Charger Active Defense v1.0 Team 2 - Group 12

Noah Sickels, Adam Brannon, William Lochte

## Background / Abstract

Modern attack tools are highly efficient, while the cybersecurity industry struggles with active defense capabilities. Current passive defenses primarily focus on mitigating threats through means that can disrupt business operations or can be circumvented. Additionally, with the rapid advancement of artificial intelligence and prompt engineering, attackers can instantly generate various attack tools to target organizations.

The Charger Active Defense project focuses on developing a network-based fuzzing workflow to effectively and comprehensively test these known and AI/LLM-generated attack tools. It aims to identify responses that may cause the attacking application to crash or hang. Responses generated could be saved and sent back to the adversary through a Python replay service during a detected attack. Due to the multi-threading nature of many attack tools like Masscan and Medusa, effectively fuzzing these tools is complicated. If a fuzzed response leads to a crash or hang, it typically occurs in a thread separate from the attacking application's instance. This crashed thread may either be reported as a false positive or not counted at all. Because of this discovery, at the customer's request, we are pivoting towards thoroughly fuzzing generated attack tools from large language models.

Since we have successfully applied the fuzzing workflow to Masscan, we will also pursue the integration of ThreadSanitizer (TSan) to determine if current fuzz testing is properly crashing threads. The end goal of this workflow is to assess how we can broaden this method to apply to other attack tools, providing organizations with a strategy to defend their systems against an ever-evolving threat landscape.

## Current Project Status, Issues, & Short-Term Activities & Goals

This reporting period focused on extending our existing Active Defense Python service and generating our first attack tool: a port scanner and banner grabber, utilizing the GitHub Copilot and Phind AI models. For the Active Defense Python service, we added the ability to live capture the network traffic from a provided network interface, display the incoming packets to the UI, and the ability to craft packets with a specified port, protocol, IP address, and payload. For the generated attack tools, we prompted both AI models for the desired program and adjusted them as needed to ensure they were compiled successfully.

Our short-term goals for this period were:

* Implement the active defense response capability into the existing Python service.
* Generate a port-scanning, banner-grabber attack tool using the Phind model.
* Generate a port-scanning, banner-grabbing attack tool using the GitHub Copilot model.
* Ensured that the port scanner from Phind and Copilot could successfully compile with gcc.

During this period, we encountered a few issues:

* In order to use GitHub Copilot for personal use, we had to request a few educational licenses. Although this was done the week before, it took a couple of days before they got back to us and approved us. Fortunately, we were able to get approved in time to complete the necessary objectives for this period.
* One of the Python libraries we have to work with for the program's live network interface and active defense response functionality is Asyncio, an asynchronous input/output library. This library is incredibly complex and foreign to us and was a significant impediment throughout this period. The main problem we kept running into was that the UI library prompt\_toolkit, Asyncio, and the network library Pyshark that we are using all natively use the Asyncio library in them. This caused complications with the event loop, as many of the function calls within the Pyshark library require asynchronous calls, which meant that the calling class and functions need to have an event loop already started. We managed to get around this by creating an event loop at the start of the program and passing that same event loop through the subsequent classes.
* Despite fixing the initial issues with the main thread in the Asyncio library, we ran into a related issue with the packet capture functionality. When calling the `sniff\_continuously` method to capture the packets on the network interface, once the capture is interrupted by a keyboard interruption (CTRL+C) or after a specified number of packets, the program immediately throws an end-of-file exception and crashes. We eventually narrowed down the issue to a specific number of packets that would cause an end-of-file error on the Pyshark library interface class. We were able to solve this by limiting the number of packets to any parameter lower than 100, as any value after caused the exception. This may need to be reviewed later, but we were unable to explore any further remediations for it during this period.
* The port scanner/banner grabber generated by GitHub Copilot occasionally hangs during normal operations. As of now, it successfully compiles, and we will look at improving its functionality using Copilot in the next period.

For the next reporting period, our short-term goals are:

| Responsibility | Task |
| --- | --- |
| Noah Sickels | Apply fuzzing workflow to 1st set of Gen-AI attack tools.  Improve 1st generated banner grabber tool from GitHub Copilot for better functionality. |
| Adam Brannon | Perform LDRA static analysis on 1st set of Gen-AI attack tools. |
| William Lochte | Perform Valgrind dynamic memory leak testing on 1st set of Gen-AI attack tools. |

## Milestone Status Summary

| Milestone / Task | Projected Due Date | Completion Date | Status | Notes |
| --- | --- | --- | --- | --- |
| Develop Gen-AI Tool #1 (Phind model) | 2/14 | 2/14 | Complete | We may generate another program or ask Phind to improve upon the first version. |
| Develop Gen-AI Tool #2 (Copilot model) | 2/14 | 2/14 | Complete | We may generate another program or ask Copilot to improve upon the first version. |
| Python Service Response Module | 2/14 | 2/13 | Complete | Python service can properly craft a packet on the specified port, protocol, and IP address using the data provided to it (active defense responses). |

## Level of Effort / Individual Responsibility Record

| Member | Hours / Period | Total Hours / Spring |
| --- | --- | --- |
| Noah Sickels | 23 | 66 |
| Adam Brannon | 8 | 31 |
| William Lochte | 5 | 32 |
| Total | 36 | 129 |

| Member | Individual Accomplishments |
| --- | --- |
| Noah Sickels | Wrote active defense response handler functionality for Python service.  Wrote two network handling classes using the Socket library and the Pyshark library for the AD response handler and network traffic sending/receiving components. |
| Adam Brannon | Used Phind to generate a port scanner/banner grabber. Ensured that it compiled successfully without error and that it had the desired functionality.  Started to set up OnlyOffice for IEEE/ACM paper. |
| William Lochte | Used Github Copilot to generate a port scanner/banner grabber. Ensured that it compiled without error and that it had the required functionality. |

## Milestone Completion & Analysis

* The Active Defense Python service can now successfully craft packets and send them to a specified address using the provided data – in our case, the found crashes or hangs - as the payload.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 1: Active Defense Response Module

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 2: Test AD Response Packet

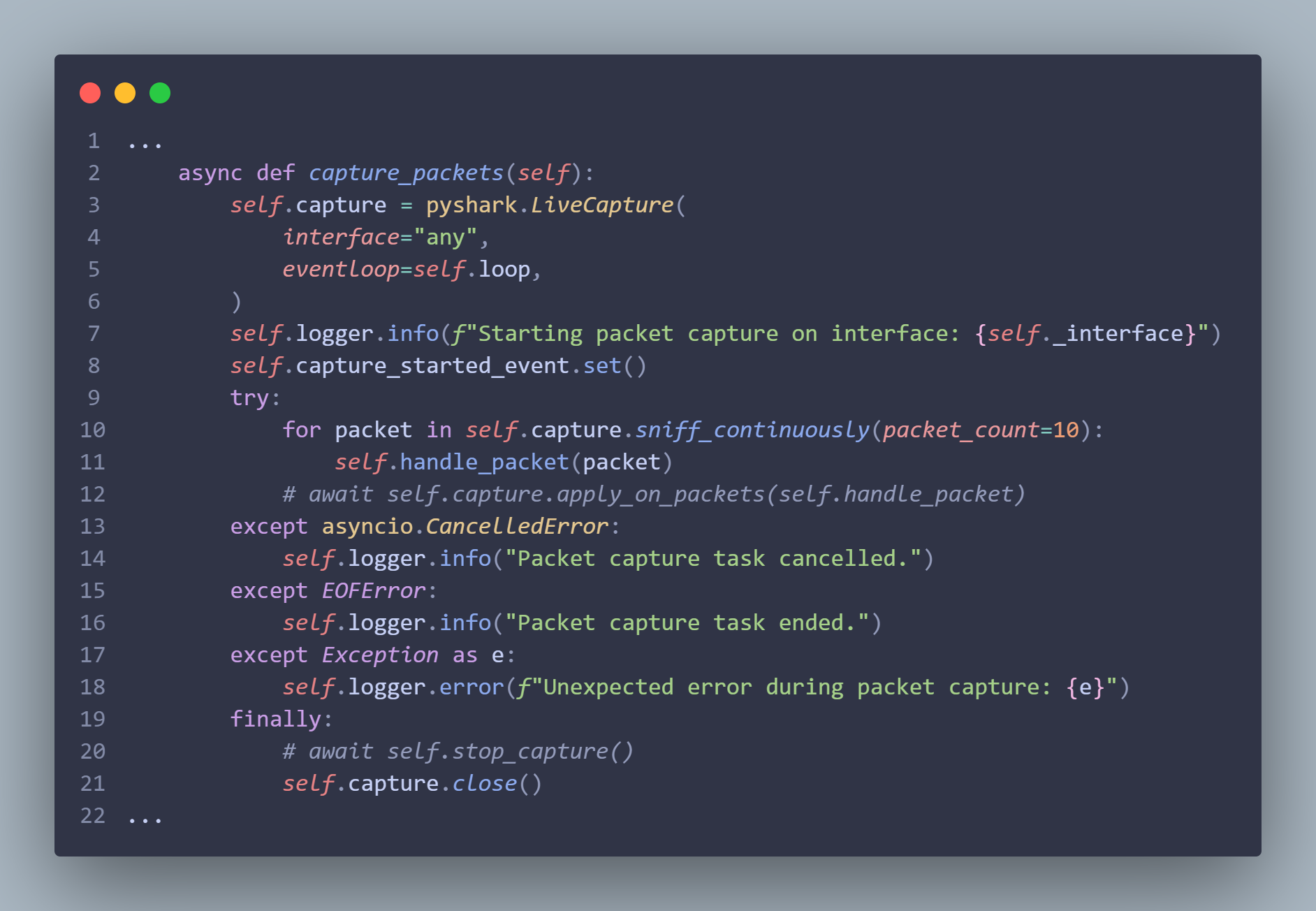


Figure 3: Capture Packets Code Snippet

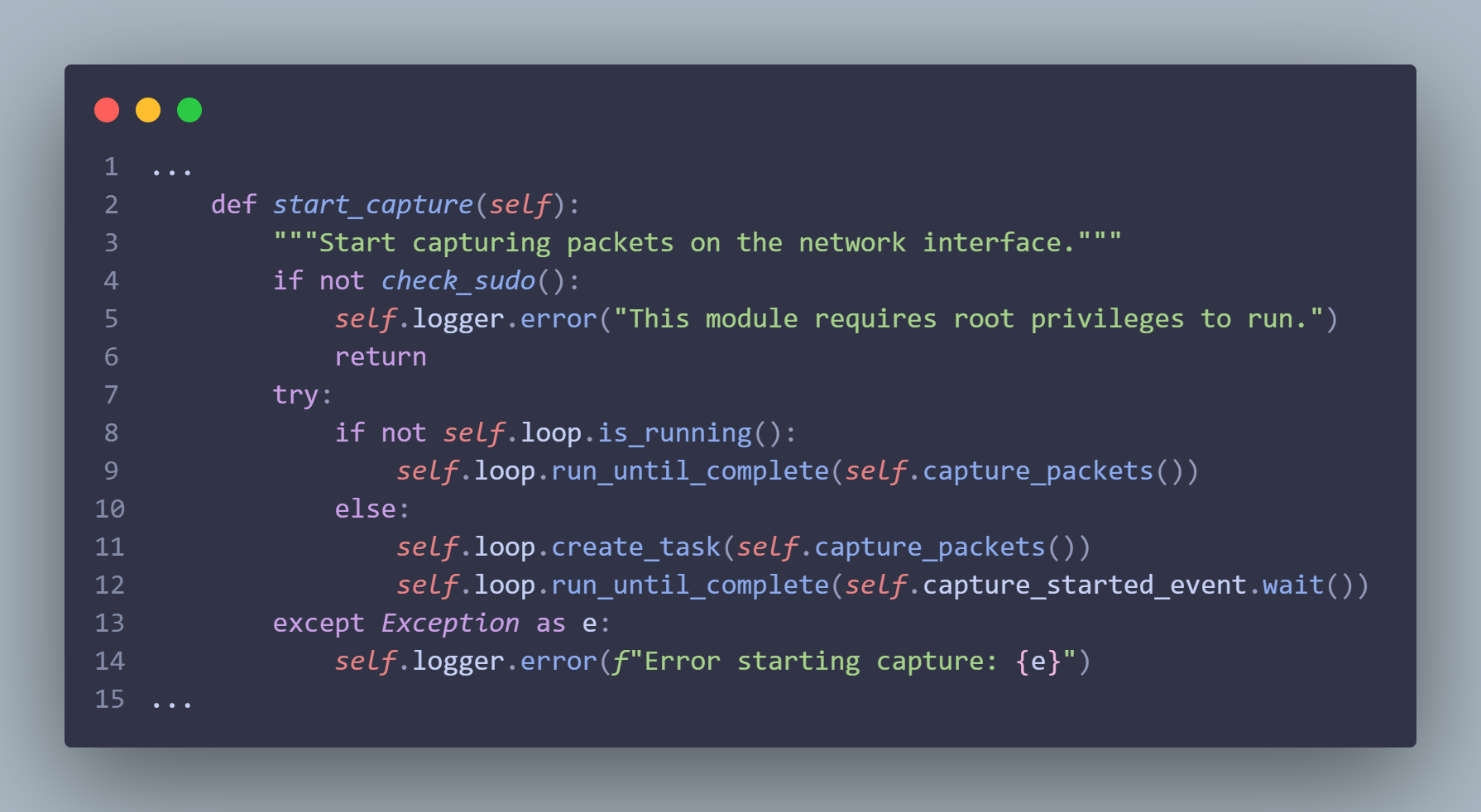


Figure 4: Start Packet Capture Code Snippet

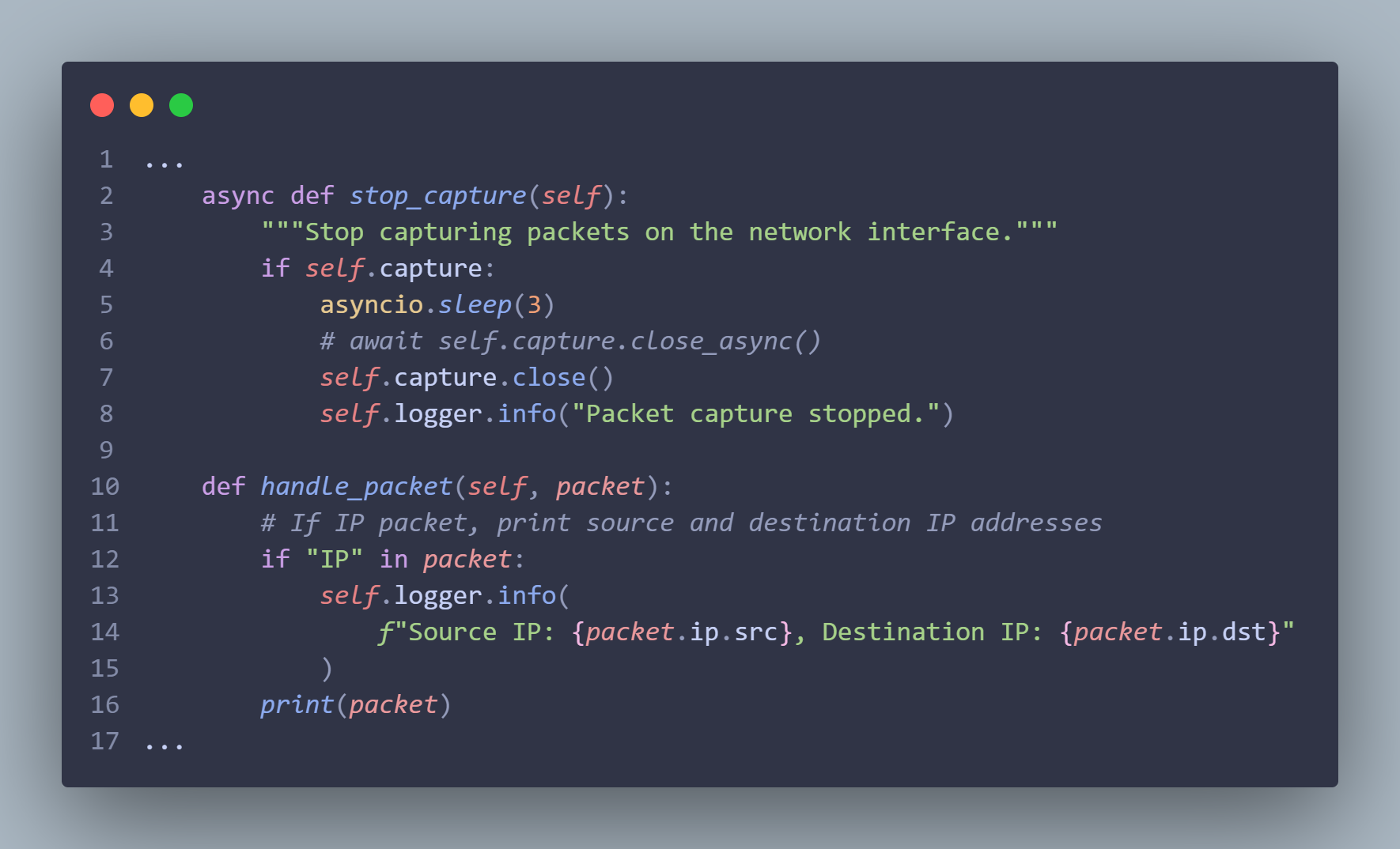


Figure 5: Stop Capture & Handle Packet Code Snippet

* The first set of Gen-AI attack tools from the GitHub Copilot and Phind models successfully compile through the normal GCC compiler.

~~~~

Figure 6: Phind Model Banner-Grabber Code Snippet

### Mitigations & Contingency Plans

* N/A - No missed milestones for this period.