1

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

ChatGPT is a large langauge 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 tasks.

**Note**

Generative AI is a branch of artificial intelligence that uses machine learning algorithms and natural language processing (NLP) to analyze patterns and structures within a dataset, and then generates new data that resemble 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 large language models goes far beyond simple autocomplete.  
  
Language language models (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 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 its deep knowledgebase and its ability to perform complex tasks like 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.

Hint

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 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, 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

Setting up a ChatGPT Account

In this recipe, we will first learn about generative AI, large language models (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.

Note

Every effort is made to ensure that every illustration and instruction is correct at the time of this 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, it’s possible that you might find slight differences.

How to do it…

1. Visit the OpenAI website at <https://platform.openai.com/> and click “Sign up”
2. Enter your email address and click continue, or you can register with your existing Google or Microsft account.

Graphical user interface, application

Description automatically generated

Figure 1.1 – OpenAI Signup Form

1. Enter a strong password and click continue.
2. Check your email for a verification message from OpenAI. Click the link provided in the email to verify your account.
3. Once your account is verified, enter the required information (first name, last name, optional organization name, and birthday) and click continue.
4. Enter your phone number to verify by phone and click send code.
5. When you receive the text message with the code, enter the code and click continue.
6. 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 the 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 an OpenAI free 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 the 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 this 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 own 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.

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 Recipe 1.1 to set up your ChatGPT account.

How to do it…

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 dropdown menu.

Table

Description automatically generated

Figure 1.2.1 – API keys screen

1. Click the "Create new secret key" button to generate a new API key.
2. Give your API key a name (optional) and click "Create secret key".

Graphical user interface, text, application, chat or text message

Description automatically generated

Icon

Description automatically generatedFigure 1.2.2 – Naming your API key

1. Your new API key will be displayed on the screen. Click the copy icon to copy the key to your clipboard.

Tip

Be sure to save your API key in a secure location immediately, as you will need it later when working with the OpenAI API and you cannot view the key again in its entirety once it is saved.

Graphical user interface, text, application, email

Description automatically generated

Figure 1.2.3 – 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:

1. Log in to your OpenAI account.
2. Click on "Playground" in the top navigation bar.

Graphical user interface, text, application

Description automatically generated

Figure 1.2.4 – The OpenAI Playground

1. In the Playground, you can choose from various different models by selecting the model you want to use from the "Model" dropdown menu.

Graphical user interface, application

Description automatically generated

Figure 1.2.5 – Selecting a model

1. Enter your prompt in the text box provided and click "Submit" to see the model's response.

Graphical user interface, text, application

Description automatically generated

Figure 1.2.6 – 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 towards your account’s token/credit usage.

1. 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.

Graphical user interface, text, application, email

Description automatically generated

Figure 1.2.7 – Adjusting settings in the Playground

Two of the most important parimeters are “Temperature” and “Maximum length”.

* The temperature parameter affects the randomness and creativity of the model's responses. A higher temperature (e.g., 0.8) will produce more diverse and creative outputs, while a lower temperature (e.g., 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.

Graphical user interface, text, application

Description automatically generated Graphical user interface, application

Description automatically generated

Figure 1.2.8 – Prompt presets Figure 1.2.9 – 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 Recipe 1.1 to set up your ChatGPT account.

How to do it…

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.

Graphical user interface, table

Description automatically generated

Figure 1.3.1 – The ChatGPT interface

1. Shape

   Description automatically generated with medium confidenceChatGPT 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:

Graphical user interface, text, application, email

Description automatically generated

Figure 1.3.2 – Entering a prompt

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

Graphical user interface, text, application

Description automatically generated

Figure 1.3.3 – ChatGPT response with code

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

Graphical user interface, text

Description automatically generated

Figure 1.3.4 – ChatGPT contextual follow-up response

1. Run the ChatGPT generated code by clicking on “Copy code”, paste it into your code editor of choice (I personally use Visual Studio Code), save it as a “.py” Python script, and run from a terminal.

PS D:\GPT\ChatGPT for Cybersecurity Cookbook> python .\my\_ip.py

Your public IP address is:

Your local network IP address is: 192.168.1.105

Figure 1.3.5 – 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 continue over the course of an entire conversation like 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 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.

Graphical user interface, application

Description automatically generated

Figure 1.3.6 – 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.

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 to 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…

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.

1. 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.

Text, letter

Description automatically generated

Figure 1.4.1 – ChatGPT response with role-based expertise

1. Experiment with assigning different roles for different scenarios, such as:

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

or

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

Caution

Keep in mind that ChatGPT's knowledge is based on the data it was trained on, which has a cutoff date 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?

1. 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…

1. When crafting your prompt, you can specify the output 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\*

…

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

Text, letter

Description automatically generated

Figure 1.5.1 – ChatGPT response with formatting (headings, bold font, lists)

Text, application

Description automatically generated with medium confidence

Figure 1.5.2 – ChatGPT response with formatting (heading, lists, italicized text)

1. 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 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 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 lists 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. In this case 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 in to your OpenAI account.

How to do it…

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.

1. ChatGPT will generate a response containing a table with the specified columns and populated with relevant information.

Table

Description automatically generated

Figure 1.6.1 – Snippet of a ChatGPT response with a table

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

Calendar

Description automatically generated

Figure 1.6.2 – ChatGPT response copy/paste 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 Recipe 1.2.

How to do it…

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

For Windows:

* 1. Open the Start menu, search for "Environment Variables," and click on "Edit the system environment variables."
  2. In the "System Properties" window, click on the "Environment Variables" button.
  3. In the "Environment Variables" window, click on "New" under "User variables" or "System variables" (depending on your preference).
  4. Enter "OPENAI\_API\_KEY" as the variable name and paste your API key as the variable value. Click "OK" to save the new environment variable.

**For MacOS/Linux:**

1. Open a terminal window.
2. 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 (e.g., ~/.zshrc, ~/.profile).

1. Restart the 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"]

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 sharing code with others or publishing it in public repositories.

There's more…

In some cases, you may want to use a Python package like 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

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

1. Start by importing the required modules.

import openai

import os

1. Set up your API key by retrieving it from an environment variable, like we setup in Recipe 1.7.

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

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

def get\_chat\_gpt\_response(prompt):

response = openai.Completion.create(

engine="text-davinci-003",

prompt=prompt,

max\_tokens=150,

n=1,

stop=None,

temperature=0.5,

)

return response.choices[0].text.strip()

1. 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. 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: We specify the engine (in this case, "text-davinci-003").
* prompt: The input text for the model to generate a response based on.
* 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 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 (e.g., 1.0) will result in more random responses, while a lower temperature (e.g., 0.1) will make the responses more focused and deterministic.

1. 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].text.strip() line of code retrieves the generated response text by accessing the first choice (index 0) in the list of choices.

The 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.

For example, a non-formatted response from the OpenAI API may look like:

{

'id': 'example\_id',

'object': 'text.completion',

'created': 1234567890,

'model': 'text-davinci-002',

'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, as 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, we 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, 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 Recipes 1.2 and 1.7.

How to do it…

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 to include 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()

1. Using the script from Recipe 1.8, 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")

1. Create a file called prompt.txt and enter the following prompt text (same prompt as Recipe 1.8):

Explain the difference between symmetric and asymmetric encryption.

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

openai.api\_key = open\_file('openai-key.txt')

1. Create a file called openai-key.txt and past your OpenAI API key into the file with nothing else.
2. Use the prompt variable in your API call as you normally would.

Here is an example of how the modified Recipe 1.8 script would look:

import openai

import os

openai.api\_key = open\_file('openai-key.txt')

def open\_file(filepath):

with open(filepath, 'r', encoding='UTF-8') as infile:

return infile.read()

def get\_chat\_gpt\_response(prompt):

response = openai.Completion.create(

engine="text-davinci-002",

prompt=prompt,

max\_tokens=150,

n=1,

stop=None,

temperature=0.5,

)

return response.choices[0].text.strip()

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 the both without needing to modify the main script or environment variables. This can be particularly helpful when working with multiple prompts or collaborating with others.

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 precations. It is also important to remember to remove your own API key from the openapi-key.txt file before sharing 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 create a Linux style manual page generator, which will accept user input in the form of tool’s name, and our script will generate the manual page output just as entering the man command in a 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…

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

import openai

import os

1. 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()

1. Set up your API key:

openai.api\_key = open\_file('openai-key.txt')

1. Create the openai-key.txt file in the same manner as the previous recipe.
2. Define the get\_chat\_gpt\_response() function to send the prompt to ChatGPT and obtain a response:

def get\_chat\_gpt\_response(prompt):

response = openai.Completion.create(

model="text-davinci-003",

prompt=prompt,

temperature=0.7,

max\_tokens=600,

top\_p=1,

frequency\_penalty=0,

presence\_penalty=0

)

text = response['choices'][0]['text'].strip()

return text

1. Receive user input for the file name and read the content of the file:

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

feed = open\_file(file)

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

prompt = open\_file("prompt.txt").replace('<<INPUT>>', feed)

1. 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>>

1. Send the modified prompt to the get\_chat\_gpt\_response() function and print the result:

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 <<INPUT>> that can be replaced with any content, allowing for dynamic modification of the prompt without changing 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 file name and reads its content using the open\_file() function. 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 literally has access to just about every manual page entry found on the internet. By instructing it to provide nothing before or after the Linux specific output, ChatGPT will not provide any additiona 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, 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. Using 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, like IP addresses, domain names, and user agents. Just make sure to replace all the necessary variables before sending the prompt to ChatGPT.
2. Customizing 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 like {input} or any other format that you find more readable and manageable.
3. Using environment variables for sensitive data: When working with sensitive data like 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, like 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.