Fuzz Tool Selection Report

Charger Active Defense – G12

Noah Sickels, Adam Brannon, William Lochte

**Background**

Our senior design group is the second team working on the Charger Active Defense project. This project aims to develop a fuzzing workflow that effectively tests the networking aspects of the selected target applications, Medusa and Masscan. We strive to identify any hangs or crashes that may occur, which can then be sent back to the host machine to potentially disrupt or halt the adversary's tool.

In addition to selecting our target attack tools, we needed to choose appropriate fuzzing tools for our tests. After researching various fuzz testing tools that are network-compatible, open-source, and suitable for testing C or C++ applications, we compiled a list of tools for use throughout the project, as shown in Table 1 below.

Table 1: Fuzz Testing Tools

|  |  |  |  |
| --- | --- | --- | --- |
| Tool | Version | Last Update | Use Case |
| AFLnet (AFLplusplus) |  | July 25, 2024 | Dynamic Protocol Fuzzing |
| Fuzzowski | 0.8.2 | January 12, 2022 | Dynamic Protocol Fuzzing |
| Scapy | 2.6.0 | September 28, 2024 | Pcap Traffic Generation |
| Radamsa | 0.7 | February 27, 2024 | Dynamic Protocol Fuzzing |
| Randpkt | 4.4.1 | October 17, 2024 | Pcap Traffic Generation |
| Peach Fuzzer | 4.3 | August 12, 2021 | Dynamic Protocol Fuzzing |

When testing Medusa and Masscan, we can utilize two main methods: dynamic protocol fuzzing and pcap traffic generation. The first method involves live fuzzing of the protocol or service stack, usually through a "wrapper." This wrapper acts as an intermediary between the attacking tool and the target machine. It enables the fuzzing tool to incorporate random data into the responses sent from the victim machine back to the host to elicit a crash or hand in the application.

The second method uses complementary tools to create packet capture files (commonly known as "pcap") with randomly fuzzed traffic. Since both Medusa and Masscan support input from pcap files, this approach enables us to test various aspects of the attacking tools while requiring fewer resources on the host machines.

**Selection Rationale**

**AFLnet (AFLplusplus)**

AFLnet is a greybox fuzzer designed for testing internet protocol implementations, and it was explicitly mentioned in our sponsor's project proposal. Unlike other protocol fuzzers, AFLnet employs a mutational approach and utilizes state feedback and code coverage feedback to guide the fuzzing process [1]. It is built as an extension of a fork of the existing AFLplusplus project, enabling the well-known American Fuzzy Lop to test networking protocols effectively.

AFLnet utilizes the first method of fuzz testing, which uses a wrapper to mediate the traffic between the attack tool and the host. To implement this method, the source code files of the target attack tools must be compiled and instrumented. Instrumenting the binary involves recompiling the source code with AFLplusplus's custom GCC compiler, afl-gcc. This custom compiler is optimized to facilitate the fuzzing process when using AFLnet or AFLplusplus. If the binary is not recompiled with afl-gcc, AFLplusplus may throw an error and prevent further testing in some cases.

During our efforts to instrument the binaries for Medusa and Masscan, we encountered several complications, particularly with Medusa. The primary issue was a missing dependency; AFLnet is designed primarily for traditional Debian operating systems, and some of the packages needed to use afl-gcc were unavailable on the Kali distribution we were using.

We switched to an Ubuntu 18.04 virtual machine to address this, where all the necessary packages were available by default. However, we faced another hurdle: Medusa relies on specific OpenSSL header files from a deprecated version of the OpenSSL library, which was not included in our version of Ubuntu.

After researching various solutions, we managed to get some initial tests working with AFLnet and Medusa. However, we require more time to conduct thorough testing before proceeding to comprehensive fuzz testing in the second semester.

Unlike Medusa, Masscan compiled with afl-gcc without issues and was verified to be compatible with fuzz testing.

There are a few limitations to AFLnet, primarily in the protocols it supports, such as the PostgreSQL protocol, which seemed most favorable for fuzzing Medusa; it does have extensive custom libraries to build them from scratch. Because of AFLnet's potential viability with Medusa and Masscan and being explicitly mentioned by the sponsor, we decided to use this as one of the main fuzz testing tools for the project.

**Fuzzowski**

Fuzzowski is a custom fuzzer developed in Python that aids in the complete process of network protocol fuzzing. It allows users to define communications, identify potential causes of service crashes, and more [2]. By default, Fuzzowski supports a wide range of networking protocols and services. As a fork of OpenRCE's Sulley, it includes an extensive library that enables users to create custom tests for protocols that are not natively supported.

Due to its ability to test virtually any specified protocol or port through custom libraries, Fuzzowski was selected as a secondary tool for dynamic protocol fuzzing for the project.

**Scapy**

Scapy is a powerful interactive packet manipulation program and library based on Python. It allows users to forge, decode, capture, and store packets across the network [3]. Although Scapy is not specifically designed for fuzz testing network applications like AFLnet or Fuzzowski, it does offer the capability to write scripts that create packet capture files (pcaps) containing fuzzed network traffic.

Since Scapy provides the means to generate fuzzed pcap files, and Medusa and Masscan can ingest input files in pcap format, we chose Scapy as one of our fuzzed packet capture generation tools.

**Radamsa**

Radamsa is a community-developed fuzzer that reads valid data samples and generates unique outputs from them. It is lightweight and scriptable, making it an excellent tool for dynamic protocol fuzzing or generating fuzzed packets, mainly when used in conjunction with other tools like Scapy [4]. One of Radamsa's notable features is the ability to specify a port/protocol and an IP address, enabling interaction with the networking components of tools like Medusa and Masscan. This functionality means that Radamsa can be used for dynamic protocol fuzzing and scripted to generate fuzzed packet captures. For these reasons, we chose to use Radamsa for this project.

**Randpkt**

Randpkt is an extension of the tool Wireshark and is used to generate fuzzed packets for a specific protocol or randomly from a list [6]. While it has a slightly limited feature set compared to Scapy, its core functionality is helpful for pcap traffic generation. Because Randpkt can quickly generate pcap files with random data, we selected it as an alternative tool if Scapy or Radamsa was insufficient.

**Peach Fuzzer**

Peach Fuzzer is a community-driven protocol fuzzing framework and was one of the first widely recognized fuzzers, released in 2004. It offers a variety of fuzzing options, including mutation-based and dynamic protocol fuzzing, and is capable of fuzzing any port and protocol provided [7]. The framework supports development with custom libraries for protocols that are not readily available. As one of the more well-known fuzz testing tools, it has received extensive media and forum coverage and allows for testing a wide range of protocols across various applications. For these reasons, we chose Peach Fuzzer as one of our testing tools for this project.

**Conclusion**

In conclusion, our senior design group has chosen fuzzing tools for the Charger Active Defense that meet our specific criteria. As we prepare for the upcoming semester, we consider two methods: packet capture (pcap) generation and dynamic protocol fuzzing. For this project, we have selected six fuzzing tools: AFLnet, Fuzzowski, Scapy, Radamsa, Randpkt, and Peach Fuzzer. While we are still determining whether these tools will be fully compatible with Medusa or Masscan, we will continue researching additional tools to utilize, develop, and possibly create our own solution.

**References**

[1] aflnet, “AFLNet: A Greybox Fuzzer for Network Protocols,” *GitHub*, Oct. 01, 2023. https://github.com/aflnet/aflnet

‌

[2] nccgroup, “GitHub - nccgroup/fuzzowski: the Network Protocol Fuzzer that we will want to use.,” *GitHub*, 2019. https://github.com/nccgroup/fuzzowski/tree/master (accessed Oct. 31, 2024).

‌

[3] “Scapy,” *GitHub*, Aug. 10, 2022. https://github.com/secdev/scapy

‌

[4] “Aki Helin / radamsa · GitLab,” *GitLab*, 2024. <https://gitlab.com/akihe/radamsa/> (accessed Oct. 27, 2024).

[5] “Wireshark Foundation / Wireshark · GitLab,” *GitLab*, 2024. https://gitlab.com/wireshark/wireshark (accessed Oct. 27, 2024).

‌

[6] “randpktdump(1),” *Wireshark.org*, 2024. <https://www.wireshark.org/docs/man->pages/randpktdump.html (accessed Oct. 27, 2024).

[7] “Files · main · GitLab.org / security-products / protocol-fuzzer-ce · GitLab,” *GitLab*, 2024. https://gitlab.com/gitlab-org/security-products/protocol-fuzzer-ce/-/tree/main?ref\_type=heads (accessed Oct. 27, 2024).

[8] “Peach Fuzzer,” *peachtech.gitlab.io*. <https://peachtech.gitlab.io/peach-fuzzer-community/>

‌

‌‌