the `analyze_threat_data` function by adding the code to query ChatGPT for threat identification, IoC extraction, and contextual analysis.

```
    # Query ChatGPT to identify and categorize potential threats
    identified_threats = call_gpt(f"Analyze the
      following threat data and identify potential
        threats: {raw_data}")

    # Extract IoCs from the threat data
    extracted_iocs = call_gpt(f"Extract all indicators
      of compromise (IoCs) from the following threat
        data: {raw_data}")

    # Obtain a detailed context or narrative behind
      the identified threats
    threat_context = call_gpt(f"Provide a detailed
      context or narrative behind the identified
        threats in this data: {raw_data}")

    # Print the results
    print("Identified Threats:", identified_threats)
    print("\nExtracted IoCs:", extracted_iocs)
    print("\nThreat Context:", threat_context)
```

6. **Run the Script**. Finally, put it all together and run the main script.

```
if __name__ == "__main__":
    file_path = input("Enter the path to the raw
      threat data .txt file: ")
    analyze_threat_data(file_path)
```

Here is the correct script that should be pasted here:

```
import openai
from openai import OpenAI
import os

# Initialize the OpenAI API client
openai.api_key = os.getenv("OPENAI_API_KEY")

def call_gpt(prompt):
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity SOC analyst with more 
than 25 years of experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7
    )
    return response.choices[0].message.content

def analyze_threat_data(file_path):
    # Read the raw threat data from the provided file
    with open(file_path, 'r') as file:
        raw_data = file.read()

    # Query ChatGPT to identify and categorize potential threats
    identified_threats = call_gpt(f"Analyze the following threat data 
and identify potential threats: {raw_data}")

    # Extract IoCs from the threat data
    extracted_iocs = call_gpt(f"Extract all indicators of compromise
(IoCs) from the following threat data: {raw_data}")

    # Obtain a detailed context or narrative behind the identified 
threats
    threat_context = call_gpt(f"Provide a detailed context or 
narrative behind the identified threats in this data: {raw_data}")

    # Print the results
    print("Identified Threats:", identified_threats)
    print("\nExtracted IoCs:", extracted_iocs)
    print("\nThreat Context:", threat_context)

if __name__ == "__main__":
    file_path = input("Enter the path to the raw threat data .txt 
file: ")
    analyze_threat_data(file_path)
```

This recipe not only demonstrates the practical application of ChatGPT in enhancing threat intelligence analysis but also underscores the evolving role of AI in cybersecurity. By integrating ChatGPT into the process, we unlock a new dimension of efficiency and depth in analyzing threat data, making it an indispensable tool for cybersecurity professionals aiming to fortify their defenses in an ever-changing threat landscape.

How the Script Works

Let's look at the steps to understand how the script works:

1. **Import the OpenAI Library**. The `import openai` statement allows your script to use the OpenAI Python package, making all its classes and functions available. This is essential for making API calls to ChatGPT for threat analysis.
2. **Initialize the OpenAI API Client**. The `'openai.api_key = os.getenv("OPENAI_API_KEY")'` initializes the OpenAI API client by setting your personal API key. This API key authenticates your requests, allowing you to interact with the ChatGPT model. Make sure to set the `'YOUR_OPENAI_API_KEY'` environment variable with the actual API key you obtained from OpenAI.
3. + The function `call_gpt(prompt)` is a utility function designed to send your query to the ChatGPT model and retrieve the response. It uses a predefined system message to set the role of ChatGPT, ensuring the model's output aligns with the task at hand. The `openai.ChatCompletion.create()` function is where the API call happens, using parameters like *model*, *messages*, and `max_tokens` to customize the query.

4. **Create the Threat Analysis Function**. The function `analyze_threat_data(file_path)` serves as the core of the threat analysis process. It starts by reading raw threat data from a file specified by `file_path`. This raw data will be processed in the subsequent steps.
5. **Complete the Threat Analysis Function**. This part of the code fills out the `analyze_threat_data` function by employing the `call_gpt` utility function defined earlier. It sends three different queries to ChatGPT: one for identifying threats, another for extracting indicators of compromise, and a final one for contextual analysis. The results are then printed to the console for review.
6. **Run the Script**. The `if __name__ == "__main__":` block ensures that the script only runs when executed directly (not imported as a module). It asks the user to input the file path of the raw threat data and then calls the `analyze_threat_data` function to start the analysis.

Real-Time Log Analysis

In the complex and ever-changing world of cybersecurity, real-time threat monitoring and detection are paramount. This recipe introduces a cutting-edge approach using the OpenAI API to perform real-time log analysis and generate alerts for potential threats. By funneling data from diverse sources like firewalls, **Intrusion Detection Systems** (**IDS**), and various logs into a centralized monitoring platform, the OpenAI API serves as an intelligent filter. It analyzes the incoming data to highlight possible security incidents, providing invaluable context to each alert and thus enabling incident responders to prioritize more effectively. This recipe not only guides you through the process of setting up these alerting mechanisms but also shows you how to establish a feedback loop, allowing for continuous system improvement and adaptability to the evolving threat landscape.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

1. `openai`: This library enables you to interact with the OpenAI API. Install it using the command `pip install openai`.

In addition to the OpenAI package, you'll need the `asyncio` library for asynchronous programming and the `watchdog` library for monitoring file system events: `pip install asyncio watchdog`.

How to do it...

To implement real-time log analysis using the OpenAI API, follow these steps to set up your system for monitoring, threat detection, and alert generation. This approach will enable you to analyze and respond to potential security incidents as they occur.

1. **Import Required Libraries**. The first step is to import all the libraries that you'll be using in the script.

```
import asyncio
import openai
from openai import OpenAI
import os
import socket
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
```

2. **Initialize the OpenAI API Client**. Before you can start sending logs to be analyzed, initialize the OpenAI API client.

```
# Initialize the OpenAI API client
#openai.api_key = 'YOUR_OPENAI_API_KEY'  # Replace with your
actual API key if you choose not to use a system environment
variable
openai.api_key = os.getenv("OPENAI_API_KEY")
```

3. **Create Function to Call GPT**. Create a function that will interact with the GPT-3.5 Turbo model to analyze log entries.

```
def call_gpt(prompt):
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity SOC
              analyst with more than 25 years of
                experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7
    )
    return response.choices[0].message.content.strip()
```

4. **Setup Asynchronous Function for Syslog**. Set up an asynchronous function to handle incoming syslog messages. We're using the UDP protocol for this example.

```
async def handle_syslog():
    UDP_IP = "0.0.0.0"
    UDP_PORT = 514
    sock = socket.socket(socket.AF_INET,
      socket.SOCK_DGRAM)
    sock.bind((UDP_IP, UDP_PORT))
    while True:
        data, addr = sock.recvfrom(1024)
        log_entry = data.decode('utf-8')
        analysis_result = call_gpt(f"Analyze the following log
entry for potential threats: {log_entry} \n\nIf you believe
there may be suspicious activity, start your response with
'Suspicious Activity: ' and then your analysis. Provide nothing
else.")

        if "Suspicious Activity" in analysis_result:
            print(f"Alert: {analysis_result}")
        await asyncio.sleep(0.1)
```

5. **Setup File System Monitoring**. Utilize the `watchdog` library to monitor a specific directory for new log files.

```
class Watcher:
    DIRECTORY_TO_WATCH = "/path/to/log/directory"

    def __init__(self):
        self.observer = Observer()

    def run(self):
        event_handler = Handler()
        self.observer.schedule(event_handler,
          self.DIRECTORY_TO_WATCH, recursive=False)
        self.observer.start()
        try:
            while True:
                pass
        except:
            self.observer.stop()
            print("Observer stopped")
```

6. **Create Event Handler for File System Monitoring**. The `Handler` class will process the newly created files in the directory being watched.

```
class Handler(FileSystemEventHandler):
    def process(self, event):
        if event.is_directory:
            return
        elif event.event_type == 'created':
            print(f"Received file: {event.src_path}")
            with open(event.src_path, 'r') as file:
                for line in file:
                    analysis_result = call_gpt(f"Analyze the 
following log entry for potential threats: {line.strip()} \n\
nIf you believe there may be suspicious activity, start your 
response with 'Suspicious Activity: ' and then your analysis. 
Provide nothing else.")
        if "Suspicious Activity" in analysis_result:
            print(f"Alert: {analysis_result}")
    def on_created(self, event):
        self.process(event)
```

7. **Run the System**. Finally, put it all together and run your system.

```
if __name__ == "__main__":
    asyncio.run(handle_syslog())
    w = Watcher()
    w.run()
```

Here is how the completed script should look:

```
import asyncio
import openai
from openai import OpenAI
import os
import socket
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

# Initialize the OpenAI API client
#openai.api_key = 'YOUR_OPENAI_API_KEY'  # Replace with your actual 
API key if you choose not to use a system environment variable
```

```
openai.api_key = os.getenv("OPENAI_API_KEY")

# Function to interact with ChatGPT
def call_gpt(prompt):
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity SOC analyst
              with more than 25 years of experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7
    )
    return response.choices[0].message.content.strip()

# Asynchronous function to handle incoming syslog messages
async def handle_syslog():
    UDP_IP = "0.0.0.0"
    UDP_PORT = 514

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((UDP_IP, UDP_PORT))

    while True:
        data, addr = sock.recvfrom(1024)
        log_entry = data.decode('utf-8')
        analysis_result = call_gpt(f"Analyze the following log entry
for potential threats: {log_entry} \n\nIf you believe there may be
suspicious activity, start your response with 'Suspicious Activity: '
and then your analysis. Provide nothing else.")

        if "Suspicious Activity" in analysis_result:
            print(f"Alert: {analysis_result}")
        await asyncio.sleep(0.1)  # A small delay to allow
          other tasks to run

# Class to handle file system events
class Watcher:
    DIRECTORY_TO_WATCH = "/path/to/log/directory"

    def __init__(self):
        self.observer = Observer()

    def run(self):
        event_handler = Handler()
        self.observer.schedule(event_handler,
          self.DIRECTORY_TO_WATCH, recursive=False)
        self.observer.start()
        try:
            while True:
                pass
        except:
            self.observer.stop()
            print("Observer stopped")

class Handler(FileSystemEventHandler):
    def process(self, event):
        if event.is_directory:
            return
        elif event.event_type == 'created':
            print(f"Received file: {event.src_path}")
            with open(event.src_path, 'r') as file:
                for line in file:
                    analysis_result = call_gpt(f"Analyze the following
log entry for potential threats: {line.strip()} \n\nIf you believe
there may be suspicious activity, start your response with 'Suspicious
Activity: ' and then your analysis. Provide nothing else.")

        if "Suspicious Activity" in analysis_result:
            print(f"Alert: {analysis_result}")

    def on_created(self, event):
        self.process(event)

if __name__ == "__main__":
    # Start the syslog handler
    asyncio.run(handle_syslog())

    # Start the directory watcher
    w = Watcher()
    w.run()
```

By following this recipe, you've equipped your cybersecurity toolkit with an advanced real-time log analysis system, leveraging the OpenAI API for efficient threat detection and alerting. This setup not only enhances your monitoring capabilities but also ensures that your security posture is robust and responsive to the dynamic nature of cyber threats.

How it works…

Understanding how the code works is essential for tweaking it to fit your specific needs or for troubleshooting. Let's break down the key elements:

- **Importing Libraries**. The script starts by importing necessary Python libraries. This includes `asyncio` for asynchronous programming, `openai` for interacting with the OpenAI API, `os` for environment variables, and `socket` and `watchdog` for network and file system operations, respectively.
- **OpenAI API Initialization**. The `openai.api_key` is initialized using an environment variable. This key allows the script to interact with the GPT-3.5 Turbo model via the OpenAI API.
- **GPT-3.5 Turbo Function**. The `call_gpt()` function serves as a wrapper for the OpenAI API call. It takes a log entry as a prompt and returns an analysis. The function is configured to initiate a chat with the system role setting the context as a seasoned cybersecurity SOC analyst, which helps in generating more context-aware responses.
- **Asynchronous Syslog Handling**. The `handle_syslog()` function is asynchronous, allowing it to handle multiple incoming syslog messages without blocking. It calls the `call_gpt()` function with the log entry and checks for the keyword **Suspicious Activity** to generate alerts.
- **File System Monitoring**. The `Watcher` class uses the `watchdog` library to monitor a directory for new log files. It triggers the `Handler` class whenever a new file is created.
- **Event Handling**. The `Handler` class reads the new log files line by line and sends each line to the `call_gpt()` function for analysis. Similar to the syslog handling, it also checks for the keyword "Suspicious Activity" in the analysis result to generate alerts.
- **Alerting Mechanism**. Both the syslog handler and the file system event handler print an alert to the console if **Suspicious Activity** is found in the analysis. This can be easily extended to send alerts via email, Slack, or any other alerting mechanism.
- **Main Execution**. The script's main execution starts the asynchronous syslog handler and the file system watcher, making the system ready for real-time log analysis.

By structuring the code this way, you get a modular and easily extendable real-time log analysis system powered by the OpenAI API.

There's more…

The code presented in this recipe serves as a foundational layer for real-time log analysis using the OpenAI API. While it showcases the core functionalities, it's a basic implementation and should be extended to maximize its utility in a production environment. Here are some avenues for extension:

- **Scalability**. The current setup is basic and might not handle large-scale, high-throughput environments well. Consider using more advanced networking setups and distributed systems to scale the solution.
- **Alerting Mechanisms**. While the code prints alerts to the console, in a production scenario, you'd likely want to integrate with existing monitoring and alerting solutions like Prometheus, Grafana, or even a simple email alert system.
- **Data Enrichment**. The script currently sends raw log entries to the OpenAI API. Adding data enrichment steps to add context or correlate entries could improve the quality of the analysis.
- **Machine Learning Feedback Loop**. With more data and results, machine learning models could be trained to reduce false positives and improve accuracy over time.
- **User Interface**. An interactive dashboard could be developed to visualize the alerts and possibly control the behavior of the system in real-time.

Note of caution

It's crucial to note that sending actual sensitive data to the OpenAI API could expose it. While the OpenAI API is secure, it's not designed to handle sensitive or classified information. However, later in this book, we'll discuss methods to use local models to analyze sensitive logs, keeping your data local and private

Detecting APTs using ChatGPT for Windows Systems

APTs are a class of cyber-attacks where the intruder gains unauthorized access to a system and remains undetected for an extended period. These attacks often target organizations with high-value information, including financial data, intellectual property, or national security details. APTs are particularly challenging to detect due to their low-and-slow operational tactics and their use of sophisticated techniques to evade traditional security measures. This recipe aims to leverage the analytical capabilities of ChatGPT to assist in the active monitoring and detection of such threats on Windows systems. By combining native Windows utilities with ChatGPT's natural language processing prowess, you can create a rudimentary, yet insightful, threat hunting tool. While this approach is not a replacement for specialized threat hunting software or experts, it serves as an educational or proof-of-concept method for understanding how AI can contribute to cybersecurity.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

1. `Openai`:This library enables you to interact with the OpenAI API. Install it using the command `pip install openai`.

Finally, the script uses native Windows command-line utilities like `reg query`, `tasklist`, `netstat`, `schtasks`, and `wevtutil`. These commands come pre-installed on most Windows systems, so no additional installation is needed for them.

Important note

This script must be executed with administrative privileges to access specific system information on a Windows machine. Ensure that you have administrative access or consult your system administrator if you're in an organization.

How to do it…

To detect **Advanced Persistent Threats** (**APTs**) on Windows systems, follow these steps to gather system data and analyze it with ChatGPT for potential security threats.

1. **Importing Required Modules**. First, import the required Python modules. You'll need the **subprocess** module to run Windows commands, **os** to fetch environment variables, and **openai** to interact with ChatGPT.

```
import subprocess
import os
import openai
from openai import OpenAI
```

2. **Initialize the OpenAI API client**. Next, initialize the OpenAI API client with your API key. You can either hardcode the API key or retrieve it from an environment variable.

```
# Initialize the OpenAI API client
#openai.api_key = 'YOUR_OPENAI_API_KEY'
openai.api_key = os.getenv("OPENAI_API_KEY")
```

3. **Define the ChatGPT Interaction Function**. Create a function that will interact with ChatGPT using a given prompt. This function takes care of sending the prompt and messages to ChatGPT and returns its response.

```
def call_gpt(prompt):
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity SOC
              analyst with more than 25 years of
                experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]
    client = OpenAI()
    response = client.chat.completions.creat(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7
    )
    response.choices[0].message.content.strip()
```

Important note

It is possible that you may need to use the model `gpt-4-turbo-preview` if the data gathering produces an error indicating that the amount of tokens exceed the model's limit.

4. **Define the Command Execution Function**. This function will run a given Windows command and return its output.

```
# Function to run a command and return its output
def run_command(command):
    result = subprocess.run(command, stdout=
      subprocess.PIPE, stderr=subprocess.PIPE,
        text=True, shell=True)
    return result.stdout
```

5. **Gather and Analyze Data**. Now that the functions are set, the next step is to gather data from the Windows system and analyze it with ChatGPT. The data gathering uses native Windows commands.

```
# Gather data from key locations
# registry_data = run_command('reg query HKLM /s')  # This
produces MASSIVE data. Replace with specific registry keys if
needed
# print(registry_data)
process_data = run_command('tasklist /v')
print(process_data)
network_data = run_command('netstat -an')
print(network_data)
scheduled_tasks = run_command('schtasks /query /fo LIST')
print(scheduled_tasks)
security_logs = run_command('wevtutil qe Security /c:10 /rd:true
/f:text')  # Last 10 security events. Adjust as needed
print(security_logs)

# Analyze the gathered data using ChatGPT
analysis_result = call_gpt(f"Analyze the following Windows
system data for signs of APTs:\nProcess Data:\n{process_data}\n\
nNetwork Data:\n{network_data}\n\nScheduled Tasks:\n{scheduled_
tasks}\n\nSecurity Logs:\n{security_logs}") # Add Registry
Data:\n{#registry_data}\n\n if used

# Display the analysis result
print(f"Analysis Result:\n{analysis_result}")
```

Here is how the completed script should look:

```
import subprocess
import os
import openai
from openai import OpenAI

# Initialize the OpenAI API client
#openai.api_key = 'YOUR_OPENAI_API_KEY'  # Replace with your actual
API key or use a system environment variable as shown below
openai.api_key = os.getenv("OPENAI_API_KEY")

# Function to interact with ChatGPT
def call_gpt(prompt):
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity SOC analyst
              with more than 25 years of experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7
    )
    return response.choices[0].message.content.strip()

# Function to run a command and return its output
def run_command(command):
    result = subprocess.run(command,
    stdout=subprocess.PIPE, stderr=subprocess.PIPE,
      text=True, shell=True)
    return result.stdout

# Gather data from key locations
# registry_data = run_command('reg query HKLM /s')  # This produces
MASSIVE data. Replace with specific registry keys if needed
# print(registry_data)
process_data = run_command('tasklist /v')
print(process_data)
network_data = run_command('netstat -an')
print(network_data)
scheduled_tasks = run_command('schtasks /query /fo LIST')
print(scheduled_tasks)
security_logs = run_command('wevtutil qe Security /c:10 /rd:true
/f:text')  # Last 10 security events. Adjust as needed
print(security_logs)

# Analyze the gathered data using ChatGPT
analysis_result = call_gpt(f"Analyze the following Windows system data
for signs of APTs:\nProcess Data:\n{process_data}\n\nNetwork Data:\
n{network_data}\n\nScheduled Tasks:\n{scheduled_tasks}\n\nSecurity
Logs:\n{security_logs}") # Add Registry Data:\n{#registry_data}\n\n if
```

```
used

# Display the analysis result
print(f"Analysis Result:\n{analysis_result}")
```

In this recipe, we've explored a novel approach to APT detection by leveraging ChatGPT's analytical capabilities. Utilizing native Windows command-line utilities for data collection and feeding this information into ChatGPT, we've created a rudimentary, yet insightful, threat hunting tool. This method offers a unique way to identify and understand APTs in real-time, aiding in the timely planning of response strategies.

How it works...

This recipe takes a unique approach by combining Python scripting with ChatGPT's natural language processing abilities to create a basic APT detection tool for Windows systems. Let's dissect each part to understand its intricacies.

- **Data Collection with Native Windows Commands**. The Python script uses a series of native Windows command-line utilities to gather relevant system data. Commands like **reg query** fetch registry entries, which could contain configurations set by an APT. Similarly, **tasklist** enumerates running processes, and **netstat -an** gives a snapshot of current network connections, among others.

  These commands are part of the Windows operating system and are executed using Python's **subprocess** module, which allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

- **Interacting with ChatGPT through OpenAI API**. The `call_gpt` function serves as the bridge between the Python script and ChatGPT. It utilizes the OpenAI API to send a prompt along with the collected system data to ChatGPT.

  The OpenAI API requires an API key for authentication, which can be obtained from OpenAI's official website. This API key is used to initialize the OpenAI API client in the script.

- **Analysis and Context by ChatGPT**. ChatGPT receives the system data along with a prompt that guides it to look for anomalies or indicators of APT activities. The prompt is crafted to be specific to the task, leveraging ChatGPT's ability to understand and analyze text.

  ChatGPT's analysis aims to find irregularities or anomalies in the data. It tries to identify unusual registry entries, suspicious running processes, or odd network connections that could indicate an APT.

- **Output and Result Interpretation**. Once the analysis is complete, ChatGPT's findings are returned as a text output. This output is then printed to the console by the Python script.

  The output should be considered a starting point for further investigation. It provides clues and potential indicators that can guide your response strategy.

- **Administrative Privileges Requirement**. It's important to note that the script must be run with administrative privileges to access certain protected system information. This ensures that the script can probe into areas of the system that are usually restricted, offering a more comprehensive data set for analysis.

By carefully combining Python's capability to interact with system-level details and ChatGPT's prowess in natural language understanding, this recipe provides a rudimentary but insightful tool for real-time threat detection and analysis.

There's more…

The recipe we've just walked through offers a basic yet effective approach to identifying potential APT activities on a Windows system. However, it's worth noting that this is just the tip of the iceberg, and there are several ways to extend this functionality for more comprehensive threat hunting and monitoring:

- **Machine Learning Integration**. While ChatGPT provides a good starting point for anomaly detection, integrating machine learning algorithms for pattern recognition could make the system even more robust.
- **Automated Response**. Currently, the script provides an analysis that can be used for manual response planning. You could extend this by automating certain responses, such as isolating a network segment or disabling a user account based on the severity of the threat.
- **Longitudinal Analysis**. The script performs a point-in-time analysis. However, APTs often reveal themselves through behaviors that change over time. Storing data over extended periods and running trend analyses could provide more accurate detection.
- **Integrate with Security Information and Event Management (SIEM) Solutions**. SIEM solutions can provide a more comprehensive view of an organization's security posture. Integrating the script's output into a SIEM could allow for correlation with other security events, enhancing the overall detection capability.
- **Multi-System Analysis**. The current script focuses on a single Windows system. Extending it to collect data from multiple systems in a network can provide a more holistic view of potential threats.
- **User Behavior Analytics** *(***UBA***)*. Incorporating UBA can add another layer of sophistication. By understanding normal user behaviors, the system can more accurately identify anomalous activities that may indicate a threat.
- **Scheduled Runs**. Instead of running the script manually, you could schedule it to run at regular intervals, providing a more continuous monitoring solution.
- **Alerting Mechanism**. Implementing an alerting mechanism that notifies system administrators or security teams in real-time can expedite the response process.

- **Customizable Threat Indicators**. Allow for customization in the script where operators can define their threat indicators based on the evolving threat landscape.
- **Documentation and Reporting**. Enhancing the script to generate detailed reports can aid in post-incident analysis and compliance reporting.

By considering these extensions, you can transform this rudimentary tool into a more comprehensive, dynamic, and responsive threat monitoring system.

Building Custom Threat Detection Rules

In the evolving landscape of cybersecurity, generic threat detection rules often fall short. The nuances of each organization's network and systems necessitate custom rules tailored for specific threat landscapes. This recipe aims to equip you with the skills to identify unique threats and draft custom detection rules, specifically YARA rules, using ChatGPT. By walking you through the process—from threat identification to rule deployment—with hands-on sample scenarios, this recipe serves as a comprehensive guide for enhancing your organization's threat monitoring and detection capabilities.

Getting ready

The prerequisites for this recipe are simple. You just need a web browser and an OpenAI account. If you haven't already created an account or need a refresher on how to use the ChatGPT interface, refer back to *Chapter 1* for a comprehensive guide.

You should also have a clear understanding of your organizational environment. This includes an inventory of the types of systems deployed, the software in use, and the most critical assets requiring protection.

Ensure you have:

1. A test environment where you can safely deploy and test the rules. This could be a virtualized network or an isolated lab setup.
2. An existing threat detection system capable of using YARA rules or similar for testing purposes.

For those who are not familiar with YARA rules, you may want to brush up on the basics as this recipe will require some understanding of how they work in a threat detection context.

How to do it…

Important note

Two sample threat scenarios can be found in the official GitHub repository for this book. These scenarios can be used to test the prompts in this recipe and also provide guidance on creating your own practice scenarios.

The process of building custom threat detection rules with ChatGPT involves a series of steps. These steps will take you from identifying unique threats to deploying effective rules.

1. **Identify Unique Threats**.

   - *Sub-step 1*: Conduct an internal assessment or consult your cybersecurity team to identify the specific threats most relevant to your environment.
   - *Sub-step 2*: Review any recent incidents, logs, or threat intelligence reports for patterns or indicators.

Important note

The objective here is to find something specific—a unique file, an unusual system behavior, or a particular network pattern—that isn't already covered by generic detection rules.

2. **Draft Rules with ChatGPT**.

   - *Sub-step 1*: Open your web browser and navigate to the ChatGPT web UI.
   - *Sub-step 2*: Initiate a conversation with ChatGPT. Be as specific as possible about the threat characteristics. For example, if you're dealing with a malware that leaves a unique file, say so.

   Sample Prompt:

   ```
   I've noticed suspicious network activity where an unknown
   external IP is making multiple failed SSH login attempts on
   one of our critical servers. The IP is 192.168.1.101 and it's
   targeting Server-XYZ on SSH port 22. Can you help me draft a
   YARA rule to detect this specific activity?
   ```

   - *Sub-step 3*: Review the YARA rule that ChatGPT drafts for you. Make sure it includes the characteristics that are specific to the threat you've identified.

3. **Test Rules**.

   - *Sub-step 1*: Access your test environment, which should be isolated from your production network.
   - *Sub-step 2*: Deploy the YARA rule by adding it to your threat detection system. If you're new to this, most systems have an **Import** or **Upload** feature for new rules.
   - *Sub-step 3*: Run initial scans to check for false positives and the rule's overall effectiveness.

Important note

Be prepared to roll back changes or disable the rule if it causes disruptions.

4. **Refinement**.

   - *Sub-step 1*: Assess the test results. Note down any false positives or misses.
   - *Sub-step 2*: Return to ChatGPT with this data for refinement.

   Sample Prompt for Refinement:

   ```
   The YARA rule for detecting the suspicious SSH activity is
   generating some false positives. It's alerting on failed SSH
   attempts that are part of routine network scans. Can you help me
   refine it to focus only on the pattern described in the initial
   scenario?
   ```

5. **Deployment**.

   - *Sub-step 1:* Once you're confident with the rule's performance, prepare for deployment.
   - *Sub-step 2*: Integrate the refined rule into your production threat detection systems using the system's rule management interface.

How it works…

Understanding the mechanics behind each step will provide you with the insights needed to adapt this recipe for other threat scenarios. Let's break down what's happening:

- **Identify Unique Threats**. At this stage, you're essentially conducting threat hunting. You're going beyond the alerts and logs to find patterns or behaviors that are unusual and specific to your environment.
- **Draft Rules with ChatGPT**. ChatGPT uses its trained model to understand the threat characteristics you provide. Based on that understanding, it drafts a YARA rule aimed at detecting the described threat. It's a form of automated rule generation, saving you the time and effort needed to write rules manually.
- **Test Rules**. Testing is crucial in any cybersecurity task. Here, you're not just checking if the rule works, but also if it works without causing disruptions or false positives. A poorly designed rule can be as problematic as having no rule at all.
- **Refinement**. This step is about iteration. Cyber threats are not static; they evolve. The rules you create will likely need to be adjusted over time, either because the threat has changed or because the initial rule wasn't perfect.
- **Deployment**. Once a rule is tested and refined, it's ready to be deployed into production. This is the final validation of your efforts. However, continuous monitoring is essential to ensure that the rule remains effective against the threat it was designed to detect.

By understanding how each step works, you can adapt this method to various threat types and scenarios, making your threat detection system more robust and responsive.

There's more...

Now that you've learned how to create custom threat detection rules with ChatGPT, you might be interested in diving deeper into related topics and advanced functionalities. Here are some areas worth exploring:

- **Advanced YARA Features**. Once you're comfortable with basic YARA rule creation, consider delving into its advanced features. YARA offers functionalities like condition statements and external variables that can make your custom rules even more effective.
- **Continuous Monitoring and Tuning**. Cyber threats are ever-evolving, and so should your detection rules. Regularly review and update your custom rules to adapt to new threat landscapes and to fine-tune their performance.
- **Integration with SIEM Solutions**. Custom YARA rules can be integrated into existing SIEM solutions. This integration allows for a more comprehensive monitoring approach, correlating rule alerts with other security events.
- **Community Resources**. For further exploration and support, check out online forums, blogs, or GitHub repositories dedicated to YARA and threat detection. These platforms can be excellent resources for learning and troubleshooting.
- **Future of AI in Threat Detection**. The landscape of threat detection is continuously changing, with machine learning and AI playing an increasingly crucial role. Tools like ChatGPT can significantly streamline the rule-creation process, acting as a valuable asset in modern cybersecurity efforts.

Network Traffic Analysis and Anomaly Detection with PCAP Analyzer

In the constantly evolving landscape of cybersecurity, keeping tabs on network traffic is crucial. Traditional methods often involve using specialized network monitoring tools and considerable manual effort. This recipe takes a different approach by leveraging the OpenAI API in conjunction with Python's SCAPY library. By the end of this recipe, you'll learn how to analyze a PCAP file containing captured network traffic and identify potential anomalies or threats, all without the need for real-time API calls. This makes the analysis not only insightful but also cost-effective. Whether you're a cybersecurity newbie or a seasoned professional, this recipe offers a novel way to bolster your network security measures.

Getting ready

Before diving into this recipe, ensure you have your OpenAI account set up and your API key on hand. If not, you should refer back to *Chapter 1* for the necessary setup details. You will also need **Python version 3.10.x or later**.

Additionally, confirm you have the following Python libraries installed:

1. `openai`:This library enables you to interact with the OpenAI API. Install it using the command `pip install openai`.
2. `SCAPY Library`: Install the SCAPY Python library, which will be used to read and analyze PCAP files. You can install it using pip: `pip install scapy`
3. `PCAP File`: Have a PCAP file ready for analysis. You can either capture network traffic using tools like Wireshark or Tcpdump, or use sample files available at: `https://wiki.wireshark.org/SampleCaptures`. A sample `example.pcap` file has also been provided in the GitHub repository for this recipe.
4. `libpcap` (Linux and MacOS) or `Ncap` (Windows): You will need to install the appropriate library to enable SCAPY to read the PCAP files. `libpcap` can be found at `https://www.tcpdump.org/` and `Ncap` can be found at `https://npcap.com/`.

How to do it…

This recipe will guide you through a step-by-step process to analyze network traffic and detect anomalies using ChatGPT and Python's SCAPY library.

1. **Initialize OpenAI API Client**. Before you can interact with the OpenAI API, you need to initialize the OpenAI API client. Replace `YOUR_OPENAI_API_KEY` with your actual API key.

```
import openai
from openai import OpenAI
import os
#openai.api_key = 'YOUR_OPENAI_API_KEY'  # Replace with your
actual API key or set the OPENAI_API_KEY environment variable
openai.api_key = os.getenv("OPENAI_API_KEY")
```

2. **Create Function to Interact with the OpenAI API**. Define a function named `chat_with_gpt` that takes a prompt and sends it to the API for analysis.

```
# Function to interact with the OpenAI API
def chat_with_gpt(prompt):
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity SOC
       analyst with more than 25 years of experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7
    )
    return response.choices[0].message.content.strip()
```

3. **Read and Pre-process PCAP File**. Utilize the SCAPY library to read a captured PCAP file and summarize the network traffic.

```
from scapy.all import rdpcap, IP, TCP
# Read PCAP file
packets = rdpcap('example.pcap')
```

4. **Summarize Traffic**. Process the PCAP file to summarize key traffic aspects like unique IP addresses, ports, and protocols used.

```
# Continue from previous code snippet
ip_summary = {}
port_summary = {}
protocol_summary = {}

for packet in packets:
    if packet.haslayer(IP):
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        ip_summary[f"{ip_src} to {ip_dst}"] =
        ip_summary.get(f"{ip_src} to {ip_dst}", 0) + 1
    if packet.haslayer(TCP):
        port_summary[packet[TCP].sport] =
          port_summary.get(packet[TCP].sport, 0) + 1

    if packet.haslayer(IP):
        protocol_summary[packet[IP].proto] =
        protocol_summary.get(packet[IP].proto, 0) + 1
```

5. **Feed Summarized Data to ChatGPT**. Send the summarized data to the OpenAI API for analysis. Use OpenAI's API to look for anomalies or suspicious patterns.

```
# Continue from previous code snippet
analysis_result = chat_with_gpt(f"Analyze the following
summarized network traffic for anomalies or potential threats:\
n{total_summary}")
```

6. **Review Analysis and Alert**. Check the analysis provided by the LLM. If any anomalies or potential threats are detected, alert the security team for further investigation.

```
# Continue from previous code snippet
print(f"Analysis Result:\n{analysis_result}")
```

Here's how the completed script should look:

```
from scapy.all import rdpcap, IP, TCP
import os
import openai
from openai import OpenAI

# Initialize the OpenAI API client
#openai.api_key = 'YOUR_OPENAI_API_KEY'  # Replace with your actual
API key or set the OPENAI_API_KEY environment variable
openai.api_key = os.getenv("OPENAI_API_KEY")

# Function to interact with ChatGPT
def chat_with_gpt(prompt):
    messages = [
        {
            "role": "system",
            "content": "You are a cybersecurity SOC analyst
              with more than 25 years of experience."
        },
        {
            "role": "user",
            "content": prompt
        }
    ]
    client = OpenAI()
    response = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=messages,
        max_tokens=2048,
        n=1,
        stop=None,
        temperature=0.7
    )
    return response.choices[0].message.content.strip()

# Read PCAP file
packets = rdpcap('example.pcap')

# Summarize the traffic (simplified example)
ip_summary = {}
port_summary = {}
protocol_summary = {}

for packet in packets:
    if packet.haslayer(IP):
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        ip_summary[f"{ip_src} to {ip_dst}"] =
          ip_summary.get(f"{ip_src} to {ip_dst}", 0) + 1

    if packet.haslayer(TCP):
        port_summary[packet[TCP].sport] =
          port_summary.get(packet[TCP].sport, 0) + 1

    if packet.haslayer(IP):
        protocol_summary[packet[IP].proto] =
          protocol_summary.get(packet[IP].proto, 0) + 1

# Create summary strings
ip_summary_str = "\n".join(f"{k}: {v} packets" for k,
  v in ip_summary.items())
port_summary_str = "\n".join(f"Port {k}: {v} packets"
  for k, v in port_summary.items())
protocol_summary_str = "\n".join(f"Protocol {k}:
  {v} packets" for k, v in protocol_summary.items())

# Combine summaries
total_summary = f"IP Summary:\n{ip_summary_str}\n\nPort Summary:\
n{port_summary_str}\n\nProtocol Summary:\n{protocol_summary_str}"

# Analyze using ChatGPT
```

```
analysis_result = chat_with_gpt(f"Analyze the following summarized
network traffic for anomalies or potential threats:\n{total_summary}")

# Print the analysis result
print(f"Analysis Result:\n{analysis_result}")
```

With the completion of this recipe, you've taken a significant step forward in utilizing AI for network traffic analysis and anomaly detection. By integrating Python's SCAPY library with ChatGPT's analytical capabilities, you've crafted a tool that not only simplifies the identification of potential network threats but also enriches your cybersecurity arsenal, making your network monitoring efforts both efficient and insightful.

How it works…

This recipe is designed to break down the complexity of network traffic analysis into a set of manageable tasks that utilize Python programming and the OpenAI API. Let's delve into each aspect to understand it better:

- **SCAPY for Traffic Summarization**. SCAPY is a Python library for networking that allows you to handle, manipulate, and analyze network packets. In our case, we use SCAPY's `rdpcap` function to read the PCAP file, which is essentially a capture of network packets saved to a file. After reading this file, we loop through each packet to collect data on IP addresses, ports, and protocols, summarizing these into dictionaries.
- **Initializing OpenAI API Client**. The OpenAI API provides programmatic access to powerful machine learning models like GPT-3. To start using the API, you need to initialize it with an API key, which you can obtain from OpenAI's website. This key is used to authenticate your requests.
- **Interaction with OpenAI API**. We define a function, `interact_with_openai_api`, which takes a text prompt as an argument and sends it to the OpenAI API. The function constructs a message structure that includes a system role, defining the context for the AI (in our case, a cybersecurity SOC analyst), and a user role, which provides the actual query or prompt. It then calls OpenAI's `ChatCompletion.create` method to get the analysis.
- **OpenAI API for Anomaly Detection**. Once the summarized data is ready, it is sent as a prompt to the OpenAI API for analysis. The API's model scans this summary and outputs its analysis, which could include detection of anomalies or suspicious activities based on the data it received.
- **Result Interpretation**. Finally, the output from the OpenAI API is printed to the console using Python's `print` function. This output can include potential anomalies and could serve as a trigger for further investigations or alerts within your cybersecurity framework.

By understanding each of these components, you'll gain the ability to adapt this recipe to specific cybersecurity tasks, even if you're relatively new to Python or OpenAI's offerings.

There's more…

While the steps outlined in this recipe provide a solid foundation for network traffic analysis and anomaly detection, there are various ways to build upon and extend this knowledge.

- **Extend the Code for Advanced Analysis**. The Python script in this recipe provides a basic overview of the network traffic and potential anomalies. You could extend this code to perform more detailed analyses, such as flagging specific types of network behavior or integrating machine learning algorithms for anomaly detection.
- **Integrate with Monitoring Tools**. While this recipe focuses on a standalone Python script, the logic could easily be integrated into existing network monitoring tools or SIEM systems to provide real-time analysis and alerting capabilities.

8
Incident Response

Incident response is a critical component of any cybersecurity strategy, involving the identification, analysis, and mitigation of security breaches or attacks. A timely and effective response to incidents is essential to minimize damage and prevent future attacks. In this chapter, we will delve into leveraging ChatGPT and the OpenAI API to enhance various aspects of the incident response process.

We will begin by exploring how ChatGPT can assist in incident analysis and triage, providing quick insights and prioritizing events based on their severity. Next, we will see how to generate comprehensive incident response playbooks tailored to specific scenarios, streamlining the response process.

Furthermore, we will utilize ChatGPT for root cause analysis, helping to identify the origins and methods of an attack. This can significantly speed up the recovery process and fortify defenses against similar threats in the future.

Lastly, we will automate the creation of briefing reports and incident timelines, ensuring that stakeholders are well-informed and that a detailed record of the incident is maintained for future reference.

By the end of this chapter, you will be equipped with a suite of AI-powered tools and techniques that can significantly enhance their incident response capabilities, making them faster, more efficient, and more effective.

In this chapter, we will cover the following recipes:

- ChatGPT-assisted incident analysis and triage
- Generating incident response playbooks
- ChatGPT-assisted root cause analysis
- Automated briefing reports and incident timeline reconstruction

Technical requirements

For this chapter, you will need a web browser and a stable internet connection to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account set up obtain your API key. If not, revisit *Chapter 1* for details. Basic familiarity with the Python programming language and working with its command line is necessary, as you'll be using Python 3.x, which needs to be installed on your system to work with the OpenAI GPT API and create Python scripts. A code editor will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter. Finally, since many penetration testing use cases rely heavily on the Linux operating system, access to and familiarity with a Linux distribution (preferably Kali Linux) is recommended:

- Incident data and logs: Access to incident logs or simulated data is important for practical exercises. This will help in understanding how ChatGPT can assist in analyzing incidents and generating reports.
- The code files for this chapter can be found here: `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

ChatGPT-assisted incident analysis and triage

In the dynamic realm of cybersecurity, incidents are inevitable. The key to mitigating the impact lies in how effectively and swiftly an organization can respond. This recipe introduces an innovative approach to incident analysis and triage, leveraging the conversational capabilities of ChatGPT. By simulating the role of an Incident Commander, ChatGPT guides users through the initial critical steps of triaging a cybersecurity event.

Through an engaging question-and-answer format, ChatGPT assists in identifying the nature of suspicious activities, the systems or data affected, triggered alerts, and the extent of the impact on business operations. This interactive method not only aids in immediate decision-making, such as isolating affected systems or escalating issues but also serves as a valuable training tool for cybersecurity professionals. Embracing this AI-driven strategy elevates an organization's incident response readiness to a new pinnacle.

Before proceeding further, it's crucial to note the sensitivity of information shared during such interactions. The upcoming chapter on private local **large language models** (**LLMs**) addresses this concern, guiding users on how to maintain confidentiality while benefiting from AI assistance in incident response.

Getting ready

Before diving into the interactive session with ChatGPT for incident triage, it's imperative to establish a foundational understanding of the incident response process and familiarize oneself with the conversational interface of ChatGPT. No specific technical prerequisites are required for this recipe, making it accessible to professionals across various levels of technical expertise. However, a basic grasp of common cybersecurity terminologies and incident response protocols will enhance the efficacy of the interaction.

Ensure that you have access to the ChatGPT interface, either through the OpenAI website or an integrated platform. Familiarize yourself with initiating conversations and providing clear, concise inputs to maximize the utility of ChatGPT's responses.

With the preparatory steps addressed, you're set to embark on the journey of AI-assisted incident triage.

How to do it…

Engaging with ChatGPT for incident triage is a collaborative effort. It's essential to guide the AI step by step, providing detailed information and context to each query. This ensures that the AI's guidance is as relevant and actionable as possible. Here's how you can proceed:

1. **Initiate the incident triage dialogue**: Begin by introducing the situation to ChatGPT using the following prompt:

   ```
   You are the Incident Commander for an unfolding cybersecurity
   event we are currently experiencing. Guide me step by step,
   one step at a time, through the initial steps of triaging this
   incident. Ask me the pertinent questions you need answers for
   each step as we go. Do not move on to the next step until we are
   satisfied that the step we are working on has been completed.
   ```

2. **Provide incident details and respond to queries**: As ChatGPT asks questions, offer specific and detailed responses. Information about the nature of suspicious activities, any affected systems or data, any triggered alerts, and any business operations impacted will be crucial. The granularity of your details will significantly influence the accuracy and relevance of ChatGPT's guidance.
3. **Follow ChatGPT's step-by-step guidance**: ChatGPT will provide instructions and recommendations one step at a time based on your responses. It's vital to follow these steps meticulously and not proceed to the next step until you've adequately addressed the current one.
4. **Iterate and update information**: Incident response is an evolving scenario where new details can come to light at any moment. Keep ChatGPT updated with the latest developments and iterate through the steps as necessary, ensuring that the AI's guidance adapts to the changing situation.
5. **Document the interaction**: Maintain a record of the dialogue for future reference. This can be a valuable resource for post-incident reviews, refining response strategies, and training team members.

How it works…

The effectiveness of this recipe hinges on the carefully crafted prompt that instructs ChatGPT to act as an Incident Commander, guiding the user through the incident triage process. The prompt is designed to elicit a structured, interactive dialogue, mirroring the step-by-step decision-making typical of real-world incident response.

The specificity of the prompt, emphasizing the step-by-step and one step at a time process, is crucial. It instructs ChatGPT to avoid overwhelming the user with information and, instead, provides guidance in manageable, sequential steps. This approach allows for a more focused response from ChatGPT, closely aligning with how an Incident Commander would progressively assess and address an unfolding situation.

By requesting ChatGPT to ask pertinent questions before proceeding to the next step, the prompt ensures that each phase of the triage is thoroughly addressed. This mimics the iterative nature of incident response, where each action is based on the most current and relevant information.

ChatGPT's programming and training on a diverse range of texts allows it to understand the context provided by the user and the intent behind the prompt. As a result, it responds by simulating the role of an Incident Commander, drawing from best practices and protocols in cybersecurity incident response. The AI's responses are generated based on patterns it has learned during its training, enabling it to provide relevant questions and actionable recommendations.

Furthermore, this design of this prompt encourages users to engage deeply with the AI, fostering a collaborative problem-solving environment. This not only aids in the immediate triage process but also helps users develop a more nuanced understanding of incident response dynamics.

In summary, the prompt's structure and specificity play a pivotal role in guiding ChatGPT's responses, ensuring that the AI delivers targeted, step-by-step guidance that closely resembles the thought processes and actions of an experienced Incident Commander.

There's more...

While the recipe provides a structured approach to using ChatGPT for incident triage, there are additional considerations and extensions that can enhance its utility:

- **Simulated training scenarios**: Use this recipe as a training exercise for cybersecurity teams. Simulating different types of incidents can prepare teams for a variety of real-world scenarios, improving their readiness and response capabilities.
- **Integration with incident response tools**: Consider integrating ChatGPT's guidance with your existing incident response tools and platforms. This can streamline the process, allowing for quicker implementation of the AI's recommendations.
- **Customization for organization-specific protocols**: Tailor the interaction with ChatGPT to reflect your organization's specific incident response protocols. This ensures that the guidance provided is aligned with your internal policies and procedures.
- **Confidentiality and privacy**: Be mindful of the sensitivity of information shared during the interaction. Utilize private instances of LLMs or anonymize data to ensure confidentiality. The upcoming chapter on private local LLMs provides further guidance on this matter.

By expanding upon the foundational recipe, organizations can further integrate AI into their incident response strategies, enhancing their cybersecurity posture and preparedness.

Generating incident response playbooks

In the realm of cybersecurity, preparation is key. Incident response playbooks are vital tools that guide organizations through the process of handling various cyber threats. This recipe showcases how ChatGPT can be employed to generate these playbooks tailored to specific threats and environmental contexts. We'll walk through the process of crafting prompts for ChatGPT and interpreting its responses to create comprehensive playbooks. Additionally, we introduce a Python script that automates this process, further enhancing efficiency and readiness. By the end of this recipe, you'll have a way to rapidly generate detailed incident response playbooks, a critical component in fortifying your organization's cyber defense strategy.

Getting ready

Before diving into the recipe, ensure you have the following prerequisites in place:

- **Access to ChatGPT**: You'll need access to ChatGPT or the OpenAI API to interact with the language model. Ensure you have an API key if you're using the API.
- **Python environment**: If you plan to use the provided Python script, make sure you have Python installed on your system. The script is compatible with Python 3.6 and newer.
- **OpenAI Python library**: Install the `openai` Python library, which allows you to interact with the OpenAI API. You can install it using pip, `pip install openai`.

How to do it…

Follow these steps to harness the power of ChatGPT and Python in crafting playbooks that are both comprehensive and customized to your specific scenarios.

1. **Identify the threat and environment**: Before you can generate an incident response playbook, you must identify the specific threat type and the details of the environment it affects. This information is crucial, as it will guide the customization of your playbook.
2. **Craft the prompt**: With the threat and environment details in hand, construct a prompt that you will use to communicate with ChatGPT. Here's a template to follow:

   ```
   Create an incident response playbook for handling [Threat_Type]
   affecting [System/Network/Environment_Details].
   ```

   Replace `[Threat_Type]` with the specific threat you're preparing for, and replace `[System/Network/Environment_Details]` with the relevant details of your environment.
3. **Interact with ChatGPT**: Input your crafted prompt into ChatGPT. The AI will generate a response that outlines a detailed incident response playbook that is tailored to the threat and environment you specified.

4. **Review and refine**: Once you have the generated playbook, it's time to review it. Ensure that the playbook aligns with your organization's policies and procedures. Make any necessary customizations to fit your specific needs.
5. **Implement and train**: Disseminate the playbook among your incident response team members. Conduct training sessions to make sure everyone understands their roles and responsibilities as outlined in the playbook.
6. **Maintain and update**: The threat landscape is ever-evolving, and so should your playbooks. Regularly review and update your playbooks to incorporate new threats, vulnerabilities, and changes in your environment.

How it works…

The efficacy of the prompt in generating an incident response playbook hinges on its specificity and clarity. When you input the prompt, "Create an incident response playbook for handling `[Threat_Type]` affecting `[System/Network/Environment_Details]`," you're setting a clear task for ChatGPT:

- **Task understanding**: ChatGPT interprets the prompt as a request to create a structured document, recognizing terms such as *incident response playbook* and `handling [Threat_Type]` as indicators of the document's purpose and content.
- **Contextualization**: By specifying the threat type and environment details, you're providing context. ChatGPT uses this information to tailor the playbook, ensuring relevance to the specified scenario.
- **Structured response**: ChatGPT draws on its training data, which includes various cybersecurity materials, to structure the playbook. It typically includes sections on roles, responsibilities, and step-by-step procedures, aligning with standard formats of incident response documents.
- **Customization**: The model's ability to generate content based on the provided details results in a playbook that feels custom-made. It's not a generic template but a response crafted to address the specifics of the prompt.

This interaction between the prompt and ChatGPT showcases the model's capability to generate detailed, structured, and contextually relevant documents, making it a valuable tool for cybersecurity professionals.

There's more…

While the ChatGPT web interface provides a convenient way to interact with the AI, using a Python script and leveraging the OpenAI API can take the generation of incident response playbooks to the next level. This can be a more dynamic and automated approach.

The script introduces automation, customization, integration, scalability, programmatic control, and confidentiality, which are enhancements that significantly elevate the playbook creation process. It will prompt you for the threat type and environment details, construct the prompt dynamically, and then use the OpenAI API to generate the playbook. Here's how to set it up:

1. **Set up your environment**: Ensure you have Python installed on your system. You'll also need the `openai` library, which you can install using the following pip:

   ```
   pip install openai
   ```

2. **Obtain your API key**: You'll need an API key from OpenAI to use their models. Securely store this key and ensure it's not exposed in your code or version control systems.
3. **Create the OpenAI API call**: Create a new function that directs the model to generate the playbook:

   ```
   import openai
   from openai import OpenAI
   import os

   def generate_incident_response_playbook(threat_type,
   environment_details):
       """
       Generate an incident response playbook based on
       the provided threat type and environment details.
       """
       # Create the messages for the OpenAI API
       messages = [
           {"role": "system", "content": "You are an AI
              assistant helping to create an incident
                response playbook."},
           {"role": "user", "content": f"Create a
            detailed incident response playbook for
            handling a '{threat_type}' threat affecting
            the following environment: {environment_
              details}."}
       ]

       # Set your OpenAI API key here
   openai.api_key = os.getenv("OPENAI_API_KEY")

       # Make the API call
       try:
           client = OpenAI()
           response = client.chat.completions.create(
   ```

```
                model="gpt-3.5-turbo",
                messages=messages,
                max_tokens=2048,
                n=1,
                stop=None,
                temperature=0.7
            )
            response_content = response.choices[0].message.content.
strip()
            return response_content
        except Exception as e:
            print(f"An error occurred: {e}")
            return None
```

4. **Prompt for user input:** Enhance the script to collect the threat type and environment details from the user:

```
# Get input from the user
threat_type = input("Enter the threat type: ")
environment_details = input("Enter environment
  details: ")
```

5. **Generate and display the playbook**: Call the function with the user's input and print the generated playbook:

```
# Generate the playbook
playbook = generate_incident_response_playbook
  (threat_type, environment_details)

# Print the generated playbook
if playbook:
    print("\nGenerated Incident Response Playbook:")
    print(playbook)
else:
    print("Failed to generate the playbook.")
```

6. **Run the script**: Execute the script. It will prompt you for the threat type and environment details, and then it will display the generated incident response playbook.

Here's how the completed script should look:

```
import openai
from openai import OpenAI # Updated for the new OpenAI API
import os

# Set your OpenAI API key here
```

```
openai.api_key = os.getenv("OPENAI_API_KEY")

def generate_incident_response_playbook
  (threat_type, environment_details):
    """
    Generate an incident response playbook based on the
      provided threat type and environment details.
    """
    # Create the messages for the OpenAI API
    messages = [
        {"role": "system", "content": "You are an AI
          assistant helping to create an incident response
            playbook."},
        {"role": "user", "content": f"Create a detailed
          incident response playbook for handling a
            '{threat_type}' threat affecting the following
             environment: {environment_details}."}
    ]

    # Make the API call
    try:
        client = OpenAI()
        response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=messages,
            max_tokens=2048,
            n=1,
            stop=None,
            temperature=0.7
        )
        response_content = response.choices[0].message.content.strip()
        return response_content
    except Exception as e:
        print(f"An error occurred: {e}")
        return None

# Get input from the user
threat_type = input("Enter the threat type: ")
environment_details = input("Enter environment details: ")

# Generate the playbook
playbook = generate_incident_response_playbook
```

```
    (threat_type, environment_details)

# Print the generated playbook
if playbook:
    print("\nGenerated Incident Response Playbook:")
    print(playbook)
else:
    print("Failed to generate the playbook.")
```

The Python script provided acts as a bridge between the user and the OpenAI API, facilitating the generation of incident response playbooks. Here's a breakdown of how each part of the script contributes to this process:

1. **Importing dependencies**: The script begins by importing the `openai` library, which is the official Python client library provided by OpenAI. This library simplifies the interaction with the OpenAI API, allowing us to send prompts and receive responses.
2. **Defining the playbook generation function**: The `generate_incident_response_playbook` function is the core of the script. It's responsible for crafting the API request and parsing the response.

   API messages: The function constructs a list of messages that emulates a chat session. The first message sets the context for the AI (`"You are an AI assistant..."`), and the second message contains the user's prompt with the specific threat type and environment details.

   API call: Using the `openai.ChatCompletion.create` method, the function sends the messages to the chosen model. It specifies parameters such as `max_tokens` and `temperature` to control the length and creativity of the response.

   Error handling: The script includes a `try` and `except` block to gracefully handle any errors that may occur during the API call, such as network issues or invalid API keys.
3. **User interaction**: The script gathers input from the user through the `input` function. This is where the threat type and environment details are specified by the user.
4. **Generating and displaying the playbook**: Once the function receives the user input, it generates the prompt, sends it to the OpenAI API, and receives the playbook. The script then prints the generated playbook, giving the user an immediate view of the output.

This script is a practical example of how you can integrate OpenAI's powerful language model into your cybersecurity workflow, automating the generation of detailed and contextual incident response playbooks.

Note of caution

When generating incident response playbooks using ChatGPT or the OpenAI API, be mindful of the sensitivity of the information you're inputting. Avoid sending confidential or sensitive data to the API, as it could potentially be stored or logged. If your organization has stringent confidentiality requirements, consider utilizing private local language models. Stay tuned for an upcoming chapter where we'll explore how to deploy and use local language models, offering a more secure and private alternative for sensitive applications.

ChatGPT-assisted root cause analysis

When the digital alarms ring and systems flash red, incident responders are the first line of defense in the cybersecurity battlefield. Amidst the chaos of alerts and anomalies, identifying the root cause of a security incident is akin to finding the proverbial needle in a haystack. It requires a keen eye, a systematic approach, and, often, a touch of intuition. However, even the most seasoned professionals can benefit from a structured guide through the labyrinth of logs, alerts, and symptoms that define a security incident. This is where **ChatGPT-assisted root cause analysis** comes into play.

Envision ChatGPT as your digital Sherlock Holmes, a tireless incident response advisor equipped with the collective knowledge of cybersecurity practices and the analytical prowess of artificial intelligence. This recipe unveils a conversational blueprint that leads you through the fog of digital warfare, posing critical questions and suggesting investigative pathways based on your responses. It's a dynamic dialogue that evolves with each piece of information you provide, steering you toward the probable root causes of the incident.

Whether it's a mysterious spike in network traffic, an unexpected system shutdown, or a subtle anomaly in user behavior, ChatGPT's inquisitive nature ensures no stone is left unturned. By leveraging the power of generative AI, this recipe empowers you to peel back the layers of the incident, guiding you from the initial symptoms to the underlying vulnerabilities that adversaries may have exploited.

This recipe is more than a set of instructions; it's a collaborative journey with an AI companion that's committed to aiding you in safeguarding your digital realm. So, prepare to embark on a quest to demystify the complexities of incident response and root cause analysis with ChatGPT as your guide.

Getting ready

Before diving into the heart of root cause analysis with ChatGPT, it's essential to set the stage for an effective session. This involves ensuring that you have access to the necessary information and tools and are prepared to interact with ChatGPT in a way that maximizes its potential as an incident response advisor.

- **Access to ChatGPT**: Make sure you have access to ChatGPT, preferably through the Web UI, for ease of interaction. If you're using the OpenAI API, ensure that your environment is properly configured to send and receive messages from the model.

- **Incident data**: Gather all relevant data associated with the security incident. This may include logs, alerts, network traffic data, system statuses, and any observations noted by the security team. Having this information at hand will be crucial for providing context to ChatGPT.
- **Secure environment**: Ensure that you're operating in a secure environment when interacting with ChatGPT. Be mindful of the sensitivity of the data you're discussing, and follow your organization's data handling and privacy policies.
- **Familiarity with incident response protocols**: While ChatGPT can guide you through the analysis, a foundational understanding of your organization's incident response protocols and procedures will enhance the collaboration.

By meeting these prerequisites, you'll be well-positioned to engage with ChatGPT effectively and embark on a structured journey to uncover the root cause of the security incident at hand.

How to do it...

Root cause analysis in incident response is an intricate dance of queries and deductions. With ChatGPT as your partner, this dance becomes a structured dialogue, each step bringing you closer to understanding the underlying cause of the incident. Follow these steps to leverage ChatGPT's capabilities in your incident response endeavors:

1. **Initiate the session**: Begin by clearly stating your intent to ChatGPT. Provide the following prompt:

   ```
   You are my incident response advisor. Help me identify the root
   cause of the observed suspicious activities.
   ```

2. **Describe the symptoms**: Provide a detailed description of the first symptoms or anomalies you observed. This could include unusual system behavior, unexpected alerts, or any other indicators of a potential security incident.
3. **Answer ChatGPT's questions**: ChatGPT will respond with a series of questions to narrow down the potential causes. These may include inquiries about unauthorized access alerts, unusual network traffic, or commonalities among affected systems. Answer these questions to the best of your knowledge.
4. **Follow the decision trees**: Based on your responses, ChatGPT will guide you through a decision tree, suggesting possible root causes and further investigative steps. This interactive process is designed to consider various scenarios and their likelihood based on the information provided.
5. **Investigate and validate**: Use the suggestions provided by ChatGPT to conduct further investigations. Validate the hypotheses by checking against logs, system configurations, and other relevant data.
6. **Iterate as needed**: Incident response is rarely linear. As you uncover new information, return to ChatGPT with your findings to refine the analysis. The model's responses will adapt based on the evolving situation.

7. **Document and report**: Once you've identified the probable root causes, document your findings and report them according to your organization's protocols. This documentation can be invaluable for future incident response efforts and for strengthening your security posture.

By following these steps, you can transform the daunting task of root cause analysis into a structured and manageable process, with ChatGPT serving as a knowledgeable advisor every step of the way.

How it works…

The simplicity of the initial prompt, "You are my incident response advisor. Help me identify the root cause of the observed suspicious activities," belies its effectiveness. This prompt sets the stage for a focused and purpose-driven interaction with ChatGPT. Here's why it works:

- **Clarity of the role**: By explicitly defining ChatGPT's role as an incident response advisor, we prime the AI to adopt a specific mindset geared towards problem-solving within the realm of cybersecurity incident response. This helps tailor the subsequent conversation towards actionable insights and guidance.
- **Open-ended inquiry**: The request to help me identify the root cause is intentionally open-ended, inviting ChatGPT to ask probing questions. This approach mimics the Socratic method, leveraging inquiry to stimulate critical thinking and illuminate the path toward understanding the incident's root cause.
- **Focus on suspicious activities**: The mention of observed suspicious activities provides a context for the analysis, signaling ChatGPT to concentrate on anomalies and potential indicators of compromise. This focus helps narrow down the line of questioning and analysis, making the interaction more efficient.

In the context of incident response, root cause analysis often involves sifting through a maze of symptoms, logs, and behaviors to trace back to the origin of the security incident. ChatGPT assists in this process by doing the following:

- **Asking targeted questions**: Based on the initial prompt and subsequent inputs, ChatGPT asks targeted questions that help isolate variables and identify patterns. This can help incident responders focus their attention on the most relevant areas of investigation.
- **Suggesting hypotheses**: As the conversation unfolds, ChatGPT suggests potential root causes based on the information provided. These hypotheses can serve as starting points for deeper investigation.
- **Guiding investigation**: Through its questions and suggestions, ChatGPT can guide incident responders in checking specific logs, monitoring certain network traffic, or examining affected systems more closely.
- **Providing educational insights**: If there are gaps in understanding or if clarification is needed on a specific cybersecurity concept, ChatGPT can provide explanations and insights, enhancing the educational value of the interaction.

In essence, ChatGPT acts as a catalyst for critical thinking and structured analysis, helping incident responders navigate the complex web of potential causes behind a security incident.

There's more…

While the steps outlined in the previous section provide a solid framework for conducting root cause analysis with ChatGPT, there are additional considerations and strategies that can further enrich the process:

- **Leveraging ChatGPT's knowledge base**: ChatGPT has been trained on a diverse set of data, including cybersecurity concepts and incidents. Don't hesitate to ask for explanations or clarifications on security terms, attack vectors, or remediation strategies.
- **Contextualizing the conversation**: As you interact with ChatGPT, provide as much context as possible. The more detailed and specific your inputs are, the more tailored and relevant ChatGPT's guidance will be.
- **Exploring multiple hypotheses**: Often, there may be more than one plausible root cause. Use ChatGPT to explore various hypotheses simultaneously, comparing and contrasting their likelihood based on the evidence at hand.
- **Incorporating external tools**: ChatGPT can suggest tools and techniques for deeper analysis. Whether it's recommending a network analysis tool or a specific log query, integrating these suggestions can provide a more comprehensive view of the incident.
- **Continuous learning**: Each incident response engagement is an opportunity to learn. Reflect on the dialogue with ChatGPT, noting which questions and decision paths were most helpful. This can inform and improve future interactions.
- **Feedback loop**: Provide feedback to ChatGPT about the accuracy and usefulness of its suggestions. This can help refine the model's responses over time, making it an even more effective advisor for incident response.

By incorporating these additional strategies, you can maximize the value of ChatGPT in your root cause analysis efforts, turning it into a powerful ally in the quest to safeguard your digital assets.

Notes of caution

As you engage with ChatGPT for root cause analysis in incident response scenarios, it's crucial to remain vigilant about the sensitivity of the information being discussed. Remember that while ChatGPT can be an invaluable advisor, it operates within the constraints of its training and the information it has been provided. It's not privy to the confidential details of your organization's security infrastructure or incident specifics unless you share them.

Therefore, exercise caution and adhere to your organization's data handling and privacy policies when interacting with ChatGPT. Avoid sharing sensitive or identifiable information that could compromise your organization's security posture. In the upcoming chapter on private local LLMs, we'll explore how to leverage the benefits of language models, such as ChatGPT, in a more controlled and secure environment, mitigating the risks associated with transmitting sensitive data.

By staying mindful of these considerations, you can harness the power of ChatGPT for effective root cause analysis while maintaining the integrity and security of your organization's information.

Automated briefing reports and incident timeline reconstruction

Generative AI and **LLMs** offer profound enhancements to threat monitoring capabilities. By leveraging the sophisticated understanding of language and context inherent in these models, cybersecurity systems can now analyze and interpret vast volumes of data with a level of nuance and depth previously unattainable. This transformative technology enables the identification of subtle anomalies, patterns, and potential threats hidden within complex datasets, providing a more proactive and predictive approach to security. The integration of generative AI and LLMs into cybersecurity workflows not only augments the efficiency and accuracy of threat detection but also significantly reduces the response time to emerging threats, thereby fortifying digital infrastructures against sophisticated cyber-attacks.

In this recipe, we delve into the innovative application of OpenAI's embeddings API/model alongside **Facebook AI Similarity Search** (**FAISS**) to elevate the analysis of cybersecurity log files. By harnessing the power of AI-driven embeddings, we aim to capture the nuanced semantic content of log data, transforming it into a format conducive to mathematical analysis. Coupled with the efficiency of FAISS for rapid similarity searches, this approach enables us to categorize log entries with unprecedented precision, identifying potential security incidents by their likeness to known patterns. This recipe is designed to provide you with a practical, step-by-step guide to integrating these cutting-edge technologies into your cybersecurity toolkit, offering a robust method for sifting through log data and enhancing your security posture.

Getting ready

Before we begin scripting our automated briefing reports and incident timeline reconstruction, there are a few prerequisites to ensure everything runs smoothly:

- **Python environment**: Ensure you have Python installed on your system. This script is compatible with Python 3.6 and newer.
- **OpenAI API key**: You'll need access to the OpenAI API. Obtain your API key from the OpenAI platform, as it will be crucial for interacting with ChatGPT and the embedding model.
- **Required Libraries**: Install the `openai` library, which allows for seamless communication with the OpenAI API. You can install it using pip: `pip install openai`. You will also need the `numpy` and `faiss` libraries, which can also be installed using pip.

- **Log Data**: Have your incident logs ready. These logs can be in any format, but for the purpose of this script, we'll assume they are in text format, containing timestamps and event descriptions. Sample log files are provided in the GitHub repository, along with a script that will allow you to generate sample log data.
- **Secure environment**: Ensure that you're working in a secure environment, especially when handling sensitive data. As we'll discuss in a later chapter, using private local LLMs can enhance data security.

Once you have these prerequisites in place, you're ready to dive into the script and begin crafting your automated incident reports.

How to do it…

The following steps will guide you through creating a Python script for analyzing log files with AI-powered embeddings and FAISS (Facebook AI Similarity Search) for efficient similarity search. The task involves parsing log files, generating embeddings for log entries, and categorizing them as "Suspicious" or "Normal" based on their similarity to predefined templates.

1. **Import Required Libraries**: Start by importing necessary Python libraries for handling API requests, regular expressions, numerical operations, and similarity search.

   ```
   import openai
   from openai import OpenAI
   import re
   import os
   import numpy as np
   import faiss
   ```

2. **Initialize the OpenAI Client**: Set up the OpenAI client and configure it with your API key. This is crucial for accessing the embeddings API.

   ```
   client = OpenAI()
   openai.api_key = os.getenv("OPENAI_API_KEY")
   ```

3. **Parse the Raw Log File**: Define a function to parse the raw log file into a JSON format. This function uses regular expressions to extract timestamps and event descriptions from the log entries.

   ```
   def parse_raw_log_to_json(raw_log_path):
       timestamp_regex = r'\[\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\]'
       event_regex = r'Event: (.+)'
       json_data = []
       with open(raw_log_path, 'r') as file:
           for line in file:
               timestamp_match = re.search(timestamp_regex, line)
               event_match = re.search(event_regex, line)
   ```

```
            if timestamp_match and event_match:
                json_data.append({"Timestamp": timestamp_match.
group().strip('[]'), "Event": event_match.group(1)})
    return json_data
```

4. **Generate the Embeddings**: Create a function to generate embeddings for a given list of text strings using the OpenAI API. This function handles the API response and extracts the embedding vectors.

```
def get_embeddings(texts):
    embeddings = []
    for text in texts:
        response = client.embeddings.create(input=text,
model="text-embedding-ada-002")
        try:
            embedding = response['data'][0]['embedding']
        except TypeError:
            embedding = response.data[0].embedding
        embeddings.append(embedding)
    return np.array(embeddings)
```

5. **Create the FAISS Index**: Define a function to create a FAISS index for efficient similarity search. This index is later used to find the nearest template embeddings to a given log entry embedding.

```
def create_faiss_index(embeddings):
    d = embeddings.shape[1]
    index = faiss.IndexFlatL2(d)
    index.add(embeddings.astype(np.float32))
    return index
```

6. **Analize the Logs and Categorize Entries**: Implement the function to analyze log entries and categorize them based on their similarity to predefined "suspicious" and "normal" templates. This function utilizes the FAISS index for nearest neighbor searches.

```
def analyze_logs_with_embeddings(log_data):
    suspicious_templates = ["Unauthorized access attempt
detected", "Multiple failed login attempts"]
    normal_templates = ["User logged in successfully", "System
health check completed"]
    suspicious_embeddings = get_embeddings(suspicious_templates)
    normal_embeddings = get_embeddings(normal_templates)
    template_embeddings = np.vstack((suspicious_embeddings,
normal_embeddings))
    index = create_faiss_index(template_embeddings)
    labels = ['Suspicious'] * len(suspicious_embeddings) +
['Normal'] * len(normal_embeddings)
    categorized_events = []
    for entry in log_data:
        log_embedding = get_embeddings([entry["Event"]]).
astype(np.float32)
        _, indices = index.search(log_embedding, k=1)
        categorized_events.append((entry["Timestamp"],
entry["Event"], labels[indices[0][0]]))
    return categorized_events
```

7. **Process the results**: Finally, use the defined functions to parse a sample log file, analyze the logs, and print the categorized timeline.

```
raw_log_file_path = 'sample_log_file.txt'
log_data = parse_raw_log_to_json(raw_log_file_path)
categorized_timeline = analyze_logs_with_embeddings(log_data)
for timestamp, event, category in categorized_timeline:
    print(f"{timestamp} - {event} - {category}")
```

Here is how the completed script should look:

```
import openai
from openai import OpenAI  # Updated for the new OpenAI API
import re
import os
import numpy as np
import faiss  # Make sure FAISS is installed

client = OpenAI()  # Updated for the new OpenAI API

# Set your OpenAI API key here
openai.api_key = os.getenv("OPENAI_API_KEY")

def parse_raw_log_to_json(raw_log_path):
    #Parses a raw log file and converts it into a JSON format.
    # Regular expressions to match timestamps and event descriptions
in the raw log
    timestamp_regex = r'\[\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\]'
    event_regex = r'Event: (.+)'

    json_data = []

    with open(raw_log_path, 'r') as file:
        for line in file:
            timestamp_match = re.search(timestamp_regex, line)
            event_match = re.search(event_regex, line)

            if timestamp_match and event_match:
                timestamp = timestamp_match.group().strip('[]')
                event_description = event_match.group(1)
                json_data.append({"Timestamp": timestamp, "Event":
event_description})

    return json_data

def get_embeddings(texts):
    embeddings = []
    for text in texts:
        response = client.embeddings.create(
            input=text,
            model="text-embedding-ada-002"  # Adjust the model as
needed
        )
        try:
            # Attempt to access the embedding as if the response is a
dictionary
            embedding = response['data'][0]['embedding']
        except TypeError:
            # If the above fails, access the embedding assuming
'response' is an object with attributes
            embedding = response.data[0].embedding

        embeddings.append(embedding)

    return np.array(embeddings)

def create_faiss_index(embeddings):
    # Creates a FAISS index for a given set of embeddings.
    d = embeddings.shape[1]  # Dimensionality of the embeddings
    index = faiss.IndexFlatL2(d)
    index.add(embeddings.astype(np.float32))  # FAISS expects float32
    return index

def analyze_logs_with_embeddings(log_data):
    # Define your templates and compute their embeddings
    suspicious_templates = ["Unauthorized access attempt detected",
"Multiple failed login attempts"]
    normal_templates = ["User logged in successfully", "System health
check completed"]
    suspicious_embeddings = get_embeddings(suspicious_templates)
    normal_embeddings = get_embeddings(normal_templates)

    # Combine all template embeddings and create a FAISS index
    template_embeddings = np.vstack((suspicious_embeddings, normal_
embeddings))
    index = create_faiss_index(template_embeddings)

    # Labels for each template
    labels = ['Suspicious'] * len(suspicious_embeddings) + ['Normal']
* len(normal_embeddings)

    categorized_events = []

    for entry in log_data:
        # Fetch the embedding for the current log entry
        log_embedding = get_embeddings([entry["Event"]]).astype(np.
float32)

        # Perform the nearest neighbor search with FAISS
        k = 1  # Number of nearest neighbors to find
        _, indices = index.search(log_embedding, k)

        # Determine the category based on the nearest template
        category = labels[indices[0][0]]
        categorized_events.append((entry["Timestamp"], entry["Event"],
category))

    return categorized_events

# Sample raw log file path
raw_log_file_path = 'sample_log_file.txt'

# Parse the raw log file into JSON format
log_data = parse_raw_log_to_json(raw_log_file_path)

# Analyze the logs
categorized_timeline = analyze_logs_with_embeddings(log_data)

# Print the categorized timeline
for timestamp, event, category in categorized_timeline:
    print(f"{timestamp} - {event} - {category}")
```

By completing this recipe, you have harnessed the power of generative AI to automate the creation of briefing reports and the reconstruction of incident timelines from log data. This approach not only helps streamline the process of incident analysis but it can also enhance the accuracy and depth of your cybersecurity investigations, empowering your team to make informed decisions based on structured and insightful data narratives.

How it works...

This recipe provides a sophisticated tool designed to analyze log files using artificial intelligence and efficient similarity search techniques. It leverages the power of OpenAI's embeddings to understand the semantic content of log entries and employs FAISS for rapid similarity searches, categorizing each entry based on its resemblance to predefined templates. This approach allows for an advanced analysis of log data, identifying potential security incidents by comparing them against known patterns of suspicious and normal activities.

- *Importing Libraries*: The script begins by importing essential libraries. `openai` is used to interact with the OpenAI API for generating embeddings. `re` is for regular expressions, crucial for parsing log files. `os` allows the script to interact with the operating system, such as accessing environment variables. `numpy` provides support for arrays and numerical operations, and `faiss` is imported for fast similarity searches within high-dimensional spaces of embeddings.
- *Initializing OpenAI Client*: An instance of the OpenAI client is created, and the API key is set up. This client is necessary for making requests to the OpenAI API, specifically to generate text embeddings that capture the semantic meaning of the log entries and templates.
- *Parsing Log Files*: The `parse_raw_log_to_json` function reads a raw log file line by line, using regular expressions to extract and structure timestamps and event descriptions into a JSON-like format. This structured data is essential for the subsequent analysis, as it provides a clear separation of the time and content of each log entry.
- *Generating Embeddings*: The `get_embeddings` function interacts with the OpenAI API to convert textual data (log entries and templates) into numerical vectors, known as embeddings. These embeddings are dense representations that capture the semantic nuances of the text, enabling mathematical operations such as similarity comparisons.
- *Creating FAISS Index*: With the `create_faiss_index` function, the script sets up a FAISS index for the embeddings of the predefined templates. FAISS is optimized for fast similarity search in large datasets, making it ideal for quickly finding the most similar template to a given log entry embedding.
- *Analyzing Logs and Categorizing Entries*: In the `analyze_logs_with_embeddings` function, the script first generates embeddings for the log entries and predefined templates. It then uses the FAISS index to find the nearest template embedding to each log entry embedding. The category of the nearest template (either "Suspicious" or "Normal") is assigned to the log entry. This step is where the core analysis happens, utilizing the semantic understanding provided by embeddings and the efficiency of FAISS for similarity searches.
- *Processing the Results*: Finally, the script puts everything together by parsing a sample log file, analyzing the log data, and printing out a categorized timeline of events. This output provides insights into the log entries, highlighting potential security issues based on their similarity to the "suspicious" templates.

This script exemplifies how AI and similarity search technologies can be combined to enhance log file analysis, offering a more nuanced understanding of log data than traditional keyword-based approaches. By leveraging embeddings, the script can grasp the contextual meaning behind log entries, and with FAISS, it can efficiently categorize vast numbers of entries, making it a powerful tool for security analysis and incident detection.

There's more...

The script you've built opens up a range of possibilities for enhancing cybersecurity practices through the application of AI and efficient data processing techniques. By analyzing log files with embeddings and FAISS, you're not just categorizing events based on their similarity to predefined templates; you're laying the groundwork for a more intelligent, responsive, and adaptive cybersecurity infrastructure. Here are some ideas on how you can expand upon this concept and leverage this type of script for broader applications in cybersecurity:

1. **Adapting to different log formats**: The script includes a function to parse raw log files into a JSON format. However, log formats can vary widely across different systems and devices. You may need to modify the regular expressions or parse logic by using the `parse_raw_log_to_json` function to accommodate the specific format of the logs you're working with. Developing a flexible parsing function or using a log management tool that normalizes log data can significantly streamline this process.
2. **Handling larger datasets**: Despite the efficiency of embeddings, as the volume of log data grows, you may still need to optimize the script for performance. Consider batch processing the log entries or parallelizing the analysis to handle larger datasets efficiently. These optimizations ensure the script remains scalable and can handle increased workloads without consuming excessive resource
3. **Anomaly Detection**: Extend the script to identify anomalies or outliers in log data that don't closely match any of the predefined templates. This could be crucial for detecting novel attacks or security breaches that don't follow known patterns.
4. **Real-time Monitoring**: Adapt the script for real-time log analysis by integrating it with live data feeds. This would allow for immediate detection and alerting of suspicious activities, minimizing the response time to potential threats.
5. **Automated Response Systems**: Combine the script with automated response mechanisms that can take predefined actions when certain types of suspicious activities are detected, such as isolating affected systems or blocking IP addresses.
6. **User Behavior Analytics (UBA)**: Use the script as a foundation for developing a UBA system, which could analyze log data to model and monitor user behavior, identifying potentially malicious activities based on deviations from established patterns.

7. **Integration with Security Information and Event Management (SIEM) Systems**: Integrate the script's capabilities with SIEM systems to enhance their ability to analyze, visualize, and respond to security data, adding an AI-powered layer to the analysis.
8. **Threat Intelligence Feeds**: Incorporate threat intelligence feeds into the script to dynamically update the list of suspicious and normal templates based on the latest intelligence, keeping the system adaptive to evolving threats.
9. **Forensic Analysis**: Utilize the script's capabilities in forensic analysis to sift through large volumes of historical log data, uncovering details of security incidents and breaches by identifying patterns and anomalies.
10. **Customizable Alerting Thresholds**: Implement customizable threshold settings that control when an event is categorized as suspicious, allowing for tuning based on the sensitivity and specificity requirements of different environments.
11. **Scalability Enhancements**: Explore ways to scale the script for handling massive datasets by leveraging distributed computing resources or cloud-based services, ensuring it can manage the volume of data generated by large-scale networks.

By exploring these avenues, you can significantly enhance the script's utility and impact in cybersecurity, moving towards a more proactive and data-driven security posture. Each expansion not only increases the script's capabilities but also contributes to a deeper understanding and more effective management of cybersecurity risks.

Notes of caution

When using this script, especially in a cybersecurity context, it's imperative to be mindful of the sensitivity of the data being processed. Log files often contain confidential information that should not be exposed outside of your secure environment. While the OpenAI API provides powerful tools for analyzing and categorizing log data, it's crucial to ensure that sensitive information is not inadvertently sent to external servers.

As an additional measure of caution, consider anonymizing your data before sending it to the API or using techniques, such as differential privacy, to add an extra layer of security.

Moreover, if you're looking for an approach that keeps all data processing within your local environment, stay tuned for an upcoming chapter on private local LLMs. This chapter will explore how you can leverage the capabilities of LLMs while maintaining strict control over your data, ensuring that sensitive information remains within the confines of your secure systems.

By being vigilant about data security, you can harness the power of AI in your cybersecurity efforts without compromising the confidentiality and integrity of your data.

9
Using Local Models and Other Frameworks

In this chapter, we explore the transformative potential of local AI models and frameworks in cybersecurity. We begin by leveraging **LMStudio** to deploy and interact with AI models locally, enhancing privacy and control in data-sensitive scenarios. **Open Interpreter** is then introduced as a tool for advanced local threat hunting and system analysis, followed by **Shell GPT**, which significantly augments penetration testing with NLP capabilities. We delve into **PrivateGPT** for its prowess in reviewing sensitive documents such as **Incident Response** (**IR**) Plans, ensuring data remains confidential. Finally, **Hugging Face AutoTrain** is showcased for its ability to fine-tune LLMs specifically for cybersecurity applications, exemplifying the integration of cutting-edge AI into various cybersecurity contexts. This chapter not only guides through practical applications but also imparts knowledge on effectively utilizing these tools for a range of cybersecurity tasks.

Important note

Open source **large language models** (**LLMs**) offer an alternative to popular proprietary models such as those from OpenAI. These open source models are developed and maintained by a community of contributors, making their source code and training data publicly accessible. This transparency allows greater customization, scrutiny, and understanding of the models, fostering innovation and trust.

The importance of open source LLMs lies in their accessibility and adaptability. They enable researchers, developers, and organizations, especially those with limited resources, to experiment with and deploy AI technologies without the constraints of licensing or cost associated with proprietary models. Moreover, open source LLMs encourage collaborative development, ensuring a broader range of perspectives and uses, which is vital for progress in AI and its application in diverse fields, including cybersecurity.

In this chapter, we will cover the following recipes:

- Implementing local AI models for cybersecurity analysis with LMStudio
- Local threat hunting with Open Interpreter
- Enhancing penetration testing with Shell GPT
- Reviewing IR Plans with PrivateGPT
- Fine-tuning LLMs for cybersecurity with Hugging Face's AutoTrain

Technical requirements

For this chapter, you will need a web browser and a stable internet connection to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account set up and have obtained your API key. If not, revisit *Chapter 1* for details. Basic familiarity with the Python programming language and working with the command line is necessary, as you'll be using **Python 3.x**, which needs to be installed on your system, for working with the OpenAI GPT API and creating Python scripts. A code editor will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter. Finally, since many penetration testing use cases rely heavily on the Linux operating system, access to and familiarity with a Linux distribution (preferably Kali Linux) is recommended. A basic understanding of command-line tools and shell scripting will be beneficial for interacting with tools such as Open Interpreter and Shell GPT. The code files for this chapter can be found here: `https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

Implementing local AI models for cybersecurity analysis with LMStudio

LMStudio has emerged as a powerful and user-friendly tool for LLMs locally and is suitable for both personal experimentation and professional application development in cybersecurity. Its user-friendly interface and cross-platform availability make it an attractive choice for a wide range of users, including cybersecurity professionals. Key features, such as model selection from **Hugging Face**, an interactive chat interface, and efficient model management, make LMStudio ideal for deploying and running open source LLMs on local machines. This recipe will explore how to use LMStudio for cybersecurity analysis, allowing you to interact with models directly or integrate them into applications via a local server.

Getting ready

Before we begin, ensure you have the following prerequisites:

- A computer with internet access for initial setup.
- Basic knowledge of AI models and familiarity with API interactions.

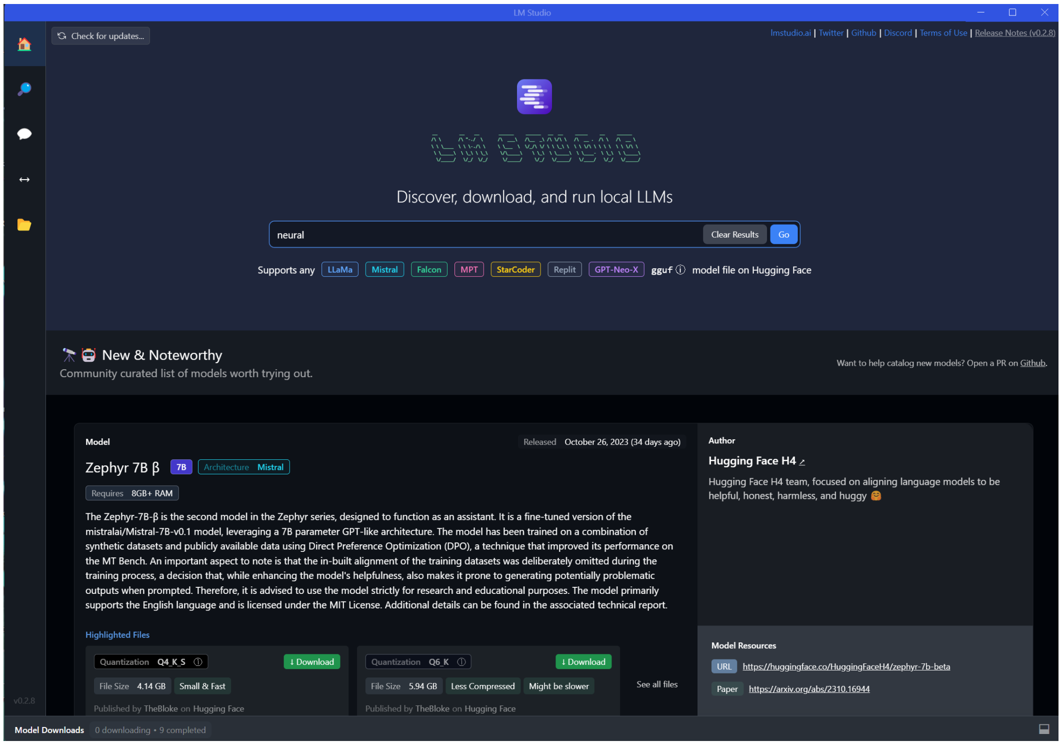
- LMStudio software downloaded and installed. Refer to the LMStudio's official website (`https://lmstudio.ai/`) and GitHub repository (`https://github.com/lmstudio-ai`) for installation instructions.

How to do it…

LMStudio offers a versatile platform for deploying and experimenting with LLMs locally. Here's how to maximize its use for cybersecurity analysis:

1. **Install and configure LMStudio**:
   - Download and install LMStudio for your operating system from `https://lmstudio.ai/`
   - Search for, choose, and download models from the Hugging Face Hub that suit your cybersecurity needs

   The following screenshot shows the LMStudio home screen.

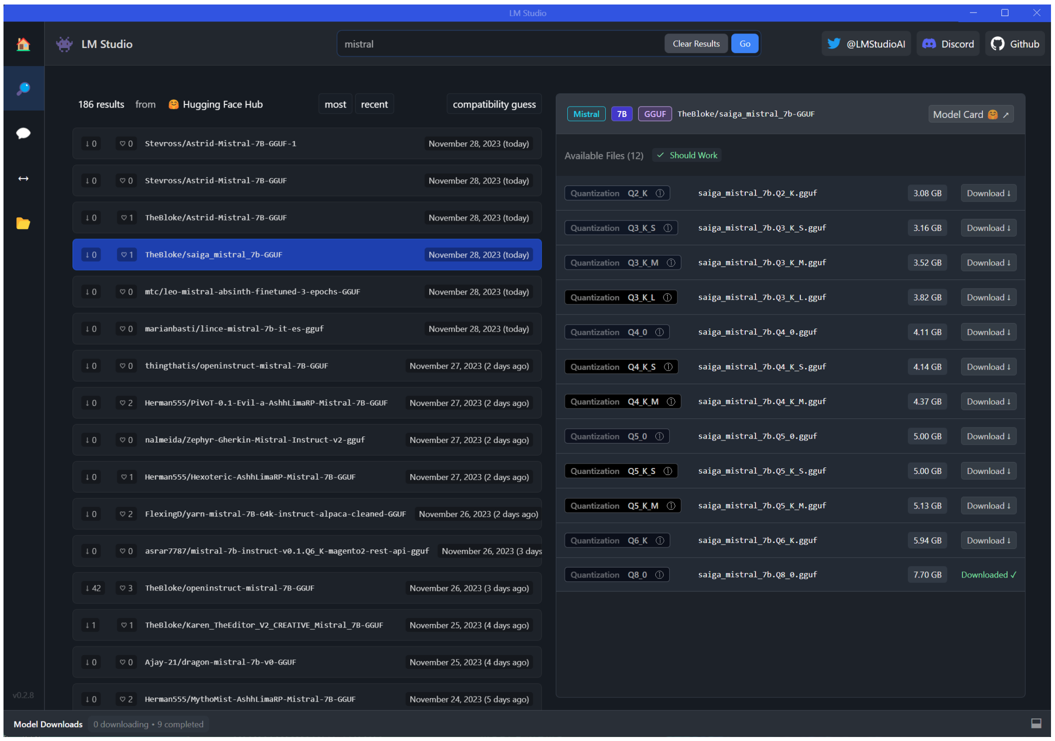

LM Studio
Check for updates...
lmstudio.ai | Twitter | Github | Discord | Terms of Use | Release Notes (v0.2.8)
Discover, download, and run local LLMs
neural
Clear Results
Go
Supports any LLaMa Mistral Falcon MPT StarCoder Replit GPT-Neo-X gguf model file on Hugging Face
New & Noteworthy
Community curated list of models worth trying out.
Want to help catalog new models? Open a PR on Github.
Model
Released October 26, 2023 (34 days ago)
Zephyr 7B β 7B Architecture Mistral
Requires 8GB+ RAM
Highlighted Files
Quantization Q4_K_S
↓ Download
File Size 4.14 GB
Small & Fast
Published by TheBloke on Hugging Face
Quantization Q6_K
↓ Download
File Size 5.94 GB
Less Compressed
Might be slower
Published by TheBloke on Hugging Face
See all files
Author
Hugging Face H4
Model Resources
URL https://huggingface.co/HuggingFaceH4/zephyr-7b-beta
Paper https://arxiv.org/abs/2310.16944
v0.2.8
Model Downloads 0 downloading • 9 completed

The Zephyr-7B-β is the second model in the Zephyr series, designed to function as an assistant. It is a fine-tuned version of the mistralai/Mistral-7B-v0.1 model, leveraging a 7B parameter GPT-like architecture. The model has been trained on a combination of synthetic datasets and publicly available data using Direct Preference Optimization (DPO), a technique that improved its performance on the MT Bench. An important aspect to note is that the in-built alignment of the training datasets was deliberately omitted during the training process, a decision that, while enhancing the model's helpfulness, also makes it prone to generating potentially problematic outputs when prompted. Therefore, it is advised to use the model strictly for research and educational purposes. The model primarily supports the English language and is licensed under the MIT License. Additional details can be found in the associated technical report.

Hugging Face H4 team, focused on aligning language models to be helpful, honest, harmless, and huggy 🤗

Figure 9.1 – LMStudio home screen

Available models are found in the search tab.



186 results from Hugging Face Hub

↓ 0	♡ 0	Stevross/Astrid-Mistral-7B-GGUF-1	November 28, 2023 (today)
↓ 0	♡ 0	Stevross/Astrid-Mistral-7B-GGUF	November 28, 2023 (today)
↓ 0	♡ 1	TheBloke/Astrid-Mistral-7B-GGUF	November 28, 2023 (today)
↓ 0	♡ 1	TheBloke/saiga_mistral_7b-GGUF	November 28, 2023 (today)
↓ 0	♡ 0	mtc/leo-mistral-absinth-finetuned-3-epochs-GGUF	November 28, 2023 (today)
↓ 0	♡ 0	marianbasti/lince-mistral-7b-it-es-gguf	November 28, 2023 (today)
↓ 0	♡ 0	thingthatis/openinstruct-mistral-7B-GGUF	November 27, 2023 (2 days ago)
↓ 0	♡ 2	Herman555/PiVoT-0.1-Evil-a-AshhLimaRP-Mistral-7B-GGUF	November 27, 2023 (2 days ago)
↓ 0	♡ 0	nalmeida/Zephyr-Gherkin-Mistral-Instruct-v2-gguf	November 27, 2023 (2 days ago)
↓ 0	♡ 1	Herman555/Hexoteric-AshhLimaRP-Mistral-7B-GGUF	November 27, 2023 (2 days ago)
↓ 0	♡ 2	FlexingD/yarn-mistral-7B-64k-instruct-alpaca-cleaned-GGUF	November 26, 2023 (2 days ago)
↓ 0	♡ 0	asrar7787/mistral-7b-instruct-v0.1.Q6_K-magento2-rest-api-gguf	November 26, 2023 (3 days
↓ 42	♡ 3	TheBloke/openinstruct-mistral-7B-GGUF	November 26, 2023 (3 days ago)
↓ 1	♡ 1	TheBloke/Karen_TheEditor_V2_CREATIVE_Mistral_7B-GGUF	November 25, 2023 (4 days ago)
↓ 0	♡ 0	Ajay-21/dragon-mistral-7b-v0-GGUF	November 25, 2023 (4 days ago)
↓ 0	♡ 2	Herman555/MythoMist-AshhLimaRP-Mistral-7B-GGUF	November 24, 2023 (5 days ago)

TheBloke/saiga_mistral_7b-GGUF

Available Files (12) ✓ Should Work

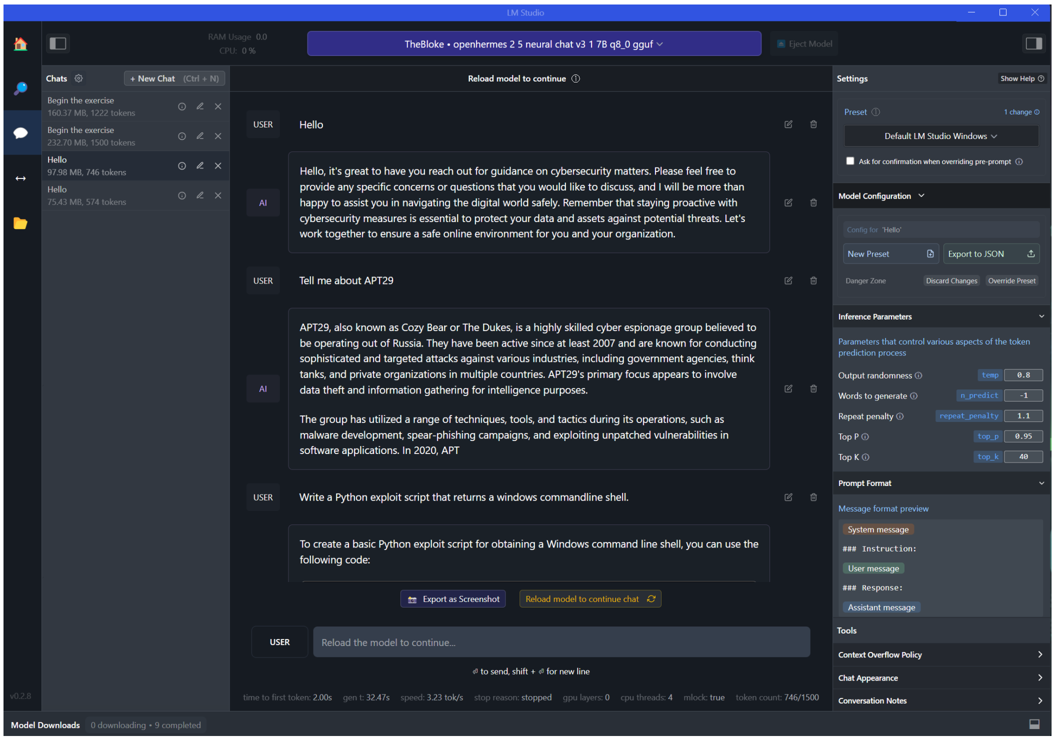
Quantization Q2_K	saiga_mistral_7b.Q2_K.gguf	3.08 GB	Download ↓
Quantization Q3_K_S	saiga_mistral_7b.Q3_K_S.gguf	3.16 GB	Download ↓
Quantization Q3_K_M	saiga_mistral_7b.Q3_K_M.gguf	3.52 GB	Download ↓
Quantization Q3_K_L	saiga_mistral_7b.Q3_K_L.gguf	3.82 GB	Download ↓
Quantization Q4_0	saiga_mistral_7b.Q4_0.gguf	4.11 GB	Download ↓
Quantization Q4_K_S	saiga_mistral_7b.Q4_K_S.gguf	4.14 GB	Download ↓
Quantization Q4_K_M	saiga_mistral_7b.Q4_K_M.gguf	4.37 GB	Download ↓
Quantization Q5_0	saiga_mistral_7b.Q5_0.gguf	5.00 GB	Download ↓
Quantization Q5_K_S	saiga_mistral_7b.Q5_K_S.gguf	5.00 GB	Download ↓
Quantization Q5_K_M	saiga_mistral_7b.Q5_K_M.gguf	5.13 GB	Download ↓
Quantization Q6_K	saiga_mistral_7b.Q6_K.gguf	5.94 GB	Download ↓
Quantization Q8_0	saiga_mistral_7b.Q8_0.gguf	7.70 GB	Downloaded ✓

Model Downloads 0 downloading • 9 completed

Figure 9.2 – Model selection and installation

2. **Interact with models using the Chat interface**:
   - Once the model is installed, use the Chat panel to activate and load the selected model.
   - Use the model for cybersecurity queries in a no-internet-required setup.
   - In most cases, the default model settings are already tuned for the specific model. However, you can modify the default presets for the model to optimize its performance according to your needs, similar to how the parameters work with OpenAI models.

The chat tab allows direct chat with the model from the user interface.



Begin the exercise
160.37 MB, 1222 tokens

Begin the exercise
232.70 MB, 1500 tokens

Hello
97.98 MB, 746 tokens

Hello
75.43 MB, 574 tokens

USER Hello

AI Hello, it's great to have you reach out for guidance on cybersecurity matters. Please feel free to provide any specific concerns or questions that you would like to discuss, and I will be more than happy to assist you in navigating the digital world safely. Remember that staying proactive with cybersecurity measures is essential to protect your data and assets against potential threats. Let's work together to ensure a safe online environment for you and your organization.

USER Tell me about APT29

AI APT29, also known as Cozy Bear or The Dukes, is a highly skilled cyber espionage group believed to be operating out of Russia. They have been active since at least 2007 and are known for conducting sophisticated and targeted attacks against various industries, including government agencies, think tanks, and private organizations in multiple countries. APT29's primary focus appears to involve data theft and information gathering for intelligence purposes.

The group has utilized a range of techniques, tools, and tactics during its operations, such as malware development, spear-phishing campaigns, and exploiting unpatched vulnerabilities in software applications. In 2020, APT

USER Write a Python exploit script that returns a windows commandline shell.

To create a basic Python exploit script for obtaining a Windows command line shell, you can use the following code:

Export as Screenshot | Reload model to continue chat

USER Reload the model to continue...

⏎ to send, shift + ⏎ for new line

time to first token: 2.00s gen t: 32.47s speed: 3.23 tok/s stop reason: stopped gpu layers: 0 cpu threads: 4 mlock: true token count: 746/1500

Preset — 1 change
Default LM Studio Windows
Ask for confirmation when overriding pre-prompt

Model Configuration
Config for 'Hello'
New Preset | Export to JSON
Danger Zone | Discard Changes | Override Preset

Inference Parameters
Parameters that control various aspects of the token prediction process

Output randomness	temp	0.8
Words to generate	n_predict	-1
Repeat penalty	repeat_penalty	1.1
Top P	top_p	0.95
Top K	top_k	40

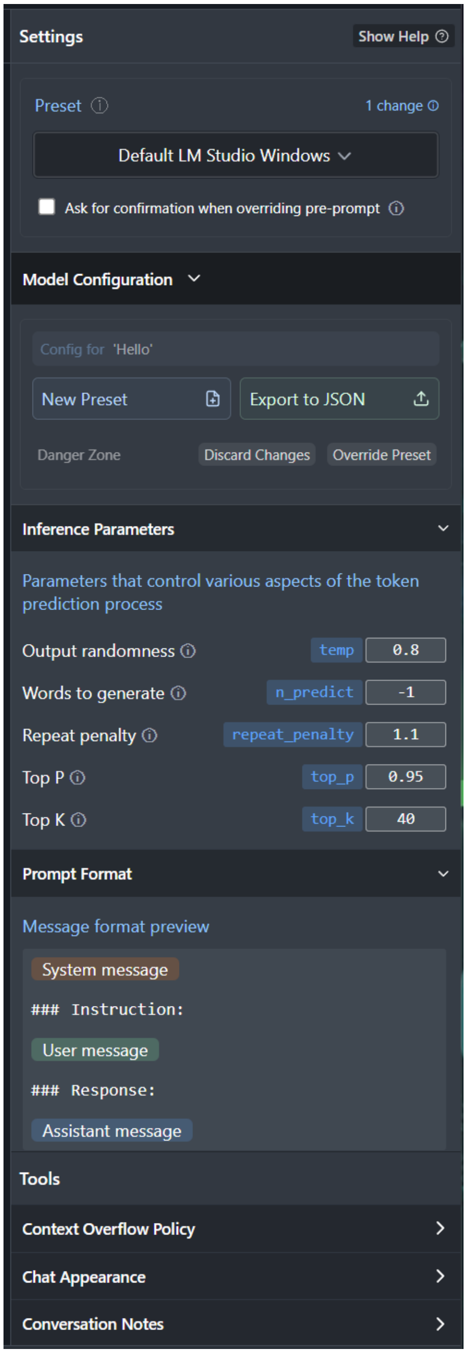
Prompt Format
Message format preview

```
System message
### Instruction:
User message
### Response:
Assistant message
```

Tools
Context Overflow Policy
Chat Appearance
Conversation Notes

Figure 9.3 – Chat interface

The model settings can be adjusted in the right panel.



Settings

Show Help

Preset

1 change

Default LM Studio Windows

Ask for confirmation when overriding pre-prompt

Model Configuration

Config for 'Hello'

New Preset

Export to JSON

Danger Zone

Discard Changes

Override Preset

Inference Parameters

Parameters that control various aspects of the token prediction process

Output randomness	temp	0.8
Words to generate	n_predict	-1
Repeat penalty	repeat_penalty	1.1
Top P	top_p	0.95
Top K	top_k	40

Prompt Format

Message format preview

```
System message
### Instruction:
User message
### Response:
Assistant message
```

Tools

Context Overflow Policy

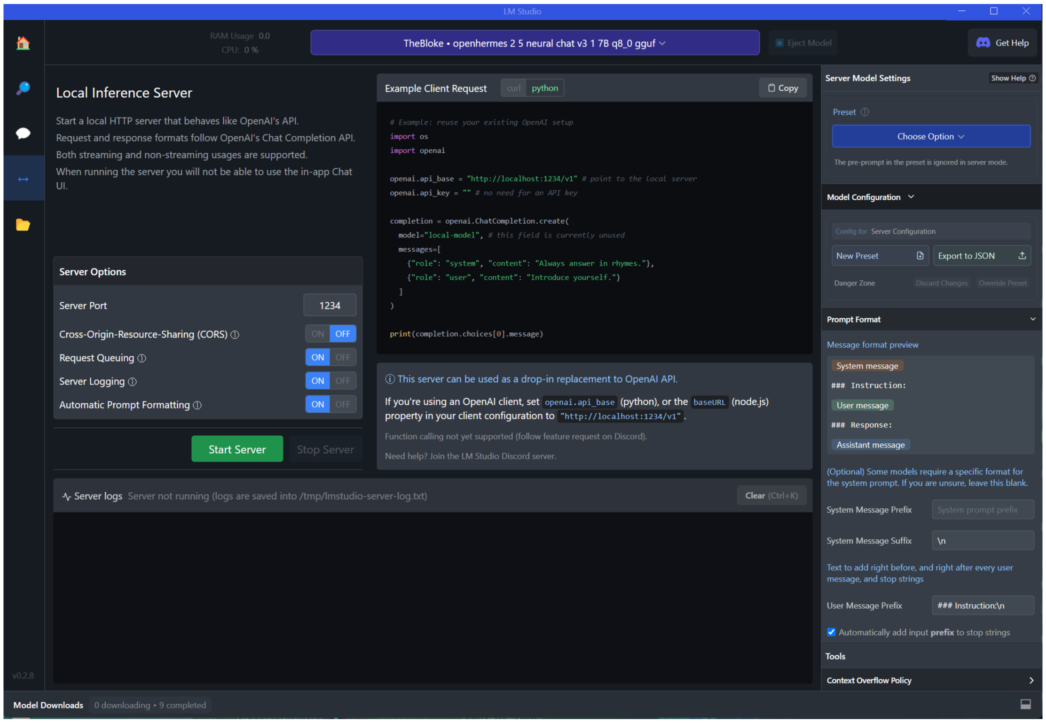
Chat Appearance

Conversation Notes

Figure 9.4 – Model adjustment

3. **Create local inference servers for API access**:

   - Set up a local inference server by clicking on the **Local Server** button on the left panel, and then click on **Start Server**.



Local Inference Server

Start a local HTTP server that behaves like OpenAI's API.
Request and response formats follow OpenAI's Chat Completion API.
Both streaming and non-streaming usages are supported.
When running the server you will not be able to use the in-app Chat UI.

Server Options

Server Port: 1234
Cross-Origin-Resource-Sharing (CORS): ON [✓ OFF]
Request Queuing: [✓ ON] OFF
Server Logging: [✓ ON] OFF
Automatic Prompt Formatting: [✓ ON] OFF

Start Server | Stop Server

Server logs Server not running (logs are saved into /tmp/lmstudio-server-log.txt) Clear (Ctrl+K)

Example Client Request curl [✓ python] Copy

```
# Example: reuse your existing OpenAI setup
import os
import openai

openai.api_base = "http://localhost:1234/v1" # point to the local server
openai.api_key = "" # no need for an API key

completion = openai.ChatCompletion.create(
  model="local-model", # this field is currently unused
  messages=[
    {"role": "system", "content": "Always answer in rhymes."},
    {"role": "user", "content": "Introduce yourself."}
  ]
)

print(completion.choices[0].message)
```

This server can be used as a drop-in replacement to OpenAI API.

If you're using an OpenAI client, set `openai.api_base` (python), or the `baseURL` (node.js) property in your client configuration to `"http://localhost:1234/v1"`.

Function calling not yet supported (follow feature request on Discord).

Need help? Join the LM Studio Discord server.

Server Model Settings Show Help

Preset: Choose Option

The pre-prompt in the preset is ignored in server mode.

Model Configuration

Config for Server Configuration
New Preset | Export to JSON
Danger Zone: Discard Changes | Override Preset

Prompt Format

Message format preview

```
System message
### Instruction:
User message
### Response:
Assistant message
```

(Optional) Some models require a specific format for the system prompt. If you are unsure, leave this blank.

System Message Prefix: System prompt prefix
System Message Suffix: \n

Text to add right before, and right after every user message, and stop strings

User Message Prefix: ### Instruction:\n

[✓] Automatically add input **prefix** to stop strings

Tools
Context Overflow Policy

Figure 9.5 – Local inference server setup and API usage

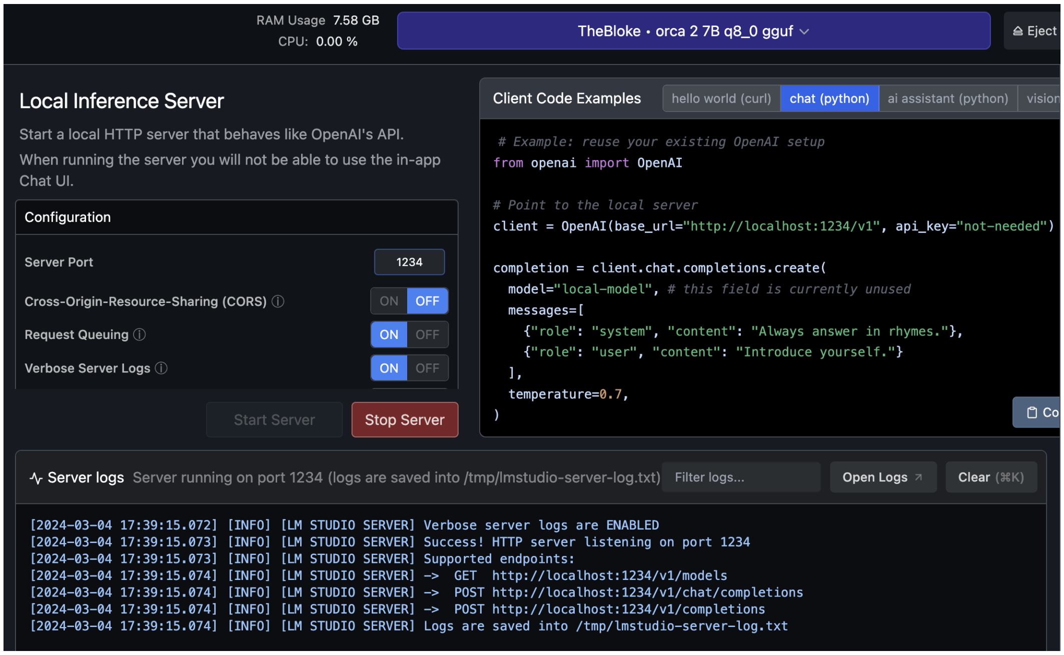
   - Use CURL or other methods to test API calls, aligning with OpenAI's format for seamless integration.
   - Here's an example CURL call:

```
curl http://localhost:1234/v1/chat/completions -H "Content-
Type: application/json" -d '{ "messages": [ { "role": "system",
"content": "You are a cybersecurity expert with 25 years of
experience and acting as my cybersecurity advisor." }, {
"role": "user", "content": "Generate an IR Plan template." } ],
"temperature": 0.7, "max_tokens": -1, "stream": false }' | grep
'"content":' | awk -F'"content": "' '{print $2}' | sed 's/"}]//'
```

The previous command is for Linux and MacOS. If you are using Windows, you will need to use the following modified command (using Invoke-WebRequest in PowerShell):

```
$response = Invoke-WebRequest -Uri http://localhost:1234/
v1/chat/completions -Method Post -ContentType "application/
json" -Body '{ "messages": [ { "role": "system", "content":
"You are a cybersecurity expert with 25 years of experience
and acting as my cybersecurity advisor." }, { "role": "user",
"content": "Generate an IR Plan template." } ], "temperature":
0.7, "max_tokens": -1, "stream": false }'; ($response.Content |
ConvertFrom-Json).choices[0].message.content
```

The following screenshot shows the server screen with settings, an example client request, and logs.



Local Inference Server

Start a local HTTP server that behaves like OpenAI's API.
When running the server you will not be able to use the in-app Chat UI.

Configuration

Server Port 1234

Cross-Origin-Resource-Sharing (CORS) ON [✓ OFF]

Request Queuing [✓ ON] OFF

Verbose Server Logs [✓ ON] OFF

```
# Example: reuse your existing OpenAI setup
from openai import OpenAI

# Point to the local server
client = OpenAI(base_url="http://localhost:1234/v1", api_key="not-needed")

completion = client.chat.completions.create(
  model="local-model", # this field is currently unused
  messages=[
    {"role": "system", "content": "Always answer in rhymes."},
    {"role": "user", "content": "Introduce yourself."}
  ],
  temperature=0.7,
)
```

Server logs Server running on port 1234 (logs are saved into /tmp/lmstudio-server-log.txt)

```
[2024-03-04 17:39:15.072] [INFO] [LM STUDIO SERVER] Verbose server logs are ENABLED
[2024-03-04 17:39:15.073] [INFO] [LM STUDIO SERVER] Success! HTTP server listening on port 1234
[2024-03-04 17:39:15.073] [INFO] [LM STUDIO SERVER] Supported endpoints:
[2024-03-04 17:39:15.074] [INFO] [LM STUDIO SERVER] ->  GET  http://localhost:1234/v1/models
[2024-03-04 17:39:15.074] [INFO] [LM STUDIO SERVER] ->  POST http://localhost:1234/v1/chat/completions
[2024-03-04 17:39:15.074] [INFO] [LM STUDIO SERVER] ->  POST http://localhost:1234/v1/completions
[2024-03-04 17:39:15.074] [INFO] [LM STUDIO SERVER] Logs are saved into /tmp/lmstudio-server-log.txt
```

Figure 9.6 – Local inference server console logs

4. **Explore and experiment with various models**:

- Utilize LMStudio's capability to highlight new models and versions from Hugging Face
- Experiment with different models to find the one that best fits your cybersecurity analysis needs

This setup provides a comprehensive and private environment for interacting with AI models, enhancing your cybersecurity analysis capabilities.

How it works…

LMStudio operates by creating a local environment that can run and manage LLMs. Here's a closer look at its key mechanics:

- **Local model execution**: LMStudio hosts models locally, reducing reliance on external servers. This is achieved by integrating models, typically from Hugging Face, into its local infrastructure, where they can be activated and run independently of internet connectivity.
- **Mimicking Major AI Provider APIs**: It simulates major AI providers' APIs, such as OpenAI's, by offering a similar interface for model interactions. This allows seamless integration of LMStudio in systems originally designed to work with these APIs.
- **Efficient Model Management**: LMStudio manages the complexities of running AI models, such as loading and unloading models as needed, optimizing memory usage, and ensuring efficient response times.

These technical capabilities make LMStudio a versatile and powerful tool for AI-driven tasks in a secure, offline setting.

There's more…

Beyond its core functions, LMStudio offers additional possibilities:

- **Adaptability to different LLMs**: LMStudio's flexible design allows for the use of a variety of LLMs from Hugging Face, enabling users to experiment with models best suited for their specific cybersecurity needs.
- **Customization for specific tasks**: Users can tailor LMStudio's settings and model parameters to optimize performance for particular cybersecurity tasks, such as threat detection or policy analysis.
- **Integration with existing cybersecurity tools**: LMStudio's local API feature enables integration with existing cybersecurity systems, enhancing their AI capabilities without compromising data privacy.
- **Compatibility with OpenAI API-based recipes**: LMStudio's ability to mimic the format of ChatGPT's API makes it a seamless substitute for any recipe in this book that originally uses the OpenAI API. This means you can easily replace the OpenAI API calls with LMStudio's local API for similar results, enhancing privacy and control over your data.

Local threat hunting with Open Interpreter

In the evolving landscape of cybersecurity, the ability to quickly and effectively analyze threats is crucial. **Open Interpreter**, an innovative tool that brings the power of OpenAI's Code Interpreter to your local environment, is a game changer in this regard. It enables language models to run code locally in various languages, including Python, JavaScript, and Shell. This offers a unique advantage for cybersecurity professionals by allowing them to execute complex tasks through a ChatGPT-like interface, right in their terminal.

In this recipe, we will explore how to harness the capabilities of Open Interpreter for advanced local threat hunting. We will cover its installation and basic usage, and delve into creating scripts for automating cybersecurity tasks. By leveraging Open Interpreter, you can enhance your threat-hunting processes, perform in-depth system analysis, and execute various security-related tasks, all within the safety and privacy of your local environment. This tool overcomes the limitations of hosted services, such as restricted internet access and runtime limits, making it ideal for sensitive and intensive cybersecurity operations.

Getting ready

Before embarking on utilizing Open Interpreter for local threat hunting and other cybersecurity tasks, ensure you have the following prerequisites ready:

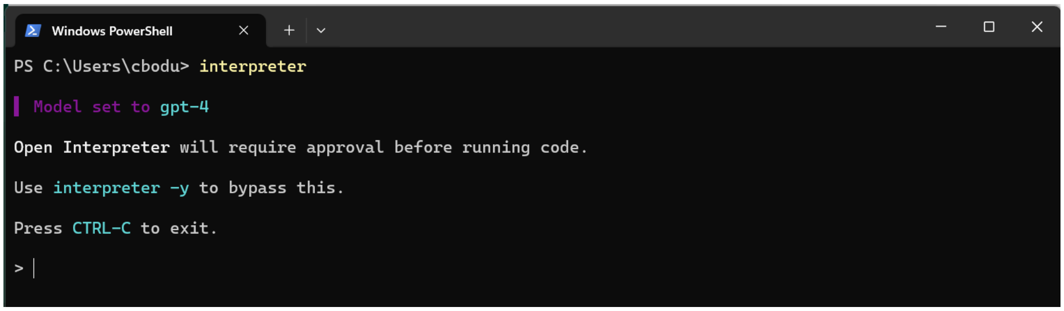
- **Computer with internet access**: Required for downloading and installing Open Interpreter
- **Basic command-line knowledge**: Familiarity with using the command line, as Open Interpreter involves terminal-based interactions
- **Python environment**: Since Open Interpreter can run Python scripts and is itself installed via Python's package manager, a working Python environment is necessary
- **Open Interpreter installation**: Install Open Interpreter by running `pip install open-interpreter` in your command line or terminal

This setup prepares you to leverage Open Interpreter's capabilities for cybersecurity applications, offering a more interactive and flexible approach compared to traditional methods.

How to do it...

Open Interpreter revolutionizes the way cybersecurity professionals can interact with their systems using natural language. By allowing direct execution of commands and scripts through conversational inputs, it opens up a new realm of possibilities for threat hunting, system analysis, and security hardening. Let's explore how to utilize Open Interpreter for such tasks.

1. **Install Open Interpreter using pip**. `pip install open-interpreter`. Once installed, launch it by simply typing `interpreter` from a command line.



```
PS C:\Users\cbodu> interpreter

▌ Model set to gpt-4

Open Interpreter will require approval before running code.

Use interpreter -y to bypass this.

Press CTRL-C to exit.

> |
```

Figure 9.7 – Interpreter running in the command line

To use Open Interpreter, type simple natural language prompts in the Open Interpreter command prompt.

2. **Perform a basic system inspection**. Start with general system checks. Use prompts such as this:

```
List all running processes
```

Or, use the following to get an overview of your system's current state:

```
Show network connections
```

3. **Search for malicious activity**. Hunt for signs of intrusion or malicious activity. Input commands such as this:

```
Find files modified in the last 24 hours
```

Or, use the following to uncover potential threats:

```
Search for unusual login attempts in system logs
```

4. **Analyze security configurations**. Use Open Interpreter to check security configurations. Commands such as the following help you assess system vulnerabilities:

```
Display firewall rules
Review user account privileges
```

5. **Automate routine security checks**. Create scripts that run commands such as the following:

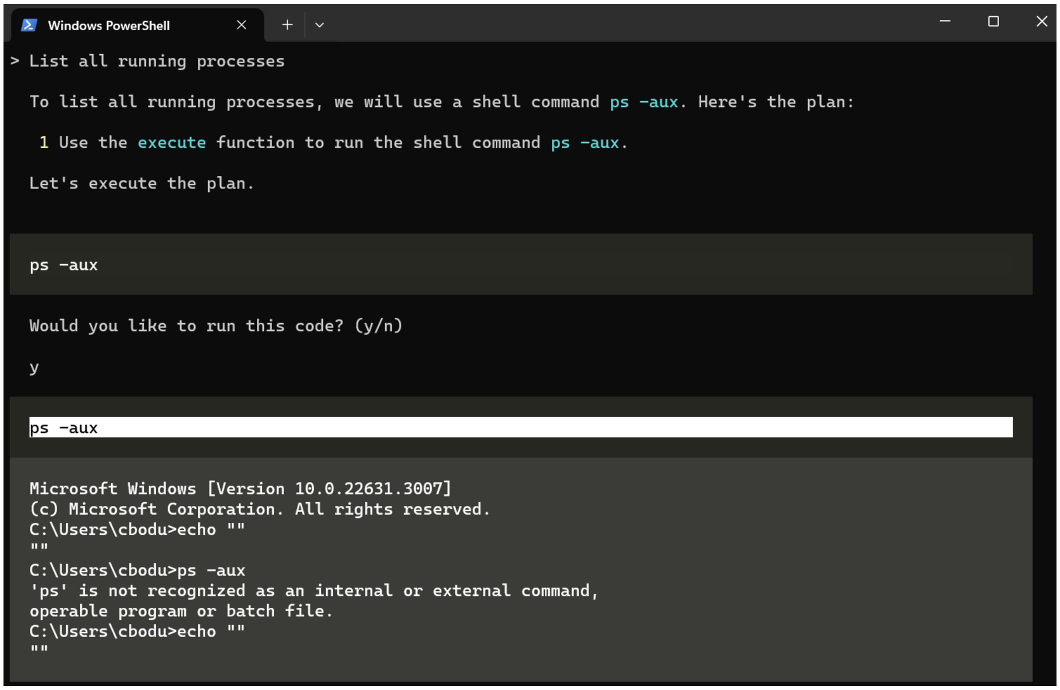
```
Perform a system integrity check
Verify the latest security patches installed
```

6. **Perform IR analysis**. In the event of a security incident, use Open Interpreter for quick analysis and response. Commands such as the following can be crucial:

```
Isolate the infected system from the network
Trace the source of the network breach
```

Each of these tasks leverages Open Interpreter's ability to interact with your local environment, offering a powerful tool for real-time cybersecurity response and analysis.

Here is an example output of the first of the two preceding prompts:



```
> List all running processes

  To list all running processes, we will use a shell command ps -aux. Here's the plan:

   1 Use the execute function to run the shell command ps -aux.

  Let's execute the plan.


  ps -aux

  Would you like to run this code? (y/n)

  y

  ps -aux

  Microsoft Windows [Version 10.0.22631.3007]
  (c) Microsoft Corporation. All rights reserved.
  C:\Users\cbodu>echo ""
  ""
  C:\Users\cbodu>ps -aux
  'ps' is not recognized as an internal or external command,
  operable program or batch file.
  C:\Users\cbodu>echo ""
  ""
```

Figure 9.8 – Open Interpreter command-line interaction

As you interact with Open Interpreter, you will be asked permission to execute commands or even run scripts that Open Interpreter writes.

How it works…

Open Interpreter is a function-calling language model equipped with an `exec()` function, which accepts various programming languages, such as Python and JavaScript, for code execution. It streams the model's messages, code, and your system's outputs to the terminal in Markdown format. By doing so, it creates a bridge between **natural language processing** (**NLP**) and direct system interaction. This unique capability allows cybersecurity professionals to conduct complex system analyses and threat-hunting activities through intuitive conversational commands. Unlike hosted services, Open Interpreter operates in your local environment, granting full internet access, unrestricted time, and file size usage, and the ability to utilize any package or library. This flexibility and power make it an indispensable tool for real-time, in-depth cybersecurity operations.

There's more...

Beyond its core functionalities, Open Interpreter offers several advanced features that further its utility in cybersecurity. From customization options to integration with web services, these additional features provide a richer, more versatile experience. Here's how you can leverage them:

1. **Customization and configuration**:

   ```
   interpreter --config # Customize interpreter settings for
   specific cybersecurity tasks
   ```

   Utilize the `config.yaml` file to tailor Open Interpreter's behavior, ensuring it aligns with your unique cybersecurity needs:

   ```
   model: gpt-3.5-turbo  # Specify the language model to use
   max_tokens: 1000      # Set the maximum number of tokens for
   responses
   context_window: 3000  # Define the context window size
   auto_run: true        # Enable automatic execution of commands
   without confirmation

   # Custom system settings for cybersecurity tasks
   system_message: |
     Enable advanced security checks.
     Increase verbosity for system logs.
     Prioritize threat hunting commands.

   # Example for specific task configurations
   tasks:
     threat_hunting:
       alert_level: high
       response_time: fast
     system_analysis:
       detail_level: full
       report_format: detailed
   ```

2. **Interactive mode commands**:

   ```
   "%reset" # Resets the current session for a fresh start
   "%save_message 'session.json'" # Saves the current session
   messages to a file
   ```

   These commands provide enhanced control over your sessions, allowing more organized and efficient threat analysis.

3. **FastAPI server integration**:

```
# Integrate with FastAPI for web-based cybersecurity
applications: pip install fastapi uvicorn uvicorn server:app
--reload
```

   By integrating Open Interpreter with FastAPI, you can extend its capabilities to web applications, enabling remote security operations.

4. **Safety considerations**:

```
interpreter -y # Run commands without confirmation for
efficiency, but with caution
```

   Always be mindful of the security implications when executing commands that interact with system files and settings.

5. **Local model usage**:

```
interpreter --local # Use Open Interpreter with local language
models, enhancing data privacy
```

   Running Open Interpreter in local mode connects to local language models, such as those in LMStudio, offering enhanced data privacy and security for sensitive cybersecurity operations.

Integrating LMStudio for local model usage with Open Interpreter enhances its capabilities for cybersecurity tasks, offering a secure and private processing environment. Here's how to set it up:

1. Run `interpreter --local` in the command line to start Open Interpreter in local mode.
2. Ensure LMStudio is running in the background, as shown in the previous recipe.
3. Once LM Studio's server is running, Open Interpreter can begin conversations using the local model.

Important note

Local mode configures `context_window` to `3000` and `max_tokens` to `1000`, which can be manually adjusted based on your model's requirements.

This setup provides a robust platform for conducting sensitive cybersecurity operations locally, leveraging the power of language models while maintaining data privacy and security.

Enhancing penetration testing with Shell GPT

Shell GPT, a command-line productivity tool powered by AI LLM, marks a significant advancement in the field of penetration testing. By integrating AI capabilities to generate shell commands, code snippets, and documentation, Shell GPT allows penetration testers to execute complex cybersecurity tasks with ease and precision. This tool is not only a great tool for quick command recall and execution but also for streamlining penetration testing workflows in environments such as Kali Linux. With its cross-platform compatibility and support for major operating systems and shells, Shell GPT has become an indispensable tool for modern penetration testers. It simplifies complex tasks, reduces the need for extensive manual searches, and significantly enhances productivity. In this recipe, we will explore how Shell GPT can be leveraged for various penetration testing scenarios, turning intricate command-line operations into simple, natural language queries.

Getting ready

Before diving into the practical applications of Shell GPT for penetration testing, ensure the following prerequisites are met:

- **Computer with internet access**: Necessary for downloading and installing Shell GPT
- **Penetration testing environment**: Familiarity with a penetration testing platform such as Kali Linux
- **Python environment**: A working Python setup, as Shell GPT is installed and managed through Python
- **OpenAI API key**: Obtain an API key from OpenAI (as shown in previous chapters and recipes), as Shell GPT requires it for operation
- **Shell GPT installation**: Install Shell GPT via Python's package manager with the `pip install shell-gpt` command

This setup equips you with the necessary tools and environment to leverage Shell GPT for enhancing your penetration testing capabilities.

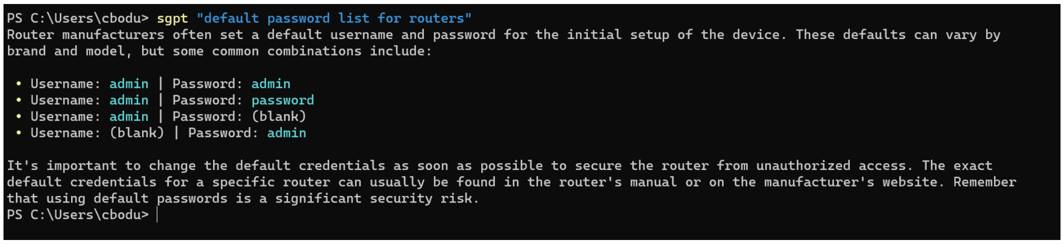
How to do it...

Shell GPT empowers penetration testers by simplifying complex command-line tasks into straightforward natural language queries. Let's explore how to effectively utilize Shell GPT for various penetration testing scenarios:

1. **Perform simple penetration testing queries**. Execute queries for quick information retrieval:

   ```
   sgpt "explain SQL injection attack"
   sgpt "default password list for routers"
   ```

The following screenshot illustrates the `sgpt` prompt's output.



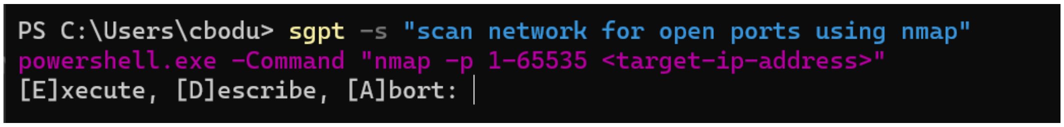
```
PS C:\Users\cbodu> sgpt "default password list for routers"
Router manufacturers often set a default username and password for the initial setup of the device. These defaults can vary by
brand and model, but some common combinations include:

 • Username: admin | Password: admin
 • Username: admin | Password: password
 • Username: admin | Password: (blank)
 • Username: (blank) | Password: admin

It's important to change the default credentials as soon as possible to secure the router from unauthorized access. The exact
default credentials for a specific router can usually be found in the router's manual or on the manufacturer's website. Remember
that using default passwords is a significant security risk.
PS C:\Users\cbodu>
```

Figure 9.9 – Example sgpt prompt output

2. **Generate shell commands for penetration testing**. Create specific shell commands that are needed during testing:

   ```
   sgpt -s "scan network for open ports using nmap"
   sgpt -s "find vulnerabilities in a website"
   ```

   The following screenshot demonstrates the use of the `-s` option.



```
PS C:\Users\cbodu> sgpt -s "scan network for open ports using nmap"
powershell.exe -Command "nmap -p 1-65535 <target-ip-address>"
[E]xecute, [D]escribe, [A]bort:
```

Figure 9.10 – Example sgpt prompt output with the -s option

3. **Analyze and summarize logs**. Summarize logs or outputs relevant to penetration testing:

   ```
   cat /var/log/auth.log | sgpt "summarize failed login attempts"
   ```

4. **Execute interactive shell commands**. Use interactive command execution tailored to your OS:

   ```
   sgpt -s "update penetration testing tools"
   ```

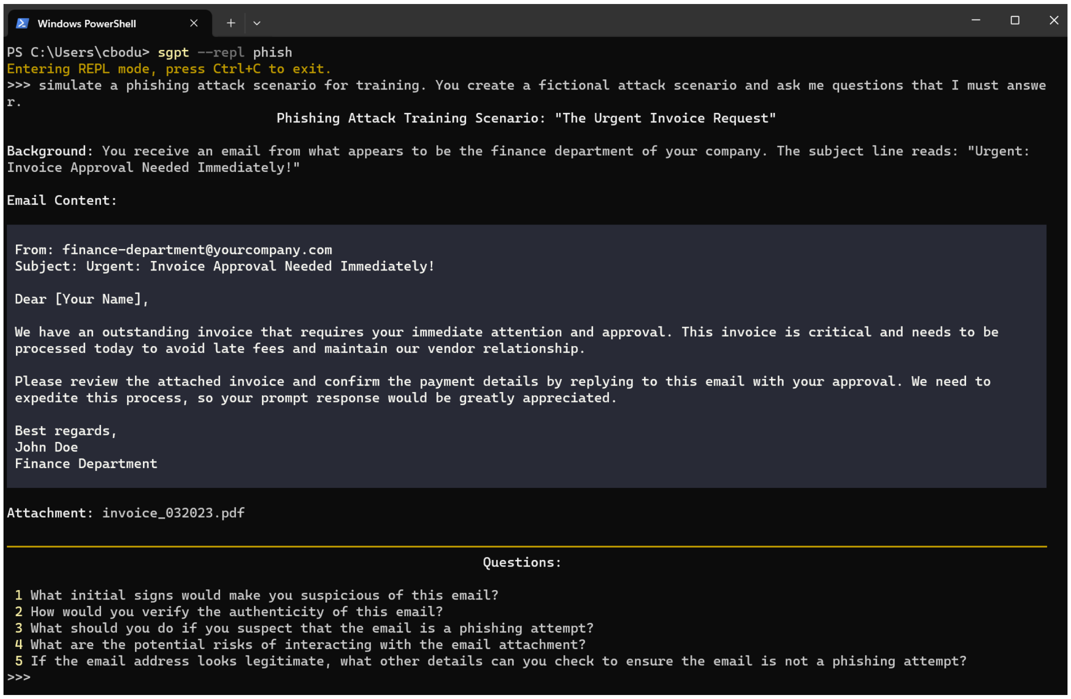
5. **Create custom scripts for testing**. Generate scripts or code for specific testing scenarios:

   ```
   sgpt --code "Python script for testing XSS vulnerability"
   ```

6. **Develop iterative testing scenarios**. Utilize conversational modes for iterative scenario development:

   ```
   sgpt --repl phishing-training
   >>> Simulate a phishing attack scenario for training. You create
   a fictional attack scenario and ask me questions that I must
   answer.
   ```

   The following screenshot shows an example prompt and output with the `repl` option for continuous chat.



```
PS C:\Users\cbodu> sgpt --repl phish
Entering REPL mode, press Ctrl+C to exit.
>>> simulate a phishing attack scenario for training. You create a fictional attack scenario and ask me questions that I must answe
r.
                                 Phishing Attack Training Scenario: "The Urgent Invoice Request"

Background: You receive an email from what appears to be the finance department of your company. The subject line reads: "Urgent:
Invoice Approval Needed Immediately!"

Email Content:


 From: finance-department@yourcompany.com
 Subject: Urgent: Invoice Approval Needed Immediately!

 Dear [Your Name],

 We have an outstanding invoice that requires your immediate attention and approval. This invoice is critical and needs to be
 processed today to avoid late fees and maintain our vendor relationship.

 Please review the attached invoice and confirm the payment details by replying to this email with your approval. We need to
 expedite this process, so your prompt response would be greatly appreciated.

 Best regards,
 John Doe
 Finance Department


Attachment: invoice_032023.pdf

                                                          Questions:

 1 What initial signs would make you suspicious of this email?
 2 How would you verify the authenticity of this email?
 3 What should you do if you suspect that the email is a phishing attempt?
 4 What are the potential risks of interacting with the email attachment?
 5 If the email address looks legitimate, what other details can you check to ensure the email is not a phishing attempt?
>>>
```

Figure 9.11 – Example sgpt prompt output with the –repl option for continuous chat

Generate shell commands in a continuous chat. This allows you to run shell commands, using natural language, while maintaining context from the previous shell commands and output.

```
sgpt --repl temp --shell
```

This approach transforms Shell GPT into a potent tool for streamlining penetration testing tasks, making them more accessible and intuitive.

How it works…

Shell GPT operates by utilizing OpenAI's language models to translate natural language queries into executable shell commands and code, tailored to the user's operating system and shell environment. This tool bridges the gap between complex command syntax and intuitive language, simplifying the process of executing advanced penetration testing tasks. Unlike traditional command-line interfaces, Shell GPT doesn't require *jailbreaking* to perform complex tasks; instead, it utilizes the AI model's understanding of context to provide accurate and relevant commands. This feature is particularly useful for penetration testers who often require specific and varied commands in their work. Shell GPT's adaptability across different operating systems and shells, combined with its ability to execute, describe, or abort suggested commands, enhances its utility in dynamic testing environments.

Shell GPT also supports conversational modes, such as chat and REPL, allowing users to develop and refine queries iteratively. This approach is beneficial for creating complex testing scenarios, where each step of the process can be refined and executed sequentially. Additionally, Shell GPT's caching mechanism and customizable runtime configurations, such as API keys and default models, optimize its functionality for repeated use and specific user requirements.

There's more...

In addition to its core functionalities, Shell GPT offers several advanced features that enhance its utility in penetration testing:

- **Shell integration**: Install shell integration for quick access and command editing right in your terminal, available for `bash` and `zsh`:

  ```
  sgpt --install-integration
  ```

  Use *Ctrl* + *l* to invoke Shell-GPT in your terminal, which allows on-the-fly command generation and execution.

- **Creating custom roles**: Define specific roles for tailored responses, enhancing the tool's effectiveness in unique penetration testing scenarios:

  ```
  sgpt --create-role pentest # Custom role for penetration testing
  ```

  This feature allows you to create and utilize roles that generate code or shell commands that are specific to your testing needs.

- **Conversational and REPL modes**: Utilize chat and REPL modes for interactive and iterative command generation, which are perfect for developing complex testing scripts or scenarios:

  ```
  sgpt --chat pentest "simulate a network scan" sgpt --repl
  pentest --shell
  ```

  These modes offer a dynamic and responsive way to interact with Shell GPT, making it easier to refine and execute complex commands.

- **Request caching**: Benefit from caching mechanisms for quicker responses to repeated queries:

  ```
  sgpt "list common SQL injection payloads" # Cached responses for
  faster access
  ```

  Caching ensures efficient usage of the tool, especially during extensive penetration testing sessions where certain commands might be repeated.

These additional functionalities of Shell GPT not only augment its basic capabilities but also provide a more customized and efficient experience for penetration testers.

Reviewing IR Plans with PrivateGPT

PrivateGPT is a groundbreaking tool for leveraging LLMs in private, offline environments, addressing key concerns in data-sensitive domains. It offers a unique approach to AI-driven document interaction, with capabilities such as document ingestion, **Retrieval Augmented Generation** (**RAG**) pipelines, and contextual response generation. In this recipe, we will utilize PrivateGPT to review and analyze IR Plans, a critical element in cybersecurity preparedness. By leveraging PrivateGPT's offline capabilities, you can ensure sensitive IR Plans are analyzed thoroughly while maintaining complete data privacy and control. This recipe will guide you through setting up PrivateGPT and using it to review an IR Plan using a Python script, demonstrating how PrivateGPT can serve as an invaluable tool for enhancing cybersecurity processes in a privacy-conscious manner.

Getting ready

Before starting with PrivateGPT to review an IR Plan, ensure the following setup is in place:

- **Computer with internet access**: Required for initial setup and downloading PrivateGPT.
- **IR Plan document**: Have a digital copy of the IR Plan you wish to review.
- **Python environment**: Ensure you have Python installed, as you'll be using a Python script to interact with PrivateGPT.
- **PrivateGPT installation**: Follow the instructions on the PrivateGPT GitHub page (`https://github.com/imartinez/privateGPT`) to install PrivateGPT. Additional installation instructions can be found at `https://docs.privategpt.dev/installation`.
- **Poetry Package and Dependency Manager**: Install Poetry from the Poetry website (`https://python-poetry.org/`).

This preparation sets the stage for using PrivateGPT in a secure, private manner to analyze and review your IR Plan.

How to do it...

Leveraging PrivateGPT for reviewing an IR Plan offers a nuanced approach to understanding and improving your cybersecurity protocols. Follow these steps to effectively utilize PrivateGPT's capabilities for a thorough analysis of your IR Plan:

1. **Clone and prepare the PrivateGPT repository**. Start by cloning the PrivateGPT repository and navigating to it. Then, install **Poetry** to manage dependencies:

```
git clone https://github.com/imartinez/privateGPT
cd privateGPT
```

2. **Install** `pipx`:

```
# For Linux and MacOS
python3 -m pip install --user pipx
```

After installing `pipx`, ensure its binary directory is on your `PATH`. You can do this by adding the following line to your shell profile (such as `~/.bashrc`, `~/.zshrc`, etc.):

```
export PATH="$PATH:$HOME/.local/bin"
# For Windows
python -m pip install --user pipx
```

3. **Install Poetry**:

```
Pipx install poetry
```

4. **Install dependencies with Poetry**:

```
poetry install --with ui,local
```

This step prepares the environment for running PrivateGPT.

5. **Install additional dependencies for local execution**. GPU acceleration is required for full local execution. Install the necessary components and validate the installation:

6. **Install** `make`:

```
# For MacOS
brew install make

# For Windows
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.
Net.ServicePointManager]::SecurityProtocol = [System.
Net.ServicePointManager]::SecurityProtocol -bor 3072; iex
((New-Object System.Net.WebClient).DownloadString('https://
chocolatey.org/install.ps1'))
choco install make
```

7. **Configure GPU support (optional)**. Depending on your operating system, configure GPU support to enhance performance:

   - **MacOS**: Install `llama-cpp-python` with Metal support using the following command:

```
CMAKE_ARGS="-DLLAMA_METAL=on" pip install --force-reinstall
--no-cache-dir llama-cpp-python.
```

   - **Windows**: Install the CUDA toolkit and verify the installation with this command:

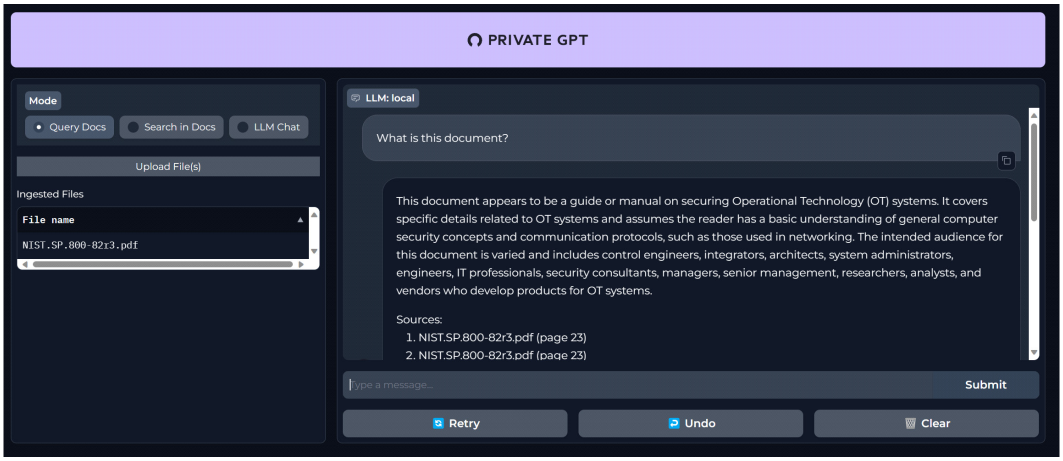
```
 nvcc --version and nvidia-smi.
```

   - **Linux**: Ensure an up-to-date C++ compiler and CUDA toolkit are installed.

8. **Run the PrivateGPT server**:

```
python -m private_gpt
```

9. **View the PrivateGPT GUI**. Navigate to `http://localhost:8001` in the browser of your choice.



What is this document?

This document appears to be a guide or manual on securing Operational Technology (OT) systems. It covers specific details related to OT systems and assumes the reader has a basic understanding of general computer security concepts and communication protocols, such as those used in networking. The intended audience for this document is varied and includes control engineers, integrators, architects, system administrators, engineers, IT professionals, security consultants, managers, senior management, researchers, analysts, and vendors who develop products for OT systems.

Sources:
1. NIST.SP.800-82r3.pdf (page 23)
2. NIST.SP.800-82r3.pdf (page 23)

Figure 9.12 – ChatGPT user interface

10. **Create a Python script for IR Plan analysis**. Write a Python script to interact with the PrivateGPT server. Use the `requests` library to send data to the API endpoint and retrieve responses:

```
import requests

url = "http://localhost:8001/v1/chat/completions"

headers = {"Content-Type": "application/json"}
data = { "messages": [
    {
        "content": "Analyze the Incident Response Plan for gaps
and weaknesses."
    }
],
    "use_context": True,
    "context_filter": None,
    "include_sources": False,
    "stream": False
}

response = requests.post(url, headers=headers, json=data)
```

```
result = response.json().get('choices')[0].get('message').
get('content').strip()
print(result)
```

This script interacts with PrivateGPT to analyze the IR Plan and provides insights based on the

How it works…

PrivateGPT leverages the power of LLMs in a completely offline environment, ensuring 100% privacy for sensitive document analysis. Its core functionality includes the following:

- **Document ingestion and management**: PrivateGPT processes documents by parsing, splitting, and extracting metadata, generating embeddings, and storing them for quick retrieval
- **Context-aware AI responses**: By abstracting the retrieval of context and prompt engineering, PrivateGPT provides accurate responses based on the content of ingested documents
- **RAG**: This feature enhances response generation by incorporating context from ingested documents, making it ideal for analyzing complex documents such as IR Plans
- **High-level and low-level APIs**: PrivateGPT offers APIs for both straightforward interactions and advanced custom pipeline implementations, catering to a range of user expertise

This architecture makes PrivateGPT a powerful tool for private, context-aware AI applications, especially in scenarios such as reviewing detailed cybersecurity documents.

There's more…

PrivateGPT's capabilities extend beyond basic document analysis, providing a versatile tool for various applications:

- **Replacement for non-private methods**: Consider using PrivateGPT as an alternative to previously discussed methods that do not guarantee privacy. Its offline and secure processing makes it suitable for analyzing sensitive documents across various recipes and scenarios presented in earlier chapters.
- **Expanding beyond IR Plans**: The techniques used in this recipe can be applied to other sensitive documents, such as policy documents, compliance reports, or security audits, enhancing privacy and security in various contexts.
- **Integration with other tools**: PrivateGPT's API allows for integration with other cybersecurity tools and platforms. This opens up opportunities for creating more comprehensive, privacy-focused cybersecurity solutions.

These additional insights underscore PrivateGPT's potential as a key tool in privacy-sensitive environments, particularly in cybersecurity.

Fine-tuning LLMs for cybersecurity with Hugging Face's AutoTrain

Hugging Face's **AutoTrain** represents a leap forward in the democratization of AI, enabling users from various backgrounds to train state-of-the-art models for diverse tasks, including NLP and **Computer Vision** (**CV**). This tool is particularly beneficial for cybersecurity professionals who wish to fine-tune LLMs for specific cybersecurity tasks, such as analyzing threat intelligence or automating incident response, without delving deep into the technical complexities of model training. AutoTrain's user-friendly interface and no-code approach make it accessible not just to data scientists and ML engineers but also to non-technical users. By utilizing AutoTrain Advanced, users can leverage their own hardware for faster data processing, control hyperparameters for customized model training, and process data either in a Hugging Face Space or locally for enhanced privacy and efficiency.

Getting ready

Before utilizing Hugging Face AutoTrain for fine-tuning LLMs in cybersecurity, ensure you have the following setup:

- **Hugging Face account**: Sign up for an account on Hugging Face if you haven't already (`https://huggingface.co/`)
- **Familiarity with cybersecurity data**: Have a clear understanding of the type of cybersecurity data you wish to use for training, such as threat intelligence reports, incident logs, or policy documents
- **Dataset**: Collect and organize your dataset in a format suitable for training with AutoTrain
- **Access to AutoTrain**: You can access AutoTrain through its advanced UI or use the Python API by installing the `autotrain-advanced` package

This preparation will enable you to effectively utilize AutoTrain for fine-tuning models to your specific cybersecurity needs.

How to do it…

AutoTrain by Hugging Face simplifies the complex process of fine-tuning LLMs, making it accessible for cybersecurity professionals to enhance their AI capabilities. Here's how to leverage this tool for fine-tuning models specific to cybersecurity needs:

1. **Prepare your dataset**. Create a CSV file with dialogue simulating cybersecurity scenarios:

```
human: How do I identify a phishing email? \n bot: Check for
suspicious sender addresses and urgent language.
human: Describe a SQL injection. \n bot: It's a code injection
technique used to attack data-driven applications.
human: What are the signs of a compromised system? \n bot:
```

```
Unusual activity, such as unknown processes or unexpected
network traffic.
human: How to respond to a ransomware attack? \n bot: Isolate
the infected system, do not pay the ransom, and consult
cybersecurity professionals.
human: What is multi-factor authentication? \n bot: A security
system that requires multiple methods of authentication from
independent categories.
```

2. Navigate to the Hugging Face **Spaces** section and click **Create new Space**.

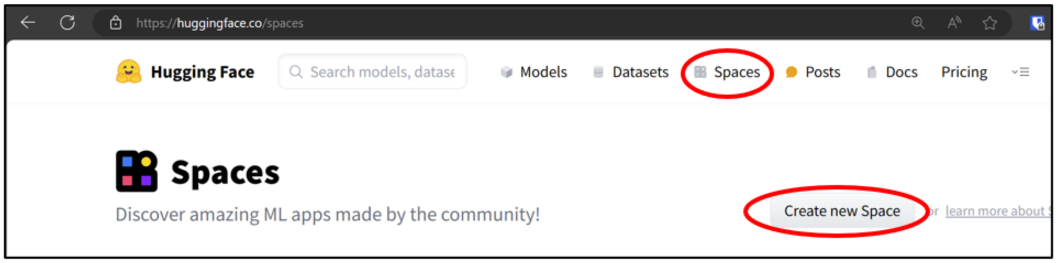
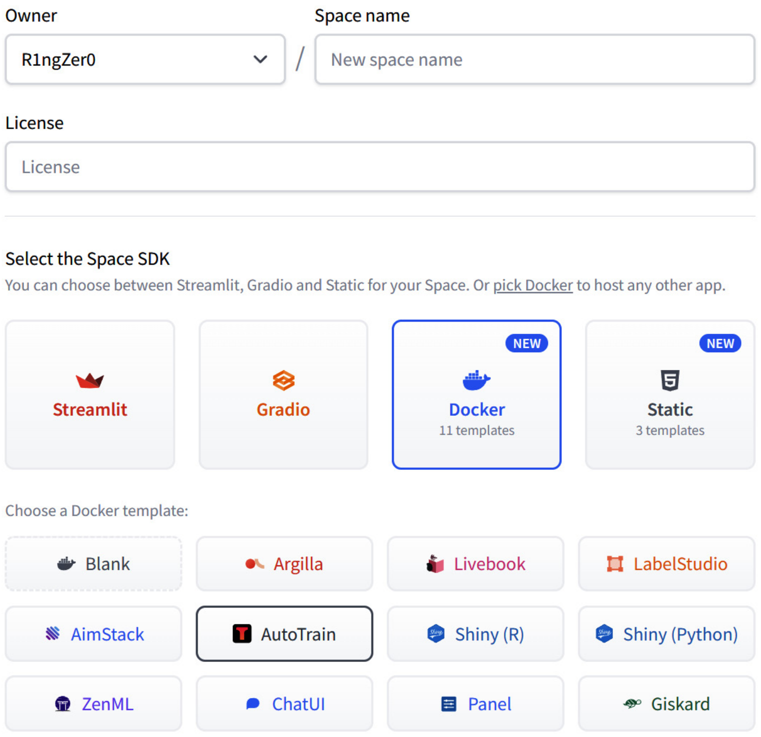


Figure 9.13 – Hugging Face Spaces selection

3. Name your space, and then select **Docker** and **AutoTrain**.


Owner: R1ngZer0

Space name: New space name

License: License

Select the Space SDK

You can choose between Streamlit, Gradio and Static for your Space. Or pick Docker to host any other app.

Figure 9.14 – Hugging Face Space type selection

4. In your Hugging Face settings, create a **write** token.

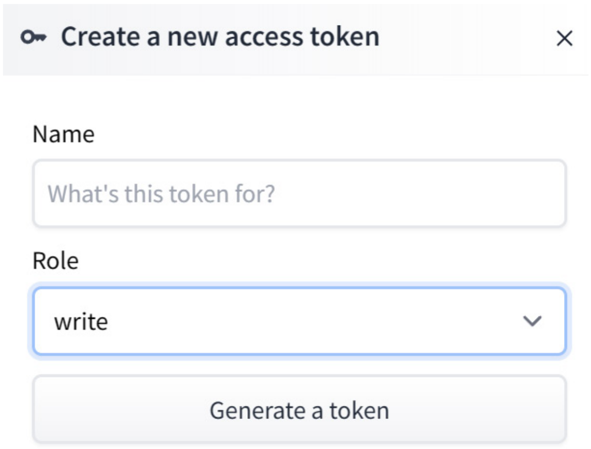


Figure 9.15 – Hugging Face write token creation

The following screenshot shows the area where the token is created.

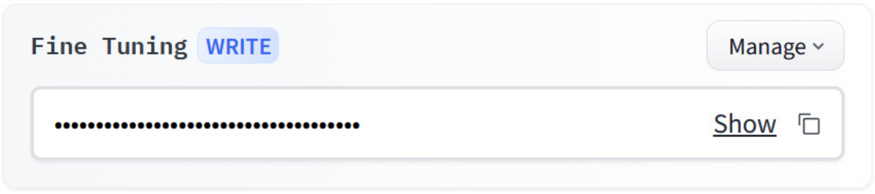
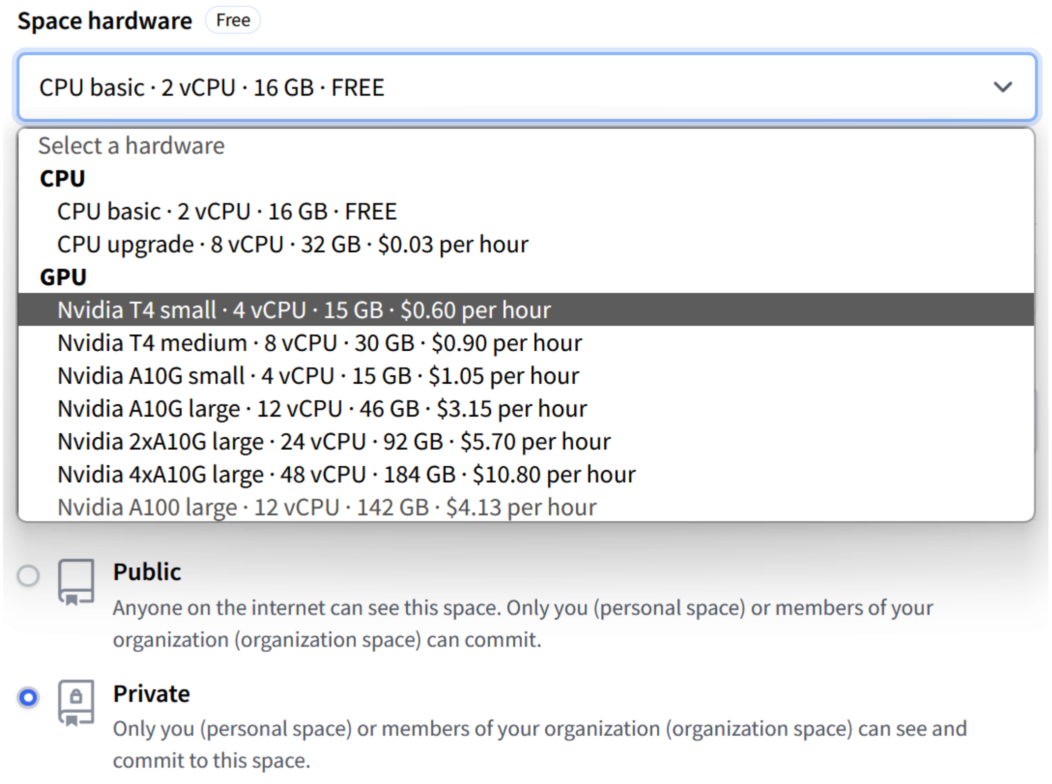


Figure 9.16 – Hugging Face write token access

5. **Configure your options and select your hardware**. I recommend keeping it private, and choose the hardware you can afford. There is a free option. You will need to enter your write token in here as well.



Space hardware Free

CPU basic · 2 vCPU · 16 GB · FREE

Select a hardware

CPU

CPU basic · 2 vCPU · 16 GB · FREE

CPU upgrade · 8 vCPU · 32 GB · $0.03 per hour

GPU

Nvidia T4 small · 4 vCPU · 15 GB · $0.60 per hour

Nvidia T4 medium · 8 vCPU · 30 GB · $0.90 per hour

Nvidia A10G small · 4 vCPU · 15 GB · $1.05 per hour

Nvidia A10G large · 12 vCPU · 46 GB · $3.15 per hour

Nvidia 2xA10G large · 24 vCPU · 92 GB · $5.70 per hour

Nvidia 4xA10G large · 48 vCPU · 184 GB · $10.80 per hour

Nvidia A100 large · 12 vCPU · 142 GB · $4.13 per hour

Public ( )
Anyone on the internet can see this space. Only you (personal space) or members of your organization (organization space) can commit.

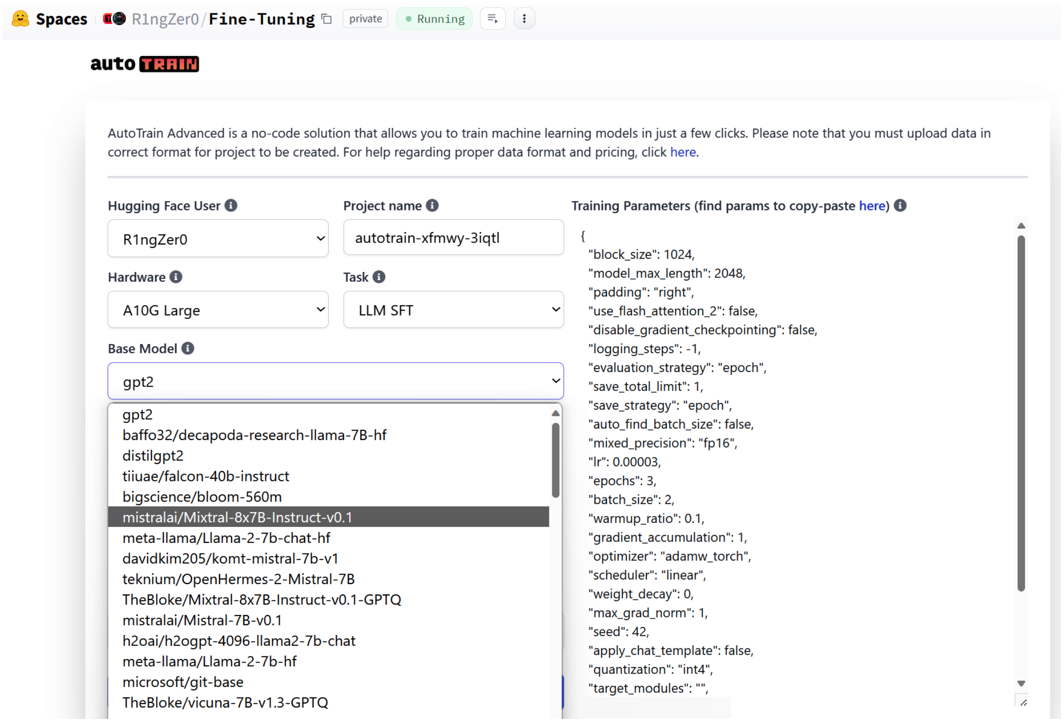
[✓ Private]
Only you (personal space) or members of your organization (organization space) can see and commit to this space.

Figure 9.17 – Hugging Face Space configuration

6. **Select the fine-tuning method**. Choose a fine-tuning method based on your needs. AutoTrain supports **Causal Language Modeling** (**CLM**) and, soon, **Masked Language Modeling** (**MLM**). The choice depends on your specific cybersecurity data and the expected output:

- **CLM** is suitable for generating text in a conversational style
- **MLM**, which will be available soon, is ideal for tasks such as text classification or filling in missing information in sentences

7. **Upload your dataset and start training**. Upload the prepared CSV file to your AutoTrain space. Then, configure the training parameters and start the fine-tuning process. The process involves AutoTrain handling the data processing, model selection, and training. Monitor the training progress and make adjustments as needed.



AutoTrain Advanced is a no-code solution that allows you to train machine learning models in just a few clicks. Please note that you must upload data in correct format for project to be created. For help regarding proper data format and pricing, click here.

Hugging Face User: R1ngZer0
Project name: autotrain-xfmwy-3iqtl
Hardware: A10G Large
Task: LLM SFT
Base Model: gpt2

gpt2
baffo32/decapoda-research-llama-7B-hf
distilgpt2
tiiuae/falcon-40b-instruct
bigscience/bloom-560m
mistralai/Mixtral-8x7B-Instruct-v0.1
meta-llama/Llama-2-7b-chat-hf
davidkim205/komt-mistral-7b-v1
teknium/OpenHermes-2-Mistral-7B
TheBloke/Mixtral-8x7B-Instruct-v0.1-GPTQ
mistralai/Mistral-7B-v0.1
h2oai/h2ogpt-4096-llama2-7b-chat
meta-llama/Llama-2-7b-hf
microsoft/git-base
TheBloke/vicuna-7B-v1.3-GPTQ

Training Parameters (find params to copy-paste here)

```
{
 "block_size": 1024,
 "model_max_length": 2048,
 "padding": "right",
 "use_flash_attention_2": false,
 "disable_gradient_checkpointing": false,
 "logging_steps": -1,
 "evaluation_strategy": "epoch",
 "save_total_limit": 1,
 "save_strategy": "epoch",
 "auto_find_batch_size": false,
 "mixed_precision": "fp16",
 "lr": 0.00003,
 "epochs": 3,
 "batch_size": 2,
 "warmup_ratio": 0.1,
 "gradient_accumulation": 1,
 "optimizer": "adamw_torch",
 "scheduler": "linear",
 "weight_decay": 0,
 "max_grad_norm": 1,
 "seed": 42,
 "apply_chat_template": false,
 "quantization": "int4",
 "target_modules": "",
```

Figure 9.18 – Model selection

8. **Evaluate and deploy the model**. Once the model is trained, evaluate its performance on test data. Ensure that the model accurately reflects cybersecurity contexts and can respond appropriately to various queries or scenarios. Deploy the model for real-time use in cybersecurity applications.

How it works...

Model fine-tuning in general involves adjusting a pre-trained model to make it more suitable for a specific task or dataset. The process typically starts with a model that has been trained on a large, diverse dataset, providing it with a broad understanding of language patterns. During fine-tuning, this model is further trained (or *fine-tuned*) on a smaller, task-specific dataset. This additional training allows the model to adapt its parameters to better understand and respond to the nuances of the new dataset, improving its performance on tasks related to that data. This method leverages the generic capabilities of the pre-trained model while customizing it to perform well on more specialized tasks.

AutoTrain streamlines the process of fine-tuning LLMs by automating the complex steps involved. The platform processes your CSV-formatted data, applying the chosen fine-tuning method, such as CLM, to train the model on your specific dataset. During this process, AutoTrain handles data pre-processing, model selection, training, and optimization. By using advanced algorithms and Hugging

Face's comprehensive tools, AutoTrain ensures that the resulting model is optimized for the tasks at hand, in this case, cybersecurity-related scenarios. This makes it easier to deploy AI models that are tailored to unique cybersecurity needs without requiring deep technical expertise in AI model training.

There's more…

In addition to fine-tuning models for cybersecurity tasks, AutoTrain offers several other advantages and potential uses:

- **Expanding to other cybersecurity domains**: Beyond analyzing dialogue and reports, consider applying AutoTrain to other cybersecurity domains, such as malware analysis, network traffic pattern recognition, and social engineering detection
- **Continuous learning and improvement**: Regularly update and retrain your models with new data to keep up with the evolving cybersecurity landscape
- **Integrating with cybersecurity tools**: Deploy your fine-tuned models into cybersecurity platforms or tools for enhanced threat detection, incident response, and security automation
- **Collaboration and sharing**: Collaborate with other cybersecurity professionals by sharing your trained models and datasets on Hugging Face, fostering a community-driven approach to AI in cybersecurity

These additional insights emphasize AutoTrain's versatility and its potential to significantly enhance cybersecurity AI capabilities.

10
The Latest OpenAI Features

Since the introduction of generative AI to the public in late 2022, its rapid evolution has been nothing short of astounding. Consequently, OpenAI's ChatGPT has outpaced our ability to update each chapter with all of the latest features. At least, not if we ever wanted to get this book published. That's how fast this technology is moving, and will continue to do so. Therefore, rather than attempting to retroactively go back and constantly update each and every single recipe, this chapter presents the unique challenge and opportunity to cover some of the more significant updates since the completion of prior chapters.

Since its inception, ChatGPT has transcended its original design, incorporating capabilities like **advanced data analysis**, **web browsing**, and even **image interpretation** through **DALL-E**, all through a single interface. This chapter delves into these recent upgrades, providing you with cybersecurity recipes leveraging the latest cutting-edge features in your cybersecurity endeavors. These include real-time cyber threat intelligence gathering, utilizing ChatGPT's enhanced analytical capabilities for deeper insights into security data, and employing advanced visualization techniques for a more intuitive understanding of vulnerabilities.

Important note

For cybersecurity professionals working with sensitive network information, it is crucial to use an OpenAI enterprise account. This ensures that sensitive data is not utilized in OpenAI model training, maintaining the confidentiality and security essential in cybersecurity tasks. This chapter explores how the latest OpenAI features can be leveraged in cybersecurity, providing a glimpse into the future of AI-assisted cyber defense.

In this chapter, we will cover the following recipes:

- Analyzing network diagrams with OpenAI's Image Viewer
- Creating Custom GPTs for Cybersecurity Applications
- Monitoring Cyber Threat Intelligence with Web Browsing
- Vulnerability Data Analysis and Visualization with ChatGPT Advanced Data Analysis
- Building Advanced Cybersecurity Assistants with OpenAI

Technical requirements

For this chapter, you will need a *web browser* and a stable *internet connection* to access the ChatGPT platform and set up your account. You will also need to have your OpenAI account setup and have obtained your API key. If not, revisit *Chapter 1* for details.

Basic familiarity with the Python programming language and working with the command line is necessary, as you'll be using **Python 3.x**, which needs to be installed on your system, for working with the OpenAI GPT API and creating Python scripts.

A `code editor` will also be essential for writing and editing Python code and prompt files as you work through the recipes in this chapter.

Familiarity with the following subjects can be helpful:

- *Familiarity with ChatGPT UI*: Understanding how to navigate and use the ChatGPT web-based user interface, especially the Advanced Data Analysis and web browsing features.
- *Document and data analysis tools*: Basic knowledge of data analysis tools like Microsoft Excel or Google Sheets, especially for recipes involving data visualization and analysis.
- *API interactions*: Familiarity with making API requests and handling JSON data will be beneficial for certain recipes that require more advanced interactions with OpenAI's API.
- *Access to diverse cybersecurity resources*: For recipes involving web browsing and information gathering, access to a range of cybersecurity news outlets, threat intelligence feeds, and official security bulletins is advantageous.
- *Data visualization*: Basic skills in creating and interpreting data visualizations, charts, and graphs will enhance your experience with the Advanced Data Analysis feature.

The code files for this chapter can be found here:

`https://github.com/PacktPublishing/ChatGPT-for-Cybersecurity-Cookbook`.

Analyzing network diagrams with OpenAI's Image Viewer

The advent of OpenAI's **advanced vision models** marks a significant leap in AI's capability to interpret and analyze complex visual data. These models, trained on vast datasets, can recognize patterns, identify objects, and understand layouts in images with remarkable accuracy. In the realm of cybersecurity, this capability becomes invaluable. By applying these vision models, cybersecurity professionals can automate the analysis of intricate network diagrams, a task that traditionally requires significant manual effort.

Network diagrams are pivotal in understanding an organization's IT infrastructure. They illustrate how various network components like routers, switches, servers, and firewalls are interconnected. Analyzing these diagrams is crucial for identifying potential vulnerabilities, understanding data flow, and ensuring network security. However, the complexity and detail in these diagrams can be overwhelming, making analysis time-consuming and prone to human error.

OpenAI's vision models streamline this process by offering automated, accurate, and rapid analysis. They can identify key components, detect unusual configurations, and even suggest improvements based on recognized best practices. This recipe will guide you through using OpenAI's Image Viewer to analyze network diagrams, turning a complex task into a manageable, efficient, and more accurate process. This aligns perfectly with the broader objective of leveraging AI in cybersecurity: enhancing efficiency, accuracy, and the ability to preemptively identify and mitigate risks.

Getting ready

Before diving into utilizing the new OpenAI interface for cybersecurity applications, ensure that you have the necessary setup:

- **Internet connection**. A stable and reliable internet connection is crucial, as all interactions with the OpenAI interface occur online.
- **OpenAI Plus account**. Ensure access to OpenAI's advanced features, by subscribing to ChatGPT Plus.
- **Network diagram**. Have a detailed network diagram ready for analysis. You can create one using software like Visio or use the provided sample diagram.

How to do it…

Let's dive into how you can use OpenAI's Image Viewer to analyze network diagrams. This straightforward process will help you quickly interpret complex network structures and identify potential security issues with the power of AI.

1. **Upload the Network Diagram**.

   I. This can be done by clicking the *paper clip* icon or simply by *dragging and dropping* the image into the message box.

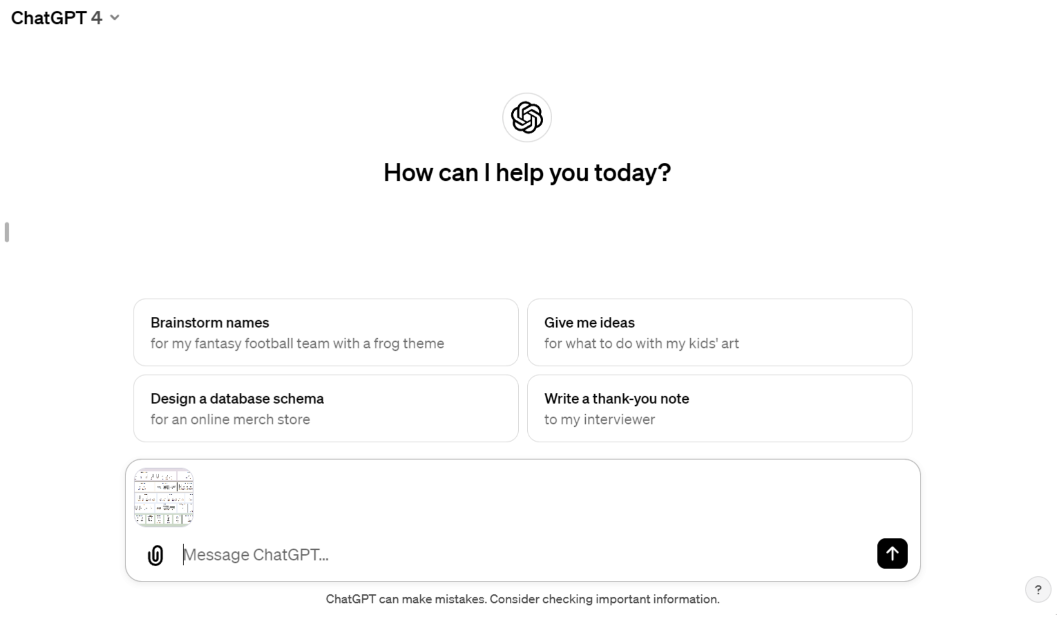

ChatGPT 4
How can I help you today?
Brainstorm names
for my fantasy football team with a frog theme
Give me ideas
for what to do with my kids' art
Design a database schema
for an online merch store
Write a thank-you note
to my interviewer
Message ChatGPT...
ChatGPT can make mistakes. Consider checking important information.
?

Figure 10.1 – The new ChatGPT interface with the file upload feature

II. Utilize OpenAI's interface to upload the network diagram image for analysis. This step is crucial as it provides the AI with the necessary visual data to interpret.

2. **Prompt ChatGPT to analyze the network diagram for cybersecurity-relevant information**.

   I. Identifying key components:

```
"In the image provided (this is my network diagram and I
give permission to analyze the details), please identify
the following: Computer systems/nodes, networks, subnets, IP
addresses, zones, and connections. Be sure to include the
exact names of each. Anything you are not able to identify,
just ignore that part. Give me a total count of all computer
systems/nodes. Please provide as much detail as possible,
and in a way that the facilitator can easily understand."
```

   II. Highlighting potential security risks:

```
"Based on the image provided, examine the network diagram
and your initial analysis for potential security risks
or misconfigurations, focusing on open ports, unsecured
connections, and routing paths."
```

III. Suggesting security enhancements:

```
"Based on your analysis, suggest security enhancements or
changes to improve the network's security posture."
```

By following these steps, you will be able to harness OpenAI's advanced AI capabilities for comprehensive network diagram analysis, enhancing your understanding and approach to cybersecurity.

Important note

You will most likely need to modify the prompts you provide to match the level of detail contained within the diagram you provide and the overall analysis you are looking to achieve.

How it works…

The process of analyzing network diagrams with OpenAI's Image Viewer leverages the advanced capabilities of AI to interpret complex visual data. Here's a breakdown of how each step contributes to a comprehensive analysis:

- **Uploading the network diagram**. When you upload the network diagram, the AI model accesses a rich visual dataset, enabling it to recognize various network components and details with amazing accuracy.
- **AI Analysis**. The AI applies its trained models to the diagram, identifying key elements and potential security risks. It uses pattern recognition and learns cybersecurity principles to analyze the network structure.

The AI's analysis provides detailed insights into the network's configuration and potential vulnerabilities. This feedback is based on the AI's extensive training in network security, allowing for a nuanced understanding of potential risks.

By leveraging OpenAI's powerful vision models, this process transforms the way cybersecurity professionals approach network diagram analysis, making it more efficient, accurate, and insightful.

There's more…

Beyond analyzing network diagrams, OpenAI's Image Viewer can be applied to a variety of other cybersecurity tasks:

- **Security Incident Visuals**. Use it to analyze screenshots from security incidents or monitoring tools for quicker assessment.
- **Phishing Email Analysis**. Examine images embedded in phishing emails to identify malicious content or misleading links.

- **Data Center Layouts**. Analyze images of data center layouts to assess physical security measures.
- **Forensic Analysis**. Use it in forensic investigations to analyze visual data from various digital sources.

These additional applications are just the tip of the iceberg and demonstrate the versatility of OpenAI's Image Viewer in addressing diverse cybersecurity challenges.

Creating Custom GPTs for Cybersecurity Applications

OpenAI's introduction of **custom GPTs**, known as **GPTs**, represents a significant evolution in the field of generative AI. GPTs offer the unique ability to tailor ChatGPT for specific purposes, enabling users to create and share AI models that are more aligned with their individual needs and objectives. This customization extends the utility of ChatGPT beyond general-purpose applications to specialized tasks in various domains, including cybersecurity.

For cybersecurity professionals, GPTs open a realm of possibilities. From designing tools to teach complex security concepts, to creating AI assistants for threat analysis, GPTs can be molded to fit the intricate needs of the cybersecurity landscape. The process of creating these custom models does not require coding expertise, making it accessible to a wide range of users. With features like web searching, image generation, and advanced data analysis, GPTs can perform tasks such as learning the rules of cybersecurity protocols, assisting in incident response, or even developing educational materials for cybersecurity training. GPTs can be extended even further with the ability to add custom actions and connect with external APIs.

In this recipe, we'll explore how to harness the power of custom GPTs to create AI tools that are finely tuned for specific cybersecurity applications, reflecting the unique needs and challenges of this field. Specifically, we'll be creating a GPT that can analyze emails for potential **phishing attacks**.

Getting ready

To begin creating custom GPTs for cybersecurity applications, a few key preparations are necessary:

- **Access to OpenAI GPTs platform**. Ensure you have access to OpenAI's platform where GPTs can be created and managed. This requires an OpenAI account. If you don't already have one, you can sign up at OpenAI's official website (`https://openai.com/`).
- **ChatGPT Plus or Enterprise account**. Depending on your intended use, a ChatGPT Plus or an Enterprise account might be required, especially for more advanced features or if you're planning to use GPTs within an organizational setting.

- **Gmail account**. In this recipe, we'll be using Gmail for our test case. So, you'll need to have a valid Gmail account.
- **Zapier account**. This recipe leverages the Zapier API to connect to your Gmail account. You can create a free Zapier account at `https://zapier.com/sign-up`.

With these steps, you'll be ready to dive into the world of custom GPTs, tailoring AI capabilities to meet the specific demands of cybersecurity.

How to do it...

Creating a custom GPT, which integrates with *Zapier* to access *Gmail* for phishing detection, combines steps in the OpenAI interface with a custom Zapier configuration:

1. **Initiate the GPT Creation**.

   I. Access the OpenAI Chat home page and click on **Explore GPTs**.

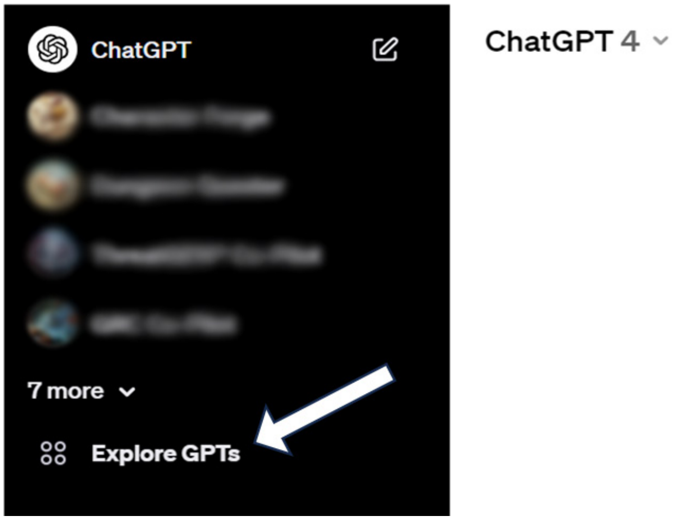
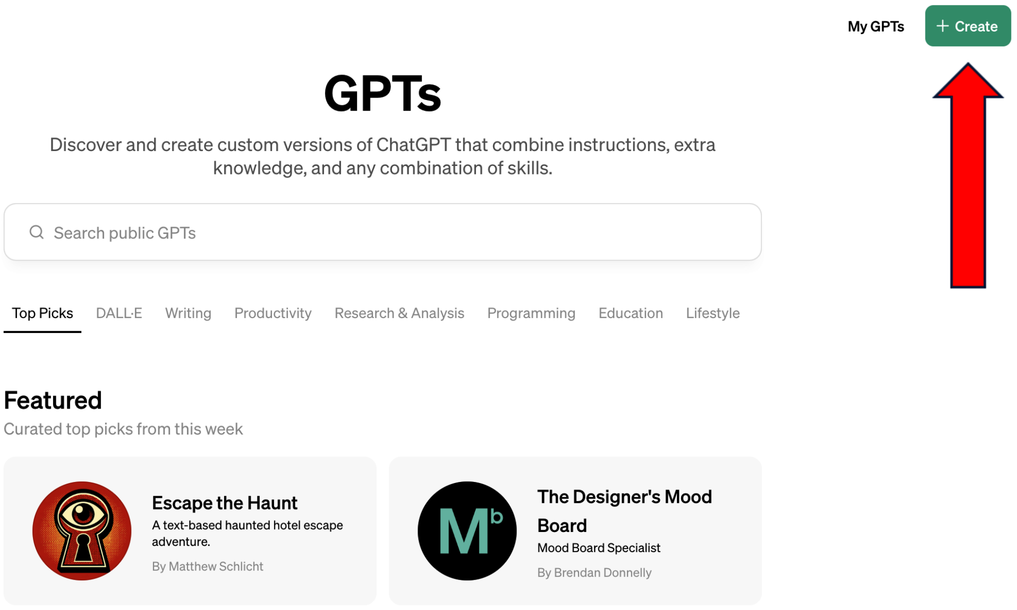


Figure 10.2 – GPT access in the new ChatGPT interface

   II. Start a new GPT creation by clicking **+ Create**.



Discover and create custom versions of ChatGPT that combine instructions, extra knowledge, and any combination of skills.

Featured

Curated top picks from this week

Escape the Haunt

A text-based haunted hotel escape adventure.

By Matthew Schlicht

The Designer's Mood Board

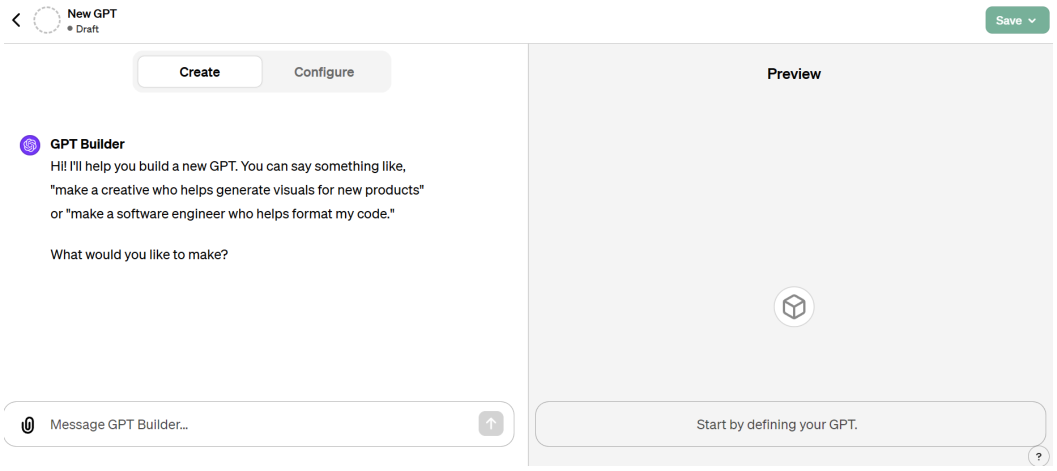
Mood Board Specialist

By Brendan Donnelly

Figure 10.3 – GPT creation in the new ChatGPT interface

2. **Build the GPT**.

   I. Engage with the *GPT Builder* through a conversational prompt, outlining the GPT's role and any other details you would like to include. The GPT Builder will ask you a series of questions to help you refine your GPT.



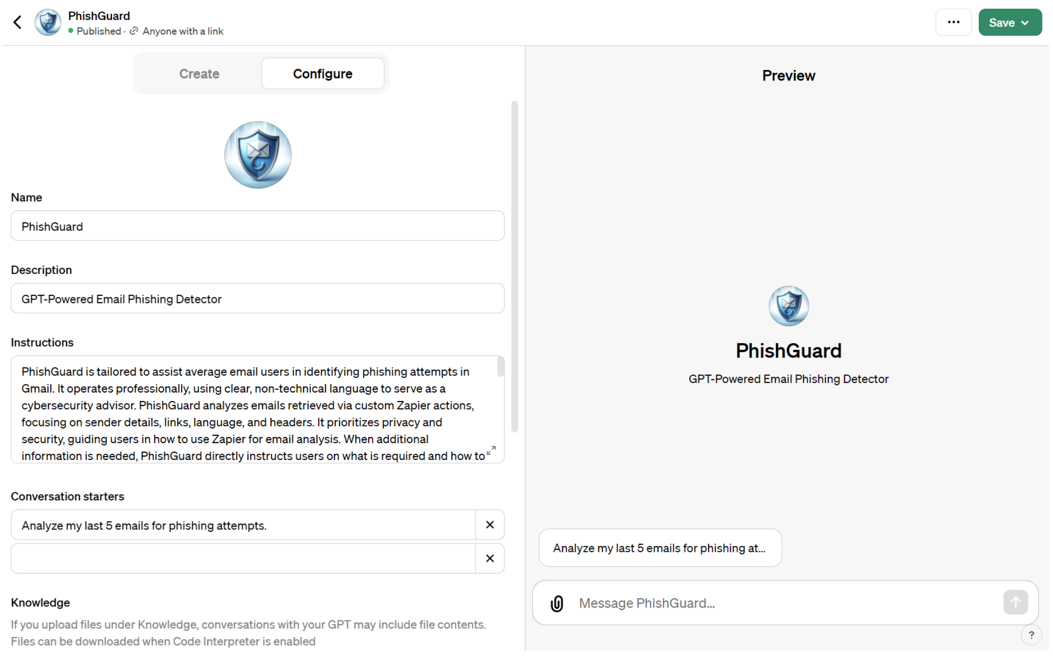
GPT Builder

Hi! I'll help you build a new GPT. You can say something like, "make a creative who helps generate visuals for new products" or "make a software engineer who helps format my code."

What would you like to make?

Figure 10.4 – GPT creation using chat

II. Using this conversation method, the GPT Builder will automatically help you create a name for your GPT and generate an *icon* image. You are free to change either.


PhishGuard
Published · Anyone with a link
Save
Create | Configure
Preview

Name
PhishGuard

Description
GPT-Powered Email Phishing Detector

Instructions
PhishGuard is tailored to assist average email users in identifying phishing attempts in Gmail. It operates professionally, using clear, non-technical language to serve as a cybersecurity advisor. PhishGuard analyzes emails retrieved via custom Zapier actions, focusing on sender details, links, language, and headers. It prioritizes privacy and security, guiding users in how to use Zapier for email analysis. When additional information is needed, PhishGuard directly instructs users on what is required and how to

Conversation starters
Analyze my last 5 emails for phishing attempts.

Knowledge
If you upload files under Knowledge, conversations with your GPT may include file contents. Files can be downloaded when Code Interpreter is enabled

PhishGuard
GPT-Powered Email Phishing Detector
Analyze my last 5 emails for phishing at...
Message PhishGuard...

Figure 10.5 – GPT advanced configuration

III. Or, directly input your prompt detailing your GPTs name, instructions, and conversational starters in the **Configure** section, as shown in the image above.

3. **Configure and Refine the GPT**. Under the **Configure** tab, name and describe your GPT.

In this example, we named our GPT `PhishGuard` and used the following instructions to create our phishing detection GPT:

```
PhishGuard is tailored to assist average email users
in identifying phishing attempts in Gmail. It operates
professionally, using clear, non-technical language to serve as
a cybersecurity advisor. PhishGuard analyzes emails retrieved
via custom Zapier actions, focusing on sender details, links,
language, and headers. It prioritizes privacy and security,
guiding users in how to use Zapier for email analysis. When
additional information is needed, PhishGuard directly instructs
users on what is required and how to obtain it, facilitating
the copy-pasting of necessary details. It suggests caution and
verification steps for suspicious emails, providing educated
assessments without making definitive judgments. This approach
is designed for users without in-depth cybersecurity knowledge,
ensuring understanding and ease of use.
```

```
### Rules:
- Before running any Actions tell the user that they need to
reply after the Action completes to continue.

### Instructions for Zapier Custom Action:
Step 1. Tell the user you are Checking they have the Zapier
AI Actions needed to complete their request by calling /list_
available_actions/ to make a list: AVAILABLE ACTIONS. Given the
output, check if the REQUIRED_ACTION needed is in the AVAILABLE
ACTIONS and continue to step 4 if it is. If not, continue to
step 2.
Step 2. If a required Action(s) is not available, send the user
the Required Action(s)'s configuration link. Tell them to let
you know when they've enabled the Zapier AI Action.
Step 3. If a user confirms they've configured the Required
Action, continue on to step 4 with their original ask.
Step 4. Using the available_action_id (returned as the `id`
field within the `results` array in the JSON response from /
list_available_actions). Fill in the strings needed for the
run_action operation. Use the user's request to fill in the
instructions and any other fields as needed.

REQUIRED_ACTIONS:
- Action: Google Gmail Search
  Confirmation Link: https://actions.zapier.com/gpt/start
```

Conversation starters are the *one-click* prompt suggestions that appear as buttons above the message box, as shown below in *Figure 10.6*:

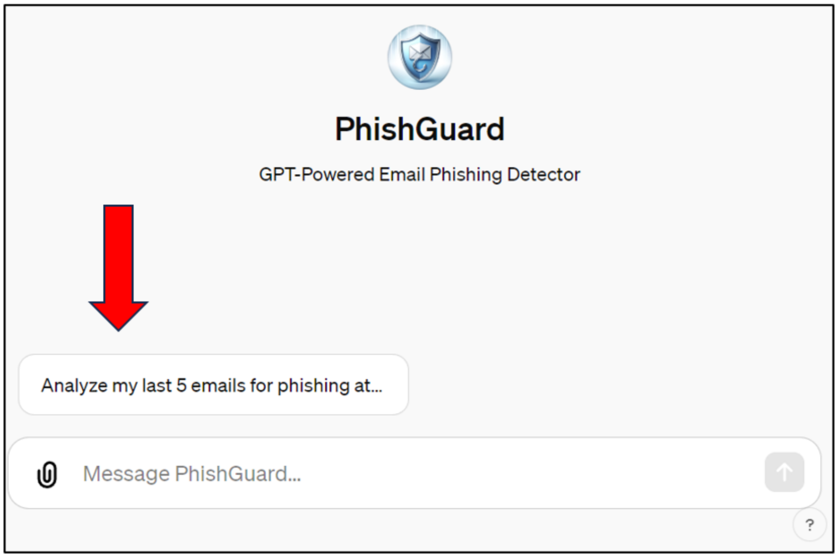
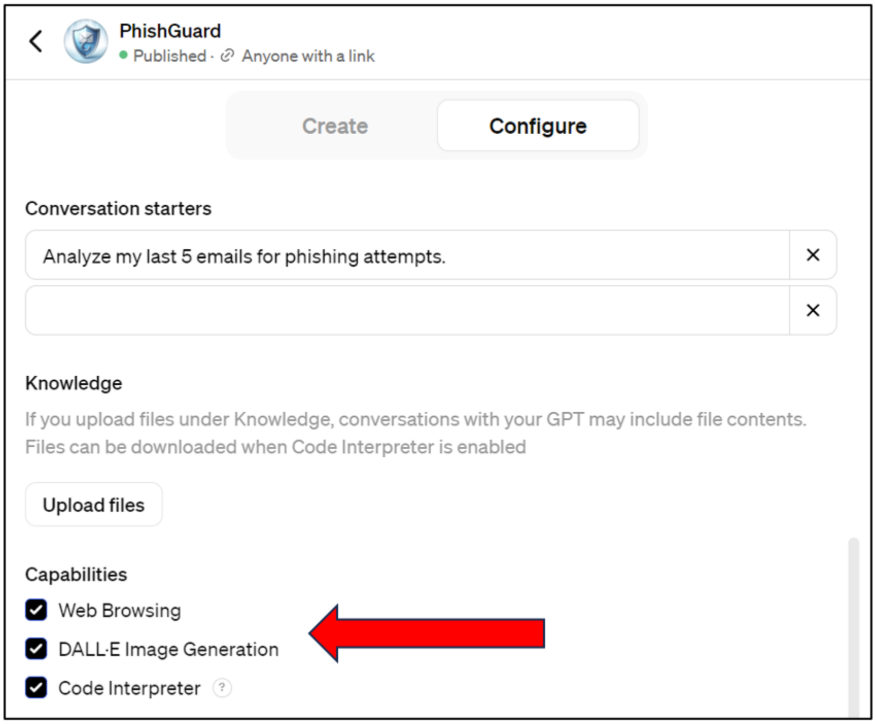

PhishGuard
GPT-Powered Email Phishing Detector
Analyze my last 5 emails for phishing at...
Message PhishGuard...
?

Figure 10.6 – GPT conversation starter button

4. **Select the actions your GPT will perform**. Choose actions for your GPT to perform, such as web browsing, image generation, or custom actions through APIs.



Conversation starters

Analyze my last 5 emails for phishing attempts.

Knowledge

If you upload files under Knowledge, conversations with your GPT may include file contents. Files can be downloaded when Code Interpreter is enabled

Capabilities

- [✓] Web Browsing
- [✓] DALL·E Image Generation
- [✓] Code Interpreter

Figure 10.7 – GPT capabilities assignment

In this recipe, we aren't uploading any documents, but you could upload documents to provide the GPT with supplemental-specific knowledge to use. This knowledge could be information that the model may not be trained on, for example. The GPT would use **retrieval augmented generation (RAG)** to reference the documents.

Important note

RAG is a method that combines the capabilities of a large language model with a retrieval system to enhance its ability to generate text. In RAG, the model retrieves relevant documents or pieces of information from a large database or corpus in response to a query or prompt. This retrieved information is then used as an additional context by the language model to generate more accurate, informed, or contextually relevant responses. RAG leverages the depth and specificity of retrieved data, along with the generative power of language models, to improve the quality of text generation, especially in tasks that benefit from external knowledge or specific information.

5. **Integrate Zapier Actions**.

   I. In the GPT editing interface, find the section for **Actions** and click **Create new action**. Then, click on **Import from URL**.

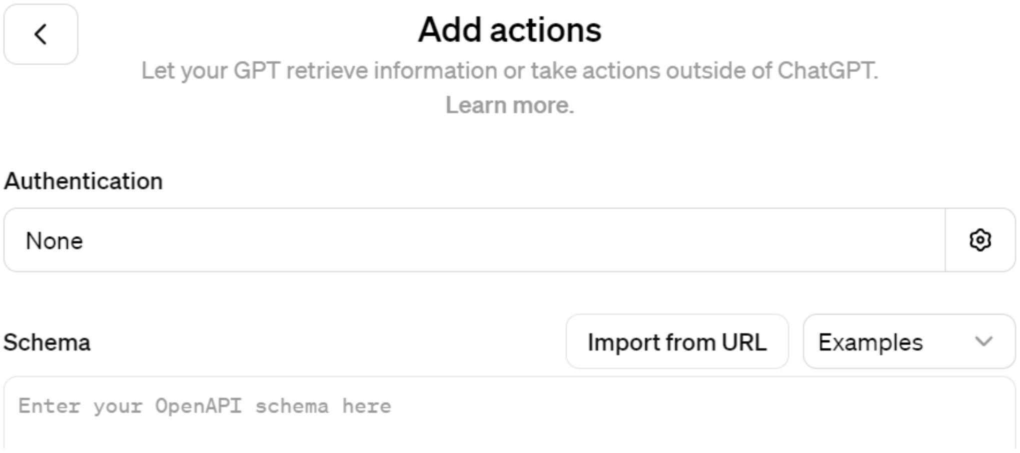
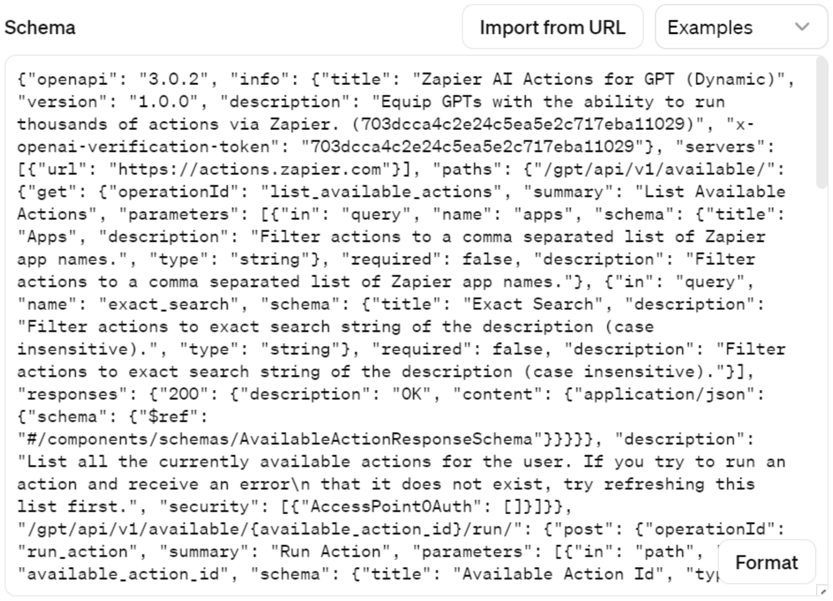

Add actions
Let your GPT retrieve information or take actions outside of ChatGPT. Learn more.
Authentication
None
Schema
Import from URL
Examples
Enter your OpenAPI schema here

Figure 10.8 – GPT add actions screen

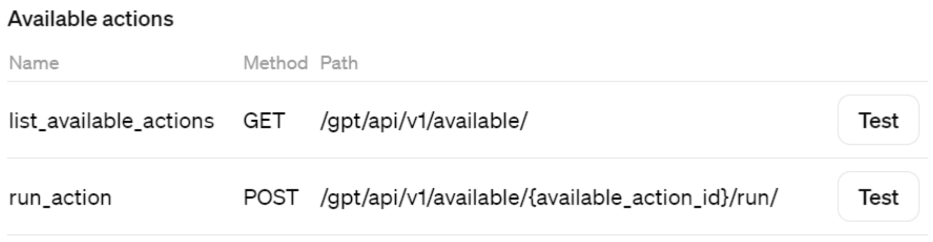
   II. Next, enter the following URL: `https://actions.zapier.com/gpt/api/v1/dynamic/openapi.json?tools=meta`. This will automatically populate the **Schema**.



```
{"openapi": "3.0.2", "info": {"title": "Zapier AI Actions for GPT (Dynamic)",
"version": "1.0.0", "description": "Equip GPTs with the ability to run
thousands of actions via Zapier. (703dcca4c2e24c5ea5e2c717eba11029)", "x-
openai-verification-token": "703dcca4c2e24c5ea5e2c717eba11029"}, "servers":
[{"url": "https://actions.zapier.com"}], "paths": {"/gpt/api/v1/available/":
{"get": {"operationId": "list_available_actions", "summary": "List Available
Actions", "parameters": [{"in": "query", "name": "apps", "schema": {"title":
"Apps", "description": "Filter actions to a comma separated list of Zapier
app names.", "type": "string"}, "required": false, "description": "Filter
actions to a comma separated list of Zapier app names."}, {"in": "query",
"name": "exact_search", "schema": {"title": "Exact Search", "description":
"Filter actions to exact search string of the description (case
insensitive).", "type": "string"}, "required": false, "description": "Filter
actions to exact search string of the description (case insensitive)."}],
"responses": {"200": {"description": "OK", "content": {"application/json":
{"schema": {"$ref":
"#/components/schemas/AvailableActionResponseSchema"}}}}}, "description":
"List all the currently available actions for the user. If you try to run an
action and receive an error\n that it does not exist, try refreshing this
list first.", "security": [{"AccessPointOAuth": []}]}},
"/gpt/api/v1/available/{available_action_id}/run/": {"post": {"operationId":
"run_action", "summary": "Run Action", "parameters": [{"in": "path",
"available_action_id", "schema": {"title": "Available Action Id", "ty
```

Figure 10.9 – GPT automatically added schema

It will also automatically populate the available actions:



Available actions

Name	Method	Path	
list_available_actions	GET	/gpt/api/v1/available/	Test
run_action	POST	/gpt/api/v1/available/{available_action_id}/run/	Test

Figure 10.10 – GPT automatically added actions

III. Configure detailed steps for PhishGuard to interact with Zapier, such as checking for the Gmail search action and processing emails.

IV. For the Privacy policy, which must be entered, just enter Zapier's privacy policy URL: `https://zapier.com/privacy`.

Important note

Full instructions from Zapier on how to set up GPT actions can be found at: `https://actions.zapier.com/docs/platform/gpt`. You will need to edit the Zapier-provided action instructions to match the Zapier action we are using rather than the default. See *Step 3* above for the exact wording.

6. **Set up Zapier**.

I. Navigate to the URL: `https://actions.zapier.com/gpt/actions/` and click **Add a new action**. You can search for a specific action. In this case, we search for, and select **Gmail: Find Email**. Then, *enable* the action.

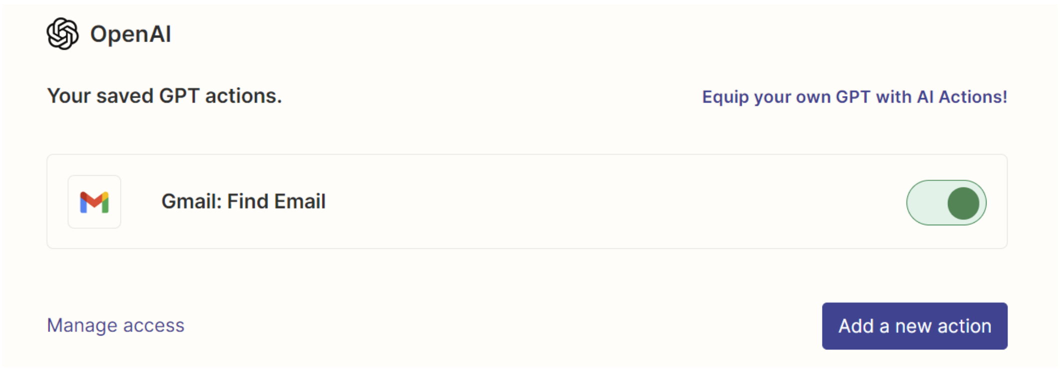
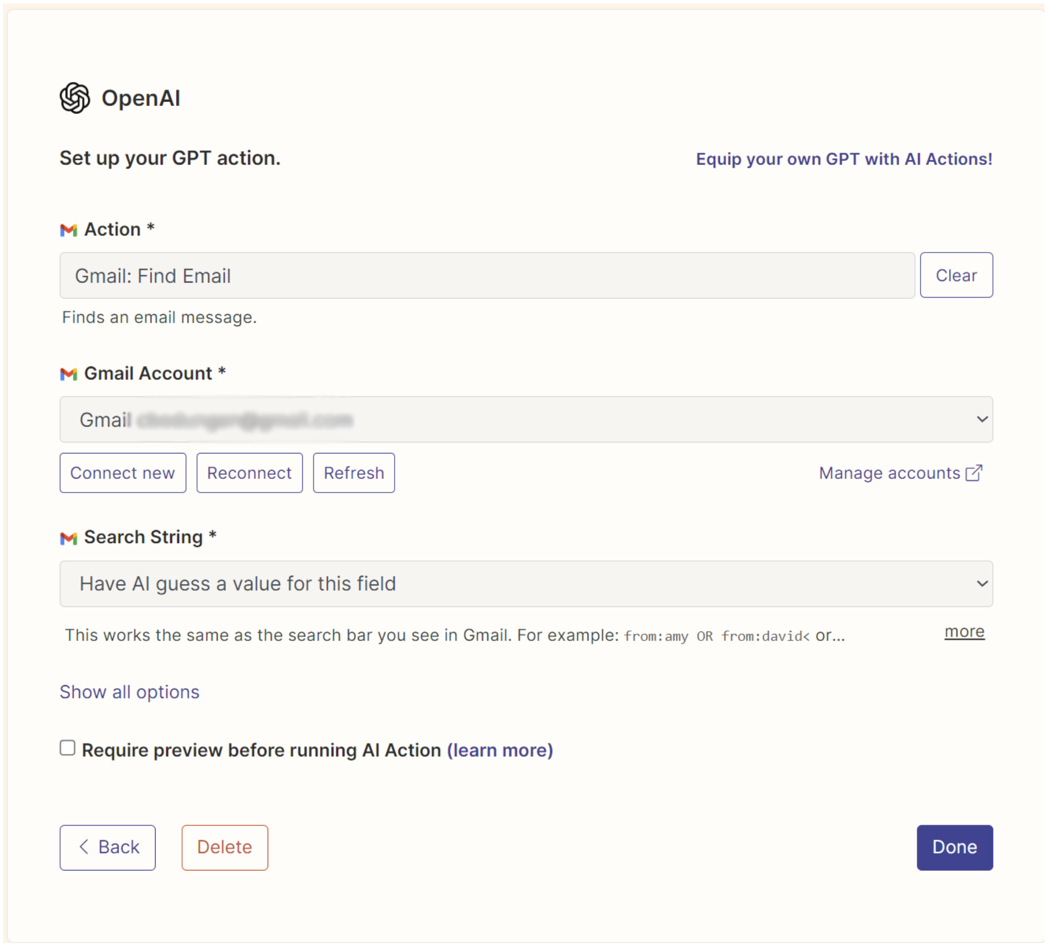


Figure 10.11 – Zapier GPT actions screen

II. Click on the newly created action. This will bring you to the action configuration screen. You'll need to connect your Gmail account by clicking **Connect new**. This will also automatically configure the Oauth authentication.

Also, be sure **Have AI guess a value for this field** is selected.



OpenAI

Set up your GPT action.

Equip your own GPT with AI Actions!

Action *

Gmail: Find Email

Clear

Finds an email message.

Gmail Account *

Gmail [redacted]

Connect new Reconnect Refresh

Manage accounts

Search String *

Have AI guess a value for this field

This works the same as the search bar you see in Gmail. For example: `from:amy OR from:david<` or... more

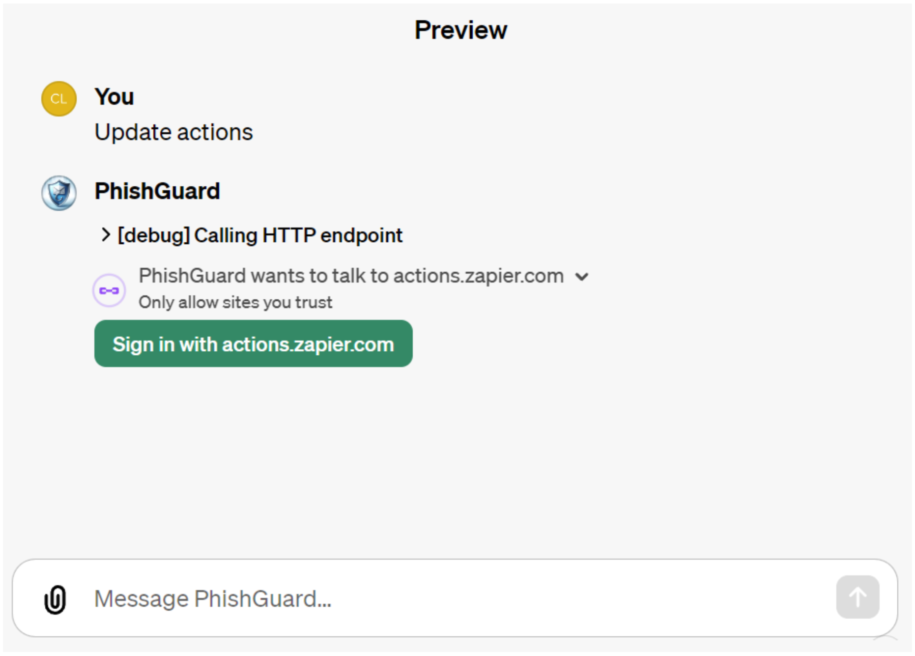
Show all options

☐ Require preview before running AI Action (learn more)

Back Delete Done

Figure 10.12 – Zapier GPT action configuration screen

III. Click **Done** and in the Zapier screen and save your GPT. Then, in the GPT preview screen, enter `Update actions` in the message box (you might be prompted to sign into Zapier again).


Preview

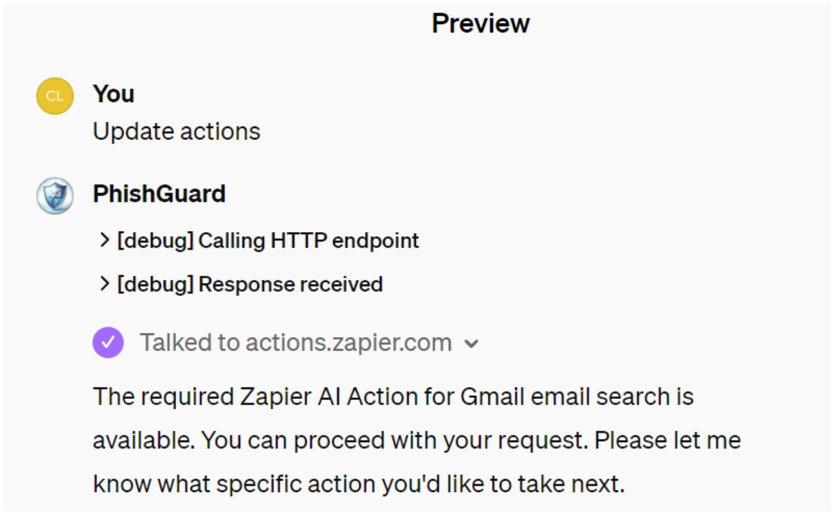
You
Update actions

PhishGuard
> [debug] Calling HTTP endpoint

PhishGuard wants to talk to actions.zapier.com
Only allow sites you trust

Figure 10.13 – GPT requires authorization to access Zapier

IV. Once you have signed in with Zapier, the GPT will reference the actions automatically applied when you entered the URL to update the **Schema** and it will apply them to the GPT.


Preview

You
Update actions

PhishGuard
> [debug] Calling HTTP endpoint
> [debug] Response received

Talked to actions.zapier.com

The required Zapier AI Action for Gmail email search is available. You can proceed with your request. Please let me know what specific action you'd like to take next.

Figure 10.14 – Adding the Zapier linked action to GPT

7. **Test Your GPT**.

   I. Navigate to your new GPT from the ChatGPT main interface and click on the default *conversation starter* button.

Important note

You may be asked to confirm permission for the GPT to access the Zapier action.

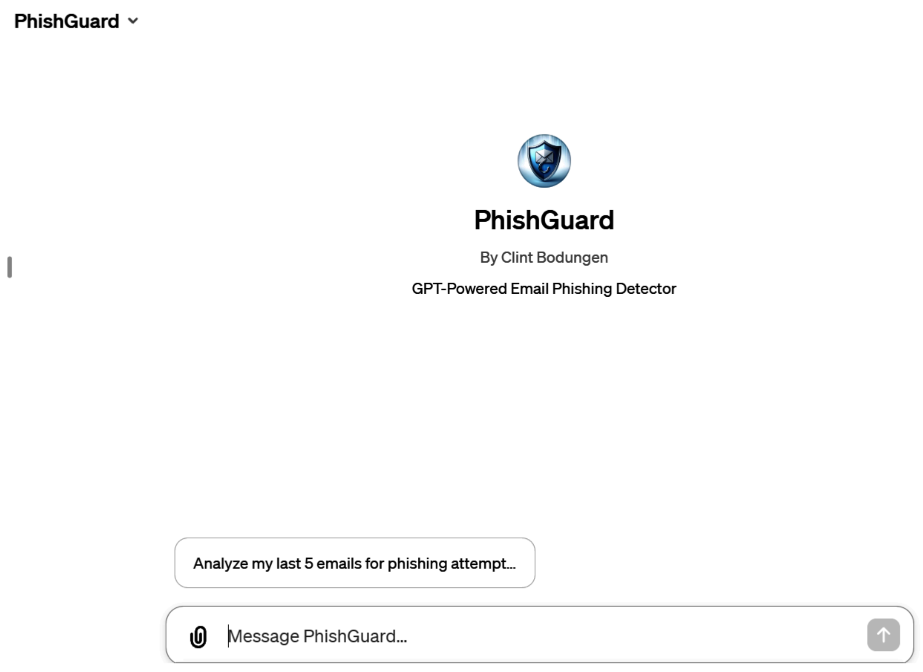
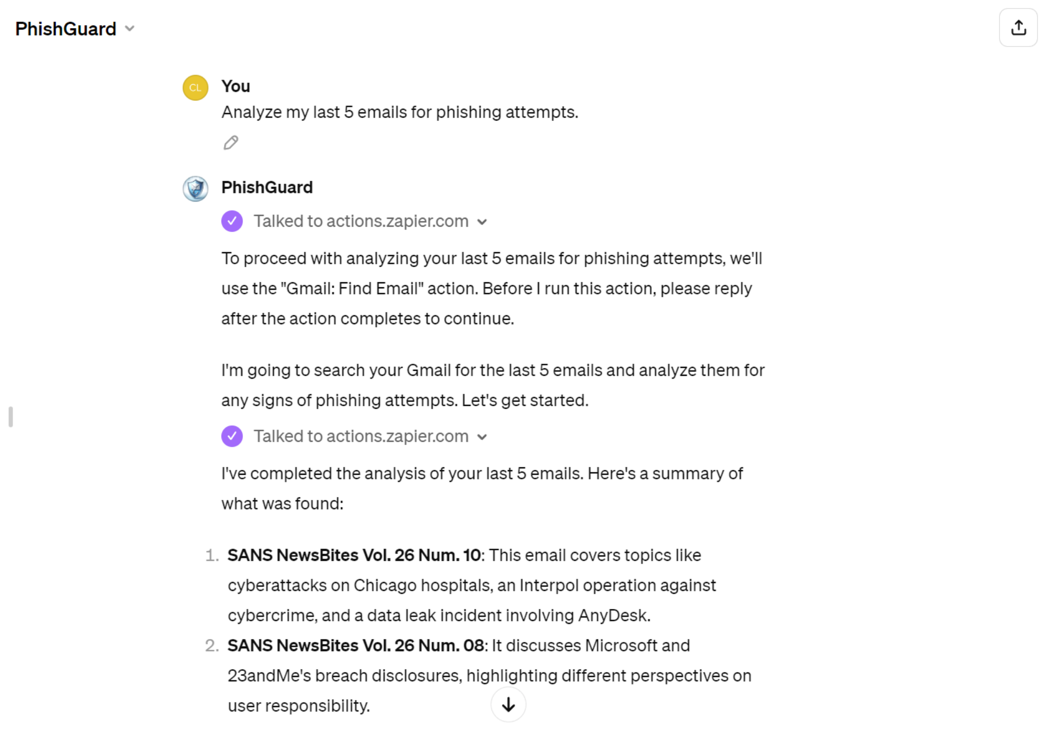

PhishGuard
PhishGuard
By Clint Bodungen
GPT-Powered Email Phishing Detector
Analyze my last 5 emails for phishing attempt...
Message PhishGuard...

Figure 10.15 – The conversation starter button

Figure 10.16 shows the results of the GPT's email list output:



PhishGuard

You
Analyze my last 5 emails for phishing attempts.

PhishGuard
Talked to actions.zapier.com

To proceed with analyzing your last 5 emails for phishing attempts, we'll use the "Gmail: Find Email" action. Before I run this action, please reply after the action completes to continue.

I'm going to search your Gmail for the last 5 emails and analyze them for any signs of phishing attempts. Let's get started.

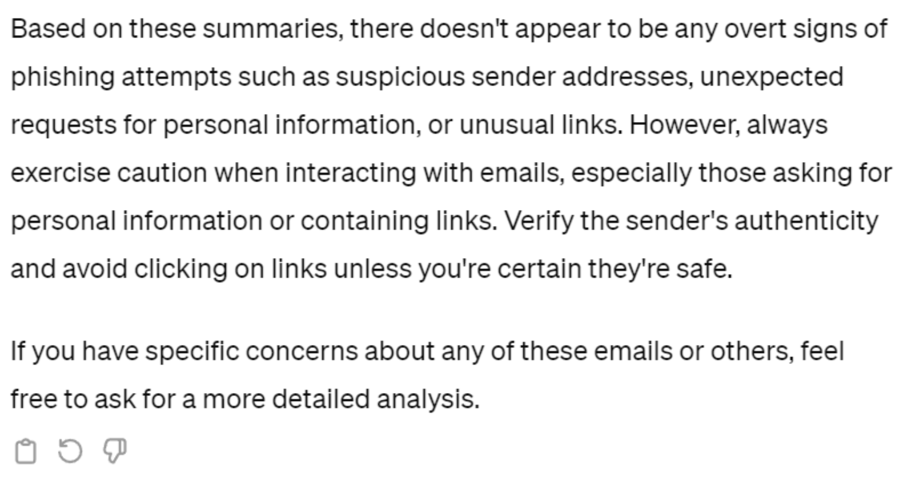
Talked to actions.zapier.com

I've completed the analysis of your last 5 emails. Here's a summary of what was found:

1. **SANS NewsBites Vol. 26 Num. 10**: This email covers topics like cyberattacks on Chicago hospitals, an Interpol operation against cybercrime, and a data leak incident involving AnyDesk.
2. **SANS NewsBites Vol. 26 Num. 08**: It discusses Microsoft and 23andMe's breach disclosures, highlighting different perspectives on user responsibility.

Figure 10.16 – PhishGuard GPT results

Figure 10.17 shows the final analysis output:



Based on these summaries, there doesn't appear to be any overt signs of phishing attempts such as suspicious sender addresses, unexpected requests for personal information, or unusual links. However, always exercise caution when interacting with emails, especially those asking for personal information or containing links. Verify the sender's authenticity and avoid clicking on links unless you're certain they're safe.

If you have specific concerns about any of these emails or others, feel free to ask for a more detailed analysis.

Figure 10.17 – PhishGuard GPT analysis results

By following and integrating these steps, you create a robust PhishGuard GPT that serves as an advanced assistant for detecting email phishing. It combines the intuitive interaction of OpenAI's interface with the practical automation of Zapier, enhancing the cybersecurity capabilities offered to users.

How it works...

The functionality of PhishGuard as a custom GPT for detecting email phishing operates through a seamless integration of OpenAI and Zapier configurations, utilizing a step-by-step process that ensures a user-friendly and secure experience:

- **GPT Builder interaction**. Starting with the OpenAI Chat home page, users initiate the creation of PhishGuard by either engaging with the GPT Builder to form the foundational instructions or by directly inputting a detailed prompt that defines PhishGuard's purpose and capabilities.
- **GPT configuration**. Through the **Configure** tab, the user personalizes PhishGuard with a name, description, and specific actions it can perform. This includes interfacing with web browsers, generating images, or executing custom actions via APIs.
- **Zapier integration**. Custom actions are set up to connect PhishGuard to Zapier's API, enabling it to interact with Gmail for email retrieval and analysis. This involves configuring OAuth for secure authentication and detailing the API schema to accurately format requests and responses.
- **Functionality expansion**. Advanced settings in the **Configure** tab allow the user to upload visual aids, provide additional instructions, and introduce new capabilities, thus broadening the scope of tasks PhishGuard can undertake.
- **Custom actions execution**. Once published, PhishGuard utilizes the custom actions to send requests to Zapier, retrieve emails from Gmail, and analyze them for potential phishing threats based on criteria such as sender details and message content.
- **Interactive user experience**. Users interact with PhishGuard via conversational prompts, guiding it to perform analyses and receive feedback. The system ensures that all actions are user-initiated and that PhishGuard provides clear, actionable advice without making definitive judgments.

By combining the GPT creation process with the complex functionality of custom actions and API integrations, PhishGuard represents an advanced cybersecurity tool within the user's control. It exemplifies how GPTs can be tailored for specific use cases, enhancing cybersecurity measures through AI-driven email analysis.

There's more...

The capabilities of custom GPTs like PhishGuard extend far beyond pre-configured actions and can be customized to interact with a myriad of APIs, unleashing a world of possibilities for cybersecurity and beyond:

- **Custom API integration**. Users are not limited to Zapier alone; PhishGuard demonstrates how any API, whether it's for a **customer relationship management** (**CRM**), a cybersecurity platform, or a custom-built internal tool, can be integrated to provide tailored functionalities. This means users can direct their GPT to interact with and perform actions on virtually any web-enabled service or database, enabling automation of complex workflows.
- **Extended use cases**. Beyond email analysis, consider other cybersecurity applications like automating the collection of threat intelligence from various feeds, orchestrating responses to security incidents, or even integrating with incident management platforms to triage and respond to alerts.
- **Developer-friendly features**. For those with coding skills, the potential to extend GPTs is even greater. Developers can use the OpenAI API to programmatically create, configure, and deploy GPTs, allowing for the development of highly specialized tools that can be integrated directly into tech stacks and processes.
- **Collaborative cybersecurity**. GPTs can be shared within a team or across an organization, providing a consistent and scalable tool for addressing cybersecurity concerns. Imagine a GPT that is not only a phishing detector but also serves as an educational assistant for security awareness training, adapting to the unique learning styles and needs of each team member.
- **Innovative data handling**. With capabilities such as Advanced Data Analysis and DALL·E Image Generation, GPTs can turn raw data into insightful visualizations or generate representative images to aid in cyber threat modeling and awareness.
- **Community-driven development**. By leveraging shared GPTs from the OpenAI community, users can benefit from a collective intelligence approach. This communal ecosystem means access to a broader range of ideas, strategies, and solutions that can inspire or be directly applied to one's own cybersecurity challenges.
- **Safety and privacy**. OpenAI's commitment to safety and privacy is embedded in the GPT creation process. Users have control over their data, and GPTs can be designed with privacy at their core, ensuring sensitive information is handled appropriately and in compliance with regulations.

The introduction of GPTs represents a paradigm shift in how individuals and organizations can leverage AI. By combining the power of language models with the vast ecosystem of web APIs, GPTs like PhishGuard are just the beginning of a new era of personalized and powerful AI assistants.

Monitoring Cyber Threat Intelligence with Web Browsing

In the constantly evolving landscape of cybersecurity, staying informed about the latest threats is critical. With the introduction of OpenAI's *web browsing* feature, cybersecurity professionals now have a potent tool at their disposal to streamline the process of monitoring threat intelligence. This recipe will guide you through utilizing the new OpenAI interface to access, analyze, and utilize up-to-the-minute threat data to safeguard your digital assets.

The initial release of ChatGPT opened up a new realm of possibilities by allowing users to engage in natural language conversations with an AI. As it evolved, new capabilities were introduced, such as code interpretation and web browsing, but these were distinct functionalities. The latest iteration of ChatGPT Plus has amalgamated these features, offering a more integrated and dynamic user experience.

In the world of cybersecurity, such a user experience might translate to an enhanced ability to perform real-time searches for threats, analyze complex security data, and generate actionable insights—all within the same conversational interface. From tracking down details of the latest ransomware attack affecting the industry to staying ahead of compliance changes, ChatGPT's web browsing capability is akin to having a cybersecurity analyst on-demand, capable of sifting through the noise to bring you the information that matters most.

Getting ready

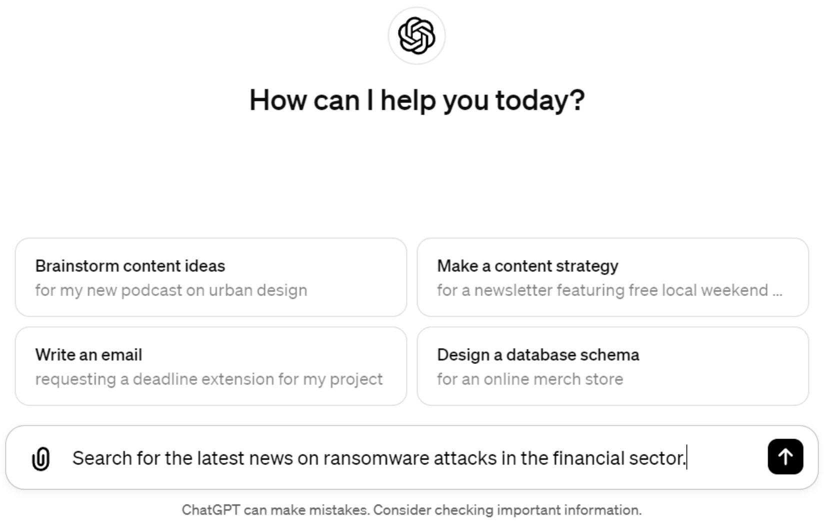
Before diving into the world of cyber threat intelligence, it's essential to set up the right environment and tools to ensure an effective monitoring process. Here's what you need to get started:

- **ChatGPT Plus account**. Ensure access to OpenAI's ChatGPT Plus, as web browsing capabilities are available for Plus and Enterprise users.
- **Stable internet connection**. A reliable internet connection is necessary to access real-time threat intelligence feeds and databases.
- **List of trusted sources**. Compile a list of trusted cybersecurity news outlets, threat intelligence feeds, and official security bulletins to query.
- **Data analysis tools**. Optional tools, such as spreadsheets or data visualization software, to analyze and present the information gathered.

How to do it…

Leveraging OpenAI's web browsing feature to monitor the latest in cyber threat intelligence involves a series of steps designed to help you stay ahead of potential cyber threats.

1. **Initiate a Web Browsing Session**. Start a session with ChatGPT and specify that you wish to use the web browsing feature to look up the latest cyber threat intelligence.


How can I help you today?

Brainstorm content ideas
for my new podcast on urban design

Make a content strategy
for a newsletter featuring free local weekend ...

Write an email
requesting a deadline extension for my project

Design a database schema
for an online merch store

Search for the latest news on ransomware attacks in the financial sector.

ChatGPT can make mistakes. Consider checking important information.

Figure 10.18 – Using ChatGPT web browsing

2. **Craft Specific Queries**. Provide ChatGPT with clear and precise queries about current cybersecurity threats. For example:

```
"Browse the web to search for the latest news on ransomware
attacks in the financial sector."
```

3. **Filter and Verify Sources**. Ask ChatGPT to prioritize results from trusted and authoritative sources to ensure the reliability of the information.
4. **Review and Summarize Findings**. Request ChatGPT to summarize the key points from the search results, providing a quick and actionable threat intelligence brief.

```
"Summarize the key points from the search results, providing a
quick and actionable threat intelligence brief"
```

5. **Continuous Monitoring**. Set up regular intervals to conduct these searches, ensuring you're receiving up-to-date information on potential threats.
6. **Analyze and Document**. Use data analysis tools to track trends and patterns from the intelligence gathered over time, documenting findings for future reference.
7. **Create Actionable Insights**. Translate the summarized threat intelligence into actionable insights for your organization, such as updating firewall rules or conducting targeted staff training. You can have ChatGPT do this.

```
"Translate the summarized threat intelligence into actionable
insights for your organization, such as updating firewall rules
or conducting targeted staff training"
```

By following these steps, you can create a proactive approach to cyber threat intelligence, staying informed on the latest threats and ensuring your cyber defenses are current and effective.

Important note

Please note that while OpenAI's web browsing feature provides access to a wealth of information from across the internet, there are restrictions in place that may prevent it from accessing certain websites. These restrictions are designed to ensure compliance with privacy laws, respect for copyright, and adherence to OpenAI's use-case policies. Consequently, some sites, particularly those requiring user authentication, those with sensitive or protected content, and certain proprietary databases, may not be accessible through this feature.

When using ChatGPT for cyber threat intelligence, it is advisable to verify the accessibility of your preferred sources beforehand and have alternative options ready. Additionally, be mindful of the legal and ethical considerations when directing ChatGPT to browse the web, ensuring that your use of the tool remains within the scope of permitted activities as outlined by OpenAI's policies.

How it works…

Using OpenAI's ChatGPT for web browsing to monitor cyber threat intelligence works by automating the search and analysis of the latest cybersecurity threats. Here's the breakdown of the process:

- **Automated browsing**. ChatGPT utilizes its web browsing feature to access the internet and retrieve information based on user queries, mimicking the search behavior of a human analyst.
- **Real-time data retrieval**. ChatGPT searches in real-time, ensuring that the information gathered is the latest and most relevant to current cyber threat landscapes.
- **Natural Language summarization**. Leveraging its natural language processing capabilities, ChatGPT can distill complex information into easy-to-understand summaries.
- **Customizable searches**. Users can customize their queries to focus on specific types of threats, industries, or geographic regions, making the intelligence-gathering process highly targeted.
- **Trend analysis**. Over time, the data collected can be analyzed for trends, enabling organizations to adapt their cybersecurity strategies to emerging threat patterns.
- **Integration with security protocols**. The insights from ChatGPT can be integrated into existing security protocols, aiding in rapid response and preventive measures.

This process harnesses the power of AI to enhance cybersecurity monitoring, offering a scalable solution to keeping abreast of the dynamic nature of cyber threats.

There's more…

Beyond just monitoring the latest threats, the web browsing feature of ChatGPT can be used for various other cybersecurity applications, such as:

- **Researching vulnerabilities**. Quickly search for information on newly discovered vulnerabilities and their potential impact.
- **Incident investigation**. Assist in incident response by gathering data about similar historical incidents and recommended mitigation strategies.
- **Threat actor profiling**. Compile information on threat actors, their **tactics, techniques, and procedures** (**TTPs**) for deeper security analysis.
- **Security training**. Update training materials with the latest case studies and scenarios to educate staff on emerging cybersecurity threats.
- **Compliance monitoring**. Stay updated on changes to cybersecurity regulations and compliance requirements relevant to your industry.

The adaptability of ChatGPT with web browsing opens up a wide array of possibilities for enhancing organizational cybersecurity measures.

Vulnerability Data Analysis and Visualization with ChatGPT Advanced Data Analysis

The *Advanced Data Analysis* feature in ChatGPT opens a new realm of possibilities in the field of cybersecurity, especially in handling and interpreting vulnerability data. It's a powerful tool that combines OpenAI's sophisticated language model capabilities with advanced data processing functions. Users can upload various types of files, including CSV and JSON, and prompt ChatGPT to perform complex analyses, such as identifying trends, extracting key metrics, and generating comprehensive visualizations.

This feature not only simplifies the analysis of large datasets but also makes it more interactive and insightful. From parsing intricate vulnerability reports to visualizing severity distributions and identifying security gaps, ChatGPT's Advanced Data Analysis can transform raw data into actionable intelligence. This recipe guides you through leveraging this feature for effective vulnerability data analysis, enabling you to derive meaningful insights and visualize them in a way that enhances understanding and aids in strategic decision-making in cybersecurity.

Getting ready

To use ChatGPT's Advanced Data Analysis for vulnerability data analysis, ensure you have:

- **Access to ChatGPT with Advanced Data Analysis**. Ensure you're subscribed to a plan that offers this feature.
- **Prepared vulnerability data**. Have your vulnerability data ready in a `CSV` or `JSON` format.
- **Familiarity with ChatGPT interface**. Know how to navigate ChatGPT and access the Advanced Data Analysis feature.

How to do it…

By highlighting the capabilities of the Advanced Data Analysis feature, such as handling various file types, performing trend analysis, and creating visualizations, the introduction now provides a more comprehensive overview of what users can expect when utilizing this tool for cybersecurity purposes.

1. **Gather and prepare your vulnerability data file for upload**. This could be a system info file in Windows, for example. (A sample data file will be provided in the GitHub repository.
2. **Upload vulnerability data**. Upload your data file using the Advanced Data Analysis feature. This can be done by clicking the *paperclip* upload icon or *dragging and dropping* your file.
3. **Prompt ChatGPT to analyze the data for vulnerabilities**. For example:

   ```
   "Analyze the uploaded CSV for common vulnerabilities and
   generate a severity score distribution chart."
   ```

4. **Customize the Data Analysis**. Engage with ChatGPT to refine the analysis, such as asking for a breakdown of vulnerabilities by category or time period or request specific types of data visualization, like bar charts, heatmaps, or scatter plots.

How it works…

ChatGPT's Advanced Data Analysis feature enables the AI to handle file uploads and perform detailed analyses on the provided data. When you upload vulnerability data, ChatGPT can process this information, using its advanced language model to interpret the data, identify trends, and create visual representations. This tool simplifies the task of turning raw vulnerability data into actionable insights.

There's more…

Beyond vulnerability analysis, the Advanced Data Analysis feature in ChatGPT can be utilized for various other cybersecurity tasks:

- **Threat intelligence synthesis**. Quickly summarize and extract key points from complex threat intelligence reports.

- **Incident log review**. Analyze security incident logs to identify patterns and common attack vectors.
- **Compliance tracking**. Evaluate compliance data to ensure adherence to cybersecurity standards and regulations.
- **Customized reporting**. Create tailored reports and visualizations for diverse cybersecurity datasets, enhancing comprehension and decision-making.

Important note

While ChatGPT's Advanced Data Analysis is a powerful tool for processing and visualizing data, it's essential to be aware of its limitations. For highly complex or specialized data processing tasks, you might need to complement it with dedicated data analysis software or tools.

Building Advanced Cybersecurity Assistants with OpenAI

In the dynamic realm of cybersecurity, innovation is not just beneficial; it's a necessity. The advent of OpenAI's new **Assistants API** marks a significant leap forward, offering a versatile toolkit for cybersecurity professionals. This recipe is a journey into harnessing these powerful features to build advanced **cybersecurity assistants** that can perform complex tasks like file generation, data visualization, and creating interactive reports.

We'll use Python and the advanced capabilities of the Assistants API to create solutions tailored to the unique demands of cybersecurity. We'll also explore using the OpenAI Playground for a more interactive, GUI-based experience, and employing Python for deeper integration and automation.

By combining the intuitive interface of the Playground with the robust, programmable nature of Python, we're set to create assistants that aren't just reactive, but proactive in their capabilities. Whether you're automating routine tasks, analyzing complex datasets, or generating comprehensive cybersecurity reports, these new features are designed to enhance efficiency and effectiveness in your cybersecurity operations.

Getting ready

To effectively utilize OpenAI's new Assistants in the realm of cybersecurity, it's essential to prepare your environment and familiarize yourself with the required tools. This section lays the groundwork for a smooth experience in building advanced cybersecurity assistants.

- **OpenAI account and API key**. First and foremost, ensure you have an OpenAI account. If you haven't already, sign up at OpenAI's official website. Once your account is set up, obtain your API key, as it will be crucial for both Playground and Python-based interactions.
- **Familiarity with OpenAI Playground**. Navigate to OpenAI's Playground. Spend some time exploring its interface, focusing on the **Assistants** feature. This intuitive GUI is an excellent way to understand the capabilities of OpenAI's models before diving into code.

- **Python setup**. Ensure that Python is installed on your system. We will be using Python to interact programmatically with the OpenAI API. For a seamless experience, it's recommended to use Python 3.6 or later.
- **Required Python libraries**. Install the `openai` library, which facilitates communication with OpenAI's API. Use the command `pip install openai` in your command line or terminal.
- **Development environment**. Set up a comfortable coding environment. This could be a simple text editor and a command line, or an **integrated development environment** (**IDE**) like PyCharm or Visual Studio Code.
- **Basic Python knowledge**. While advanced Python skills are not a prerequisite, a basic understanding of Python programming will be beneficial. This includes familiarity with making API requests and handling JSON data.

How to do it…

To bring create a cybersecurity analyst assistant using OpenAI's API, let's break down the process into manageable steps that outline everything from setup to execution.

1. **Setup Up the OpenAI Client**. Begin by importing the OpenAI library (as well as the other needed libraries) and initializing the OpenAI client. This step is crucial for establishing communication with OpenAI's services.

```
import openai
from openai import OpenAI
import time
import os

client = OpenAI()
```

2. **Upload a Data File**. Prepare your data file, which the assistant will use to provide insights. Here, we're uploading a `"data.txt"` file. Ensure your file is in a readable format (like CSV or JSON) and contains relevant cybersecurity data.

```
 file = client.files.create(
  file=open("data.txt", "rb"),
  purpose='assistants'
)
```

3. **Create the Cybersecurity Analyst Assistant**. Define your assistant's role, name, and capabilities. In this case, we're creating a *Cybersecurity Analyst Assistant* that uses the GPT-4 model and has retrieval tools enabled, allowing it to pull information from the uploaded file.

```
security_analyst_assistant = client.beta.
  assistants.create(
    name="Cybersecurity Analyst Assistant",
    instructions="You are a cybersecurity analyst that
      can help identify potential security issues.",
    model="gpt-4-turbo-preview",
    tools=[{"type": "retrieval"}],
    file_ids=[file.id],
)
```

4. **Initiate a Thread and Starting a Conversation**. Threads are used to manage interactions with the assistant. Start a new thread and send a message to the assistant, prompting it to analyze the uploaded data for potential vulnerabilities.

```
 thread = client.beta.threads.create()
message = client.beta.threads.messages.create(
    thread.id,
    role="user",
    content="Analyze this system data file for potential
      vulnerabilities."
)
```

5. **Run the Thread and Fetching Responses**. Trigger the assistant to process the thread and wait for it to complete. Once done, retrieve the assistant's responses, filtering by the role of `'assistant'` to get the insights.

```
 run = client.beta.threads.runs.create(
  thread_id=thread.id,
  assistant_id=security_analyst_assistant.id,
)

def get_run_response(run_id, thread_id):
    while True:
        run_status = client.beta.threads.runs.
          retrieve(run_id=run_id, thread_id=thread_id)
        if run_status.status == "completed":
            break
        time.sleep(5)  # Wait for 5 seconds before
          checking the status again

    messages = client.beta.threads.messages.list
      (thread_id=thread_id)
    responses = [message for message in messages.data if
      message.role == "assistant"]
    values = []
    for response in responses:
        for content_item in response.content:
            if content_item.type == 'text':
                values.append(content_item.text.value)
    return values
values = get_run_response(run.id, thread.id)
```

6. **Print the Results**. Finally, iterate over the fetched values to review the assistant's analysis. This step is where the cybersecurity insights, such as identified vulnerabilities or recommendations, are presented.

```
   for value in values:
      print(value)
```

Here is how the final script should look:

```
import openai
from openai import OpenAI
import time
import os

# Set the OpenAI API key
api_key = os.environ.get('OPENAI_API_KEY')

# Initialize the OpenAI client
client = OpenAI()

# Upload a file to use for the assistant
file = client.files.create(
  file=open(«data.txt», «rb"),
  purpose=›assistants›
)

# Function to create a security analyst assistant
security_analyst_assistant = client.beta.assistants.create(
    name=»Cybersecurity Analyst Assistant»,
    instructions=»You are cybersecurity that can help identify
      potential security issues.",
    model=»gpt-4-turbo-preview»,
    tools=[{«type»: «retrieval»}],
    file_ids=[file.id],
)

thread = client.beta.threads.create()

# Start the thread
```

```
message = client.beta.threads.messages.create(
    thread.id,
    role=»user»,
    content=»Analyze this system data file for potential
      vulnerabilities."
)

message_id = message.id

# Run the thread
run = client.beta.threads.runs.create(
  thread_id=thread.id,
  assistant_id=security_analyst_assistant.id,
)

def get_run_response(run_id, thread_id):
    # Poll the run status in intervals until it is completed
    while True:
        run_status = client.beta.threads.runs.retrieve
          (run_id=run_id, thread_id=thread_id)
        if run_status.status == "completed":
            break
        time.sleep(5)  # Wait for 5 seconds before checking
          the status again

    # Once the run is completed, retrieve the messages from
      the thread
    messages = client.beta.threads.messages.list
      (thread_id=thread_id)

    # Filter the messages by the role of ‹assistant› to get
      the responses
    responses = [message for message in messages.data if
      message.role == "assistant"]

    # Extracting values from the responses
    values = []
    for response in responses:
        for content_item in response.content:  # Assuming
          'content' is directly accessible within 'response'
            if content_item.type == 'text':  # Assuming each
              'content_item' has a 'type' attribute
                values.append(content_item.text.value)
            # Assuming 'text' object contains 'value'

    return values

# Retrieve the values from the run responses
values = get_run_response(run.id, thread.id)

# Print the extracted values
for value in values:
    print(value)
```

Using these steps will give you the foundation for creating assistants using OpenAI's Assistants API.

How it works…

he process of creating and utilizing a cybersecurity analyst assistant via OpenAI's API involves a sophisticated interaction of various components. This section delves into the underlying mechanisms that make this possible, providing insights into the functionality and integration of these components.

- **Initialization and File Upload**. The process begins with initializing the OpenAI client, a crucial step that enables communication with OpenAI's services. Following this, a data file is uploaded, serving as a crucial resource for the assistant. This file, containing relevant cybersecurity information, is tagged for `'assistants'` use, ensuring it is appropriately categorized within OpenAI's ecosystem.
- **Assistant Creation**. A specialized assistant is then created with a specific focus on cybersecurity analysis. This assistant is not just any generic model; it is tailored with instructions that define its role as a cybersecurity analyst. This customization is pivotal, as it directs the assistant's focus towards identifying potential security issues.
- **Thread Management and User Interaction**. Threads are a core component of this process, acting as individual sessions of interaction with the assistant. A new thread is created for each query, ensuring a structured and organized dialogue. Within this thread, a user message initiates the assistant's task, prompting it to analyze the uploaded data for vulnerabilities.
- **Active Analysis and Run Execution**. The *Run* represents the active phase of analysis, where the assistant processes the information within the thread. This phase is dynamic, with the assistant actively engaged in deciphering the data, guided by its underlying model and the instructions provided.
- **Response Retrieval and Analysis**. Once the run is complete, the focus shifts to retrieving and analyzing the assistant's responses. This step is critical, as it involves filtering through the messages to extract the assistant's insights, which are based on its analysis of the cybersecurity data.

- **Tool Integration**. The assistant's capabilities are further enhanced by integrating tools such as the Code Interpreter. This integration allows the assistant to perform more complex tasks, such as executing Python code, which can be particularly useful for automating security checks or parsing threat data.
- **Comprehensive Workflow**. The culmination of these steps forms a comprehensive workflow that transforms a simple query into a detailed cybersecurity analysis. This workflow encapsulates the essence of leveraging AI in cybersecurity, demonstrating how structured data, when analyzed by a specialized assistant, can yield critical insights into potential vulnerabilities.

This intricate process showcases the power of OpenAI's API in creating specialized assistants that can significantly augment cybersecurity operations. By understanding the underlying mechanisms, users can effectively leverage this technology to enhance their cybersecurity posture, making informed decisions based on the assistant's analysis.

There's more...

The Assistants API offers a rich set of features that extend far beyond the basic implementation covered in the recipe. These capabilities allow for the creation of more complex, interactive, and versatile assistants. Here's a detailed look at some of the API features that weren't covered in the initial recipe, complete with code references to illustrate their implementation:

- **Streaming Output and Run Steps**. Future enhancements may introduce streaming outputs for real-time interaction and detailed Run Steps for a granular view of the assistant's processing stages. This could be particularly useful for debugging and optimizing the assistant's performance.

```
 # Potential future code for streaming output
stream = client.beta.streams.create
  (assistant_id=security_analyst_assistant.id, ...)
for message in stream.messages():
    print(message.content)
```

- **Notifications for Status Updates**. The ability to receive notifications for object status updates could eliminate the need for polling, making the system more efficient.

```
 # Hypothetical implementation for receiving
   notifications
client.notifications.subscribe(object_id=run.id,   event_
type='status_change', callback=my_callback_function)
```

- **Integration with DALL·E or Browsing Tools**. Integrating with DALL·E for image generation or adding browsing capabilities could significantly expand the assistant's functionalities.

```
 # Example code for integrating DALL·E
response = client.dalle.generate(prompt="Visualize
  network security architecture",
    assistant_id=security_analyst_assistant.id)
```

- **User Message Creation with Images**. Allowing users to include images in their messages could enhance the assistant's understanding and response accuracy in visually dependent tasks.

  ```
   # Example code for sending an image in a user message
  message = client.beta.threads.messages.create(thread.id,
    role="user", content="Analyze this network diagram.",
      file_ids=[uploaded_image_file.id])
  ```

- **Code Interpreter Tool**. The Code Interpreter tool enables the assistant to write and execute Python code, offering a powerful way to automate tasks and perform complex analyses.

  ```
   # Enabling Code Interpreter in an assistant
  assistant = client.beta.assistants.create(
      name="Data Analysis Assistant",
      instructions="Analyze data and provide insights.",
      model="gpt-4-turbo-preview",
      tools=[{"type": "code_interpreter"}]
  )
  ```

- **Code Interpreter Tool**. This tool allows the assistant to pull information from uploaded files or databases, enriching its responses with external data.

  ```
   # Using Knowledge Retrieval to access uploaded files
  file = client.files.create(file=open("data_analysis.pdf",
    "rb"), purpose='knowledge-retrieval')
  assistant = client.beta.assistants.create(
      name="Research Assistant",
      instructions="Provide detailed answers based on the
        research data.",
      model="gpt-4-turbo-preview",
      tools=[{"type": "knowledge_retrieval"}],
      file_ids=[file.id]
  )
  ```

- **Custom Tool Development**. Beyond the provided tools, you can develop custom tools using Function calling, tailoring the assistant's capabilities to specific needs.

  ```
   # Example for custom tool development
  def my_custom_tool(assistant_id, input_data):
      # Custom tool logic here
      return processed_data

  # Integration with the assistant
  assistant = client.beta.assistants.create(
      name="Custom Tool Assistant",
      instructions="Use the custom tool to process data.",
  ```

```
    model="gpt-4-turbo-preview",
    tools=[{"type": "custom_tool", "function":
      my_custom_tool}]
)
```

- **Persistent Threads and Advanced File Handling**. Assistants can manage persistent threads, maintaining a history of interactions, and handle files in various formats, supporting complex data processing tasks.

```
 # Creating a persistent thread and handling files
thread = client.beta.threads.create(persistent=True)
file = client.files.create(file=open("report.docx",
  "rb"), purpose='data-analysis')
message = client.beta.threads.messages.create(thread.id,
  role="user", content="Analyze this report.",
    file_ids=[file.id])
```

- **Safety and Privacy Considerations**. OpenAI's commitment to data privacy and security ensures that sensitive information is handled with care, making the Assistants API suitable for applications involving confidential data.

```
 # Example of privacy-focused assistant creation
assistant = client.beta.assistants.create(
    name="Privacy-Focused Assistant",
    instructions="Handle user data securely.",
    model="gpt-4-turbo-preview",
    privacy_mode=True
)
```

These examples illustrate the breadth and depth of functionalities offered by the Assistants API, highlighting its potential to create highly specialized and powerful AI assistants. Whether it's through real-time interaction, enhanced data processing capabilities, or custom tool integration, the API provides a versatile platform for developing advanced AI solutions tailored to a wide range of applications.

More comprehensive information about the OpenAI assistants API can be found at: `https://platform.openai.com/docs/assistants/overview` and `https://platform.openai.com/docs/api-reference/assistants`

Index

Other Books You May Enjoy

Share Your Thoughts

Now you've finished *ChatGPT for Cybersecurity Cookbook*, we'd love to hear your thoughts! If you purchased the book from Amazon, please click here to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.