Charger Active Defense v1.0 Team 2 - Group 12

## Team Lead in the Reporting Period: William Lochte

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 whether current fuzz testing properly crashes 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 concentrated on dynamic memory analysis using Valgrind, static analysis with LDRA Testbed, and applying fuzzing workflows on the first set of Gen-AI attack tools from GitHub Copilot and Phind. We updated all source files in the Python service to include docstrings and type hints, and we applied several Pyright pre-commit hooks for improved readability and documentation. Additionally, we began to update our User Guide and README with some information about the Python service and this semester's efforts to ease the documentation workload required by the semester's end.

We utilized LDRA, Valgrind, and FlawFinder to analyze the AI-generated code from Copilot and Phind. Our findings showed that the C++ and C code generated by Phind did not produce any memory leaks and did not contain any known vulnerabilities. However, the C code generated by Copilot exhibited possible buffer overflow vulnerabilities along with other well-known issues.

Our short-term goals for this period were:

* Apply the fuzzing workflow to the 1st set of Gen-AI attack tools.
* Perform LDRA static analysis on 1st set of Gen-AI attack tools.
* Perform Valgrind dynamic memory leak testing on 1st set of Gen-AI attack tools.

During this period, we encountered a few issues:

* When running AFLnet against the port scanner tools from the AI models, both programs gave an indicator on the " SLOW " summary screen in bold, red text. This indicator is similar to the one seen previously when running AFLnet against Masscan and Medusa in the first semester. With this occurring again, we are unsure if there is a deeper flaw in our current AFLnet configuration, Virtual Box configuration, or kernel-level hardware incompatibility on the system causing the issue. We will bring this issue up with our sponsor in a meeting sometime during the next reporting period to discuss mitigation, plan of action, or recommendations with other AFLnet testing against the AI-generated attack tools in the future.
* We initially planned to meet with the sponsor this week to provide an in-person progress update, as we have not received any emails regarding our current status. Still, we were unable to meet due to weather conditions, but we will meet with them this week, weather permitting.

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

|  |  |
| --- | --- |
| Responsibility | Task |
| Noah Sickels | Compare the static analysis and fuzz testing results from the 1st set of Gen-AI attack tools and compile them into a written report. |
| Adam Brannon | Develop the 2nd attack tool (FTP password brute-force) using the GitHub Copilot model. |
| William Lochte | Develop the 2nd attack tool (FTP password brute-force) using the Phind model. |

## Milestone Status Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Milestone / Task | Projected Due Date | Completion Date | Status | Notes |
| Fuzzing Workflow Applied on 1st Gen-AI Attack Tools | 2/21 | 2/21 | Complete | We were able to instrument the attack tools from both models and began fuzz testing with AFLnet for a 1-week duration. |
| Static & Dynamic Analysis Performed on 1st Gen-AI Attack Tools | 2/21 | 2/21 | Complete | We used LDRA, Valgrind, and FlawFinder on the 1st set of Gen-AI attack tools. |

## Level of Effort / Individual Responsibility Record

|  |  |  |
| --- | --- | --- |
| Member | Hours / Period | Total Hours / Spring |
| Noah Sickels | 28 | 94 |
| Adam Brannon | 8 | 39 |
| William Lochte | 3 | 35 |
| Total | 39 | 168 |

|  |  |
| --- | --- |
| Member | Individual Accomplishments |
| Noah Sickels | Fixed dates in the timeline, milestones, and activities section based on feedback from brief 3.   Wrote a guide for internal use and instructions on creating issues, branches, and pull requests to our GitHub repo.   Added docstrings and type-hints to all source files in the Python service.   Ran Pyright, Ruff, and other pre-commit hooks against all source files in the Python service.  Instrumented, debugged, and applied fuzzing workflow to the 1st Gen-AI attack tools from Phind and Copilot. |
| Adam Brannon | Performed LDRA static analysis on both banner-grabbing attack tools.  Use of flawfinder on both banner-grabbing tools. |
| William Lochte | Performed Valgrind dynamic analysis for 1st set of attack tools to search for memory leaks. |

## Milestone Completion & Analysis

* Successfully instrumented and applied fuzzing workflow to the GitHub Copilot attack tool.

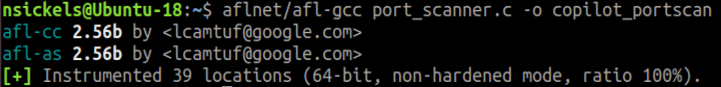


Figure 1: Copilot Port Scanner Instrumented with AFLnet

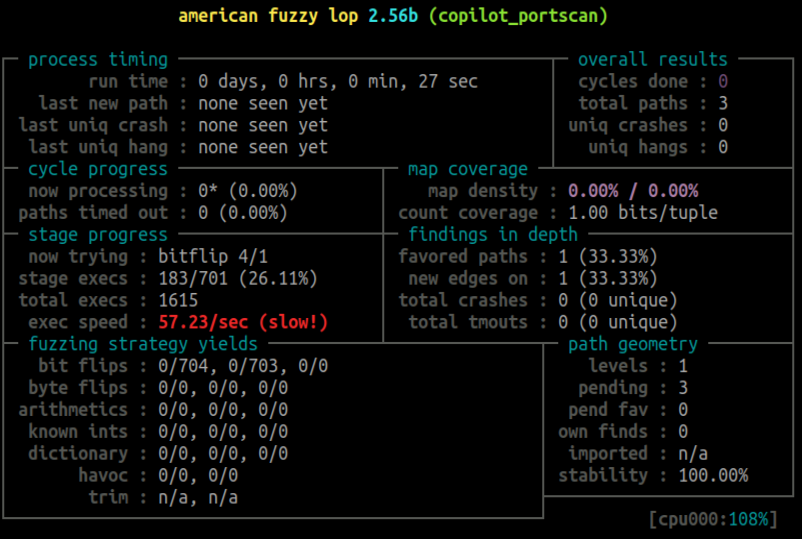


Figure 2: Fuzz Testing on Copilot Port Scanner

* Successfully instrumented and applied fuzzing workflow to the GitHub Copilot attack tool.

A black background with white text

AI-generated content may be incorrect.

Figure 3: Phind Port Scanner Instrumented with AFLnet

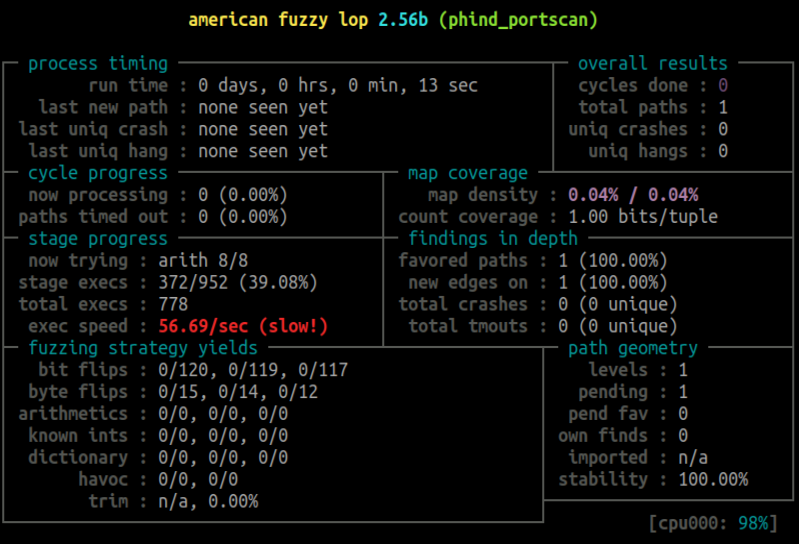


Figure 4: Fuzz Testing on Phind Port Scanner

* Valgrind dynamic memory leak analysis was performed for both generated attack tools.

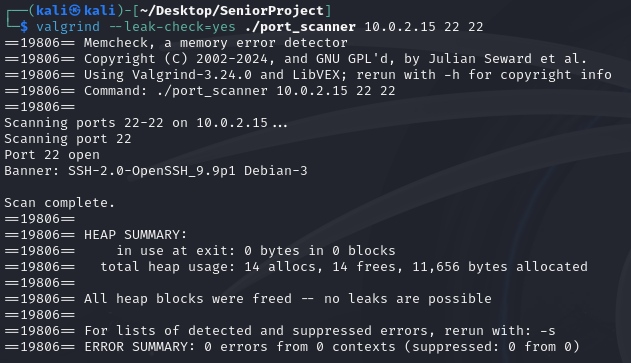


Figure 5: Valgrind Memory Analysis of Copilot Port Scanner

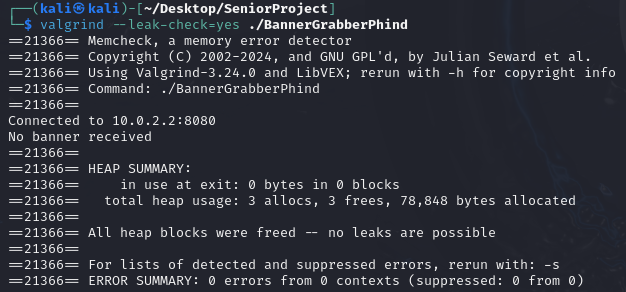


Figure 6: Valgrind Memory Analysis of Phind Port Scanner

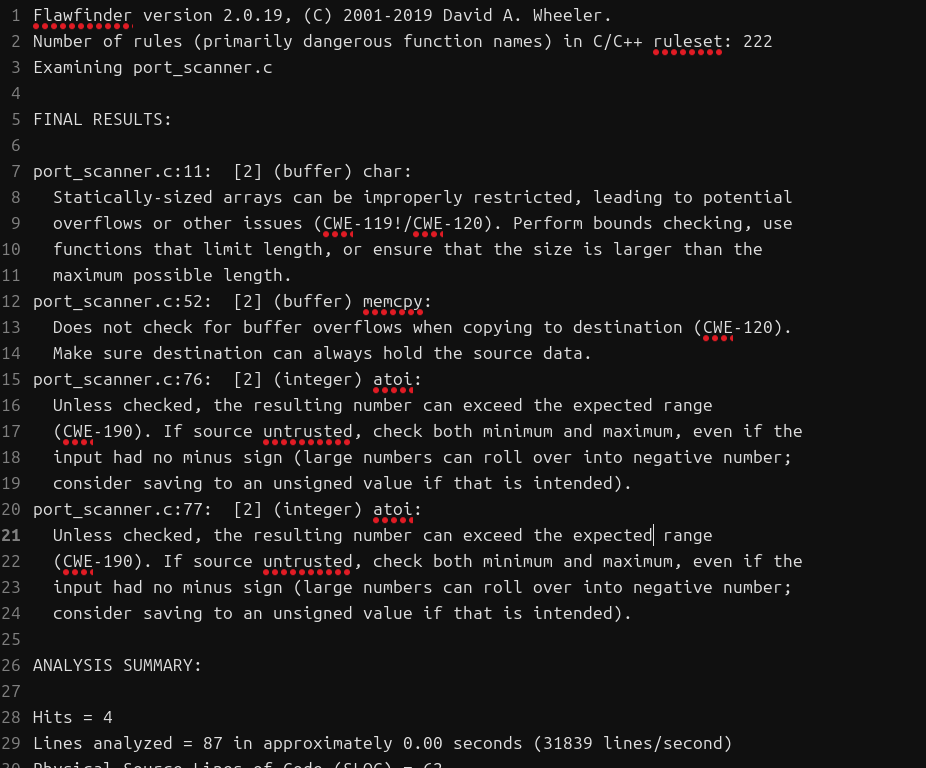


Figure 7: Flawfinder analysis of port scanner

* Performed LDRA static analysis on both generated attack tools.

### 

Figure 8: LDRA Code Review Report

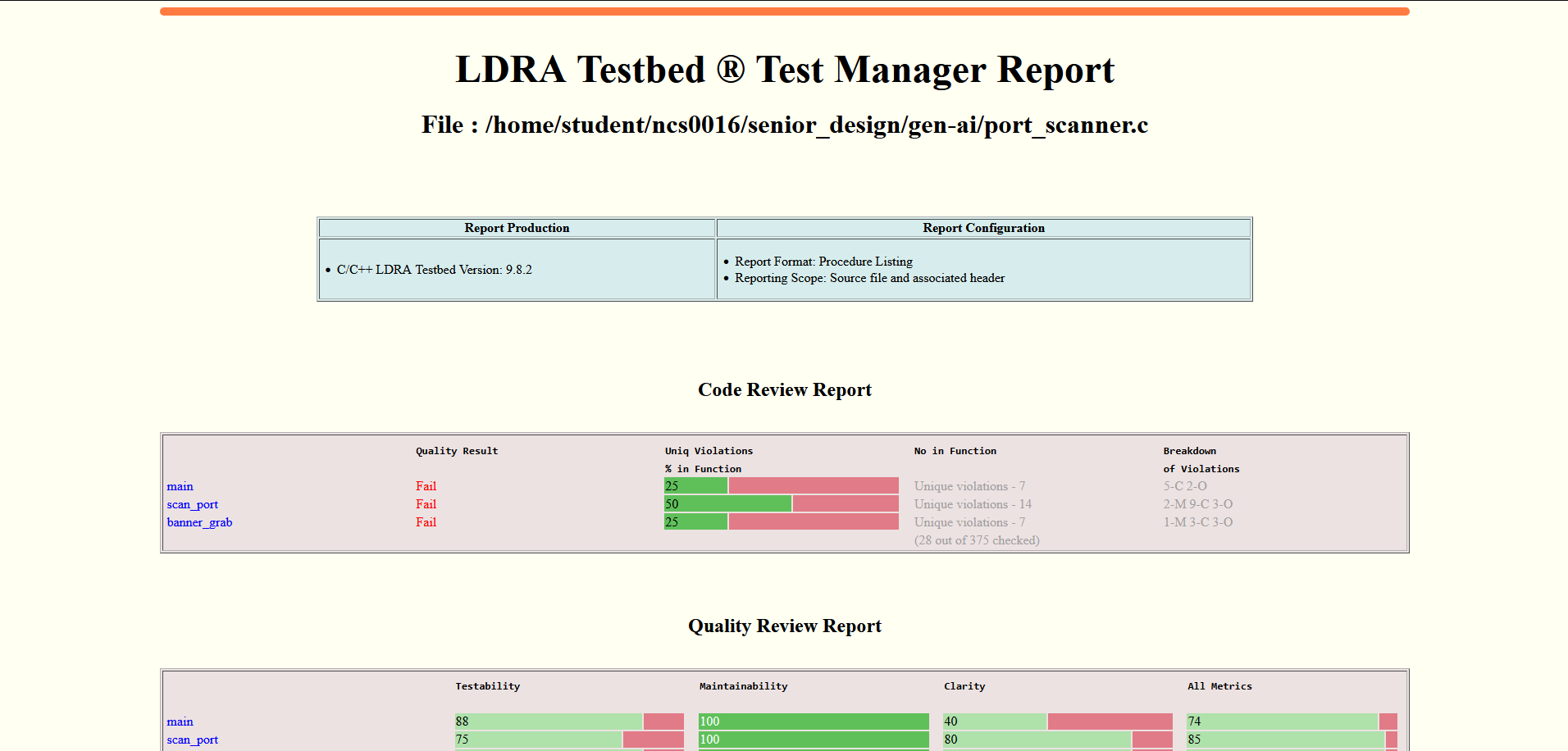


Figure 9: LDRA Test Manager Report

### Mitigations & Contingency Plans

* N/A - No missed milestones.