Hands-on Lab: Attack Pattern Analysis Using Generative AI

Estimated time needed: 30 minutes

Introduction

Welcome to the hands-on lab, Attack Pattern Analysis Using Generative AI.

Generative AI is used for various jobs in cybersecurity. In this lab, you will explore how to use it for attack pattern analysis of a programming code, which is one of the most crucial activities in cybersecurity.

You will use a generative AI model to analyze the attack pattern of a Python code. Attack pattern analysis in code refers to the process of identifying and understanding specific patterns or techniques used by attackers in malicious code or software. This analysis aims to recognize the methods employed by attackers to compromise systems, exploit vulnerabilities, or achieve their malicious objectives. Generative AI plays a crucial role in cybersecurity to enhance threat intelligence, incident response, and overall security posture.

Learning objective

After completing this lab, you will be able to:

- Analyze code to detect and understand malicious behavior
- Identify vulnerable assets at risk of infection
- Describe effective methods to assess the success of attacks and system compromises

Attack pattern analysis of a malicious code

In this exercise, you will test the capability of the generative AI model to analyze the malicious Python program and identify the attack pattern.

Scenario: Assume that you have identified a suspicious file in your system, and after reverse engineering, you have retrieved the following Python code. Now, you want to know the malicious behavior of the program using a generative AI model.

```
import time
import daemonize
import pygetwindow as gw
import keyboard
# Function to record URL to the 'history' file
def record_url(url):
    with open('history.txt', 'a') as history_file:
        history_file.write(f"{url}\n")
# Main function to monitor browser activity and record URLs
def main():
    while True:
        try:
            # Check if the browser window is active
            active_window = gw.getActiveWindow()
            if active_window and "browser" in active_window.title.lower():
                # Assuming 'Ctrl + L' is used to focus on the browser address bar
                keyboard.press_and_release('ctrl + l')
                time.sleep(0.5)  # Allow time for the address bar to be in the focus
                keyboard.press_and_release('ctrl + c')  # Copy the URL from the address bar
                url_to_record = keyboard.read_event().name  # Read the clipboard content
                # Record the URL
                record_url(url_to_record)
        except Exception as e:
            # Handle exceptions (missing libraries, window title not found)
            print(f"Error: {e}")
        # Sleep for a while before checking again
        time.sleep(2)
if __name__ == "__main__":
    # Define the paths for the daemon process
    pid = "url_recorder.pid"
    stdout = "url_recorder.log"
    stderr = "url_recorder_error.log"
    # Create the daemon context
    daemon = daemonize.Daemonize(app="url_recorder", pid=pid, action=main, keep_fds=[1, 2])
    # Start the daemon
    daemon.start()
```

Note: These examples are generic and do not contain real-world data. Detailed investigation, correlation with other logs, and additional context would be required in a real scenario to confirm and respond to such alerts appropriately.

Now, with the help of generative AI, proceed with the following tasks:

1. Analyze the code and identify the malicious behavior.
2. List the potential assets susceptible to infection.
3. Determine the method for verifying the success of the attack and the compromise of the system.

Exercise 1: Attack pattern analysis of the program code

Step 1: Copy and paste the prompt instructions in the message input box.

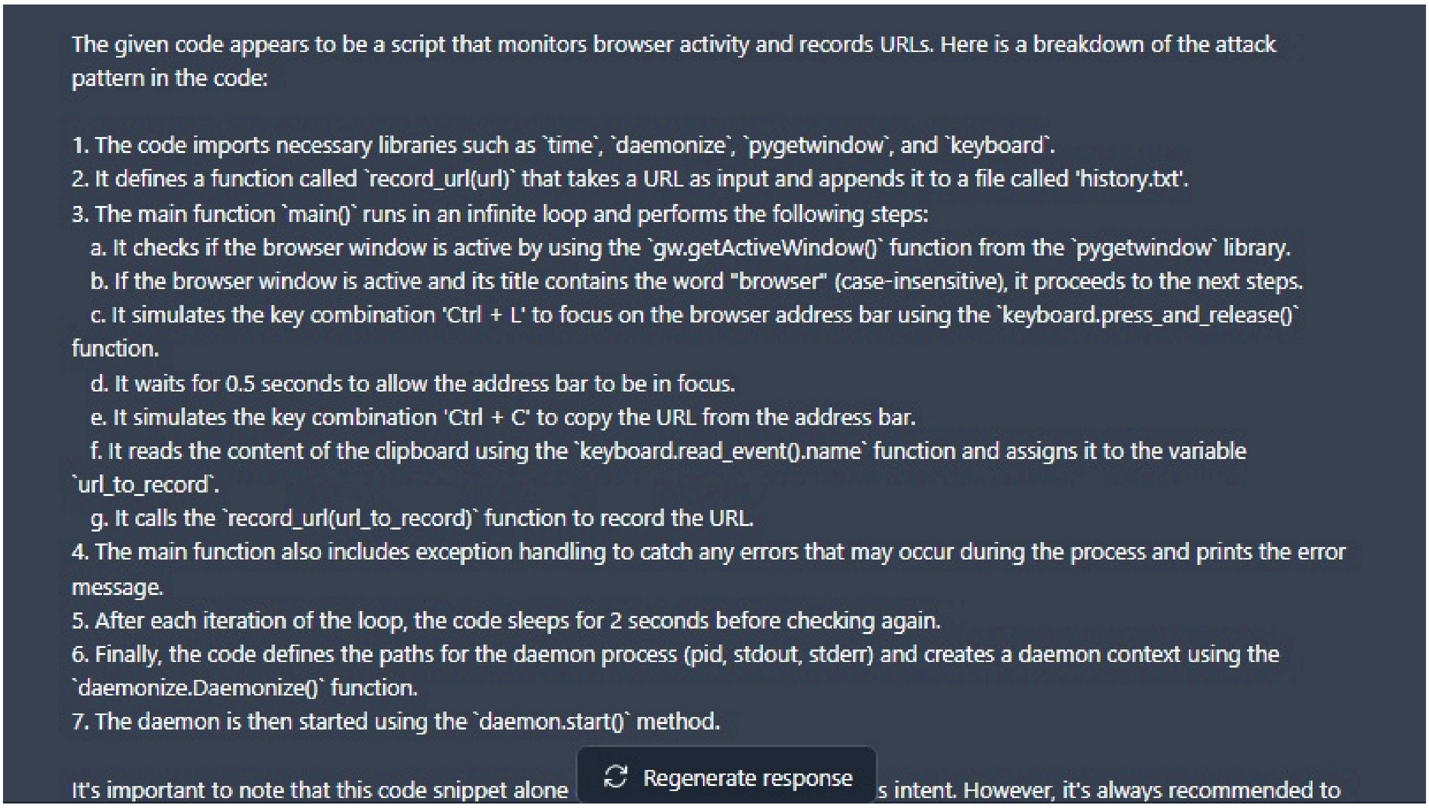
```
Analyze the attack pattern of the code
```

Step 2: Copy and paste the following code within double quotation marks in the mesaage input box.

```
import time
import daemonize
import pygetwindow as gw
import keyboard
# Function to record URL to the 'history' file
def record_url(url):
    with open('history.txt', 'a') as history_file:
        history_file.write(f"{url}\n")
# Main function to monitor browser activity and record URLs
def main():
    while True:
        try:
            # Check if the browser window is active
            active_window = gw.getActiveWindow()
            if active_window and "browser" in active_window.title.lower():
                # Assuming 'Ctrl + L' is used to focus on the browser address bar
                keyboard.press_and_release('ctrl + l')
                time.sleep(0.5)  # Allow time for the address bar to be in focus
                keyboard.press_and_release('ctrl + c')  # Copy the URL from the address bar
                url_to_record = keyboard.read_event().name  # Read the clipboard content
                # Record the URL
                record_url(url_to_record)
        except Exception as e:
            # Handle exceptions (e.g., missing libraries, window title not found)
            print(f"Error: {e}")
        # Sleep for a while before checking again
        time.sleep(2)
if __name__ == "__main__":
    # Define the paths for the daemon process
    pid = "url_recorder.pid"
    stdout = "url_recorder.log"
    stderr = "url_recorder_error.log"
    # Create the daemon context
    daemon = daemonize.Daemonize(app="url_recorder", pid=pid, action=main, keep_fds=[1, 2])
    # Start the daemon
    daemon.start()
```

Step 3: Observe the response which shows the breakdown of the code with a detail analysis.

Sample output



The given code appears to be a script that monitors browser activity and records URLs. Here is a breakdown of the attack pattern in the code:

1. The code imports necessary libraries such as `time`, `daemonize`, `pygetwindow`, and `keyboard`.
2. It defines a function called `record_url(url)` that takes a URL as input and appends it to a file called 'history.txt'.
3. The main function `main()` runs in an infinite loop and performs the following steps:
   a. It checks if the browser window is active by using the `gw.getActiveWindow()` function from the `pygetwindow` library.
   b. If the browser window is active and its title contains the word "browser" (case-insensitive), it proceeds to the next steps.
   c. It simulates the key combination 'Ctrl + L' to focus on the browser address bar using the `keyboard.press_and_release()` function.
   d. It waits for 0.5 seconds to allow the address bar to be in focus.
   e. It simulates the key combination 'Ctrl + C' to copy the URL from the address bar.
   f. It reads the content of the clipboard using the `keyboard.read_event().name` function and assigns it to the variable `url_to_record`.
   g. It calls the `record_url(url_to_record)` function to record the URL.
4. The main function also includes exception handling to catch any errors that may occur during the process and prints the error message.
5. After each iteration of the loop, the code sleeps for 2 seconds before checking again.
6. Finally, the code defines the paths for the daemon process (pid, stdout, stderr) and creates a daemon context using the `daemonize.Daemonize()` function.
7. The daemon is then started using the `daemon.start()` method.

It's important to note that this code snippet alone s intent. However, it's always recommended to

Note: The response generated by the generative AI platform may vary as it is dynamic in behavior, and you may need to tweak your prompt instructions to get a better response.

Exercise 2: Listing the potential asssets susceptible to infection

Upon analyzing the code, it is crucial to identify the assets potentially infected by the malicious code. You will use a generative AI model for thorough scrutiny and subsequent analysis.

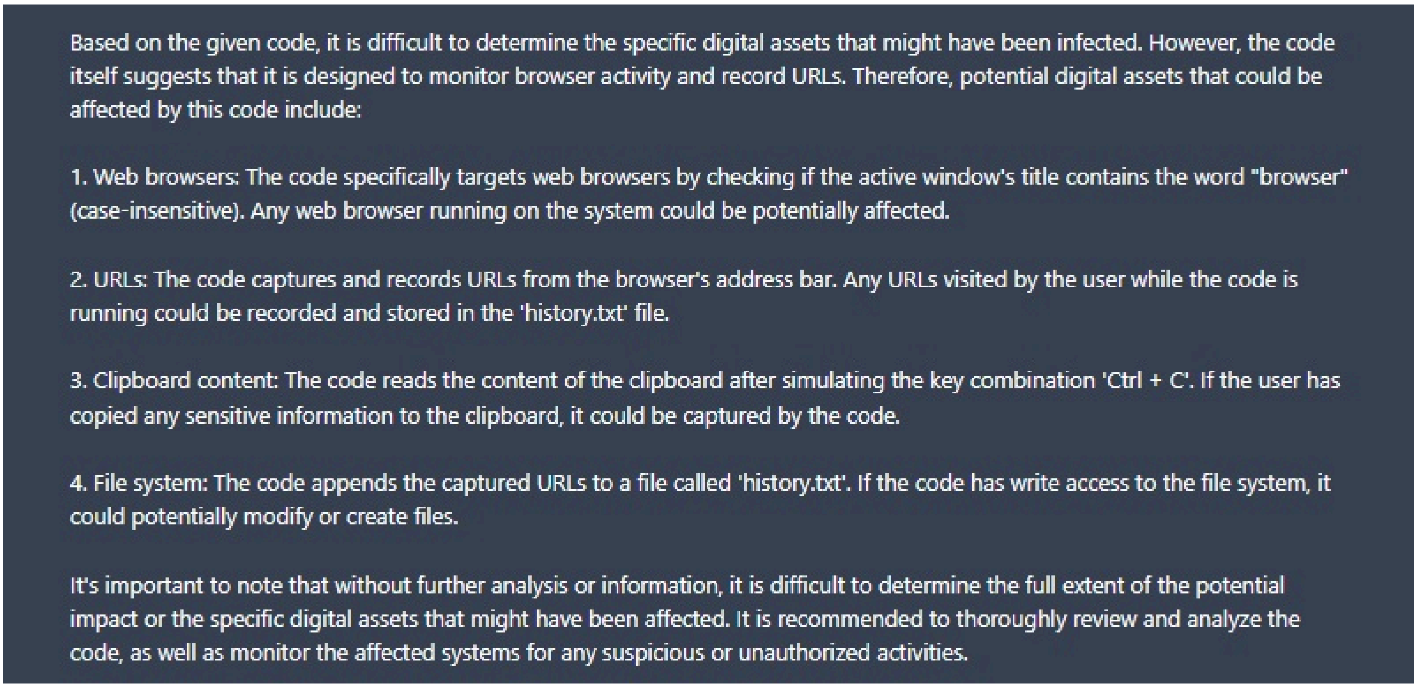
Let's follow the steps:

Step 1: Copy and paste the following prompt in the message input box.

```
Identify the list of potential digital assets might have been infected by this malicious code
```

Step 2: Review the response. The response mentions potential assets that might have been infected.

Sample output


Based on the given code, it is difficult to determine the specific digital assets that might have been infected. However, the code itself suggests that it is designed to monitor browser activity and record URLs. Therefore, potential digital assets that could be affected by this code include:

1. Web browsers: The code specifically targets web browsers by checking if the active window's title contains the word "browser" (case-insensitive). Any web browser running on the system could be potentially affected.

2. URLs: The code captures and records URLs from the browser's address bar. Any URLs visited by the user while the code is running could be recorded and stored in the 'history.txt' file.

3. Clipboard content: The code reads the content of the clipboard after simulating the key combination 'Ctrl + C'. If the user has copied any sensitive information to the clipboard, it could be captured by the code.

4. File system: The code appends the captured URLs to a file called 'history.txt'. If the code has write access to the file system, it could potentially modify or create files.

It's important to note that without further analysis or information, it is difficult to determine the full extent of the potential impact or the specific digital assets that might have been affected. It is recommended to thoroughly review and analyze the code, as well as monitor the affected systems for any suspicious or unauthorized activities.

Exercise 3: Confirm the success of the attack and system compromise

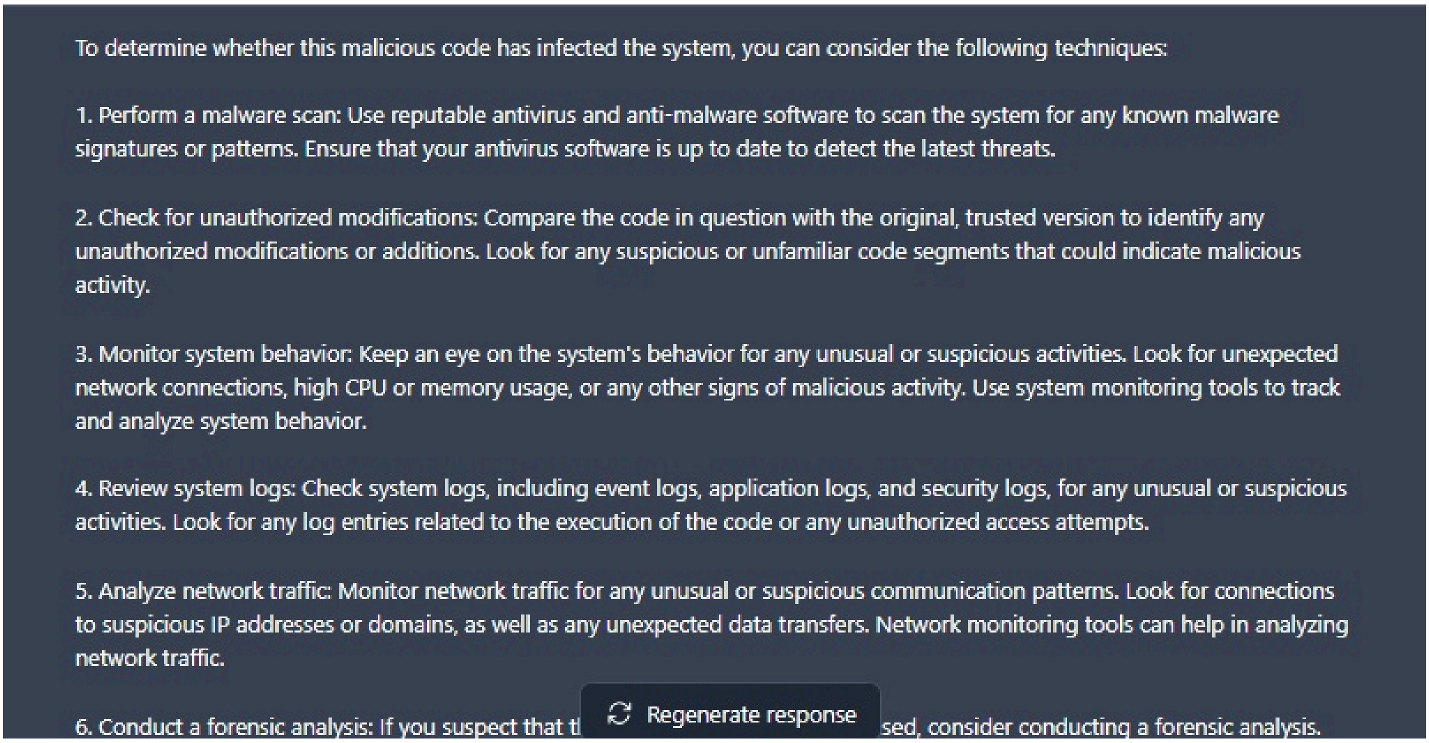
In this exercise, you will search for how the malicious code infected the system.

Step 1: Copy and paste the following prompt in the message input box.

```
Suggest techniques that will aid in determining whether the code has infected the system.
```

Step 2: Review the response.

Sample output


To determine whether this malicious code has infected the system, you can consider the following techniques:

1. Perform a malware scan: Use reputable antivirus and anti-malware software to scan the system for any known malware signatures or patterns. Ensure that your antivirus software is up to date to detect the latest threats.

2. Check for unauthorized modifications: Compare the code in question with the original, trusted version to identify any unauthorized modifications or additions. Look for any suspicious or unfamiliar code segments that could indicate malicious activity.

3. Monitor system behavior: Keep an eye on the system's behavior for any unusual or suspicious activities. Look for unexpected network connections, high CPU or memory usage, or any other signs of malicious activity. Use system monitoring tools to track and analyze system behavior.

4. Review system logs: Check system logs, including event logs, application logs, and security logs, for any unusual or suspicious activities. Look for any log entries related to the execution of the code or any unauthorized access attempts.

5. Analyze network traffic: Monitor network traffic for any unusual or suspicious communication patterns. Look for connections to suspicious IP addresses or domains, as well as any unexpected data transfers. Network monitoring tools can help in analyzing network traffic.

6. Conduct a forensic analysis: If you suspect that t sed, consider conducting a forensic analysis.

Exercises

Do the following exercise based on the Program Code used in the hands-on exercise:

1. Identify the techniques to disinfect the system if it is infected by malicious code.

▶ Click here for an example solution

2. List preventive measures.

▶ Click here for an example solution

Summary

Congratulations on completing the hands-on lab.

- Generative AI model can be employed for attack pattern analysis of a code by leveraging its natural language understanding capabilities.
- By presenting code snippets and related contextual information, generative AI can see insights into potential security vulnerabilities, threat modeling, and mitigation strategies.

Author(s)

Manish Kumar

© IBM Corporation. All rights reserved.