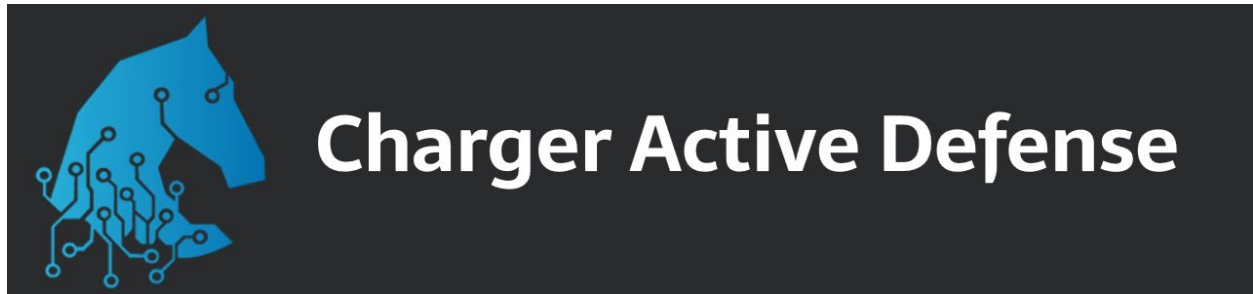


Charger Active Defense v1.0 - User Guide

Noah Sickels, Adam Brannon, William Lochte – G12



Platform [Kali](#) Platform [Ubuntu](#) Requirement [Python 3.12+](#) [Bash](#) [Docker](#)

Table of Contents

- [Project Overview](#)
 - [System Overview Diagram](#)
 - [Project Directory Structure](#)
- [Prerequisites](#)
- [Testbed Configuration](#)
 - [Network Configuration](#)
- [Usage & Installation](#)
 - [Bash Script \(Recommended\)](#)
 - [Dockerfile \(WIP\)](#)
 - [Manual Installation \(Recommended\)](#)
 - [Usage](#)
- [Demonstration Video](#)
- [References](#)

Project Overview

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.

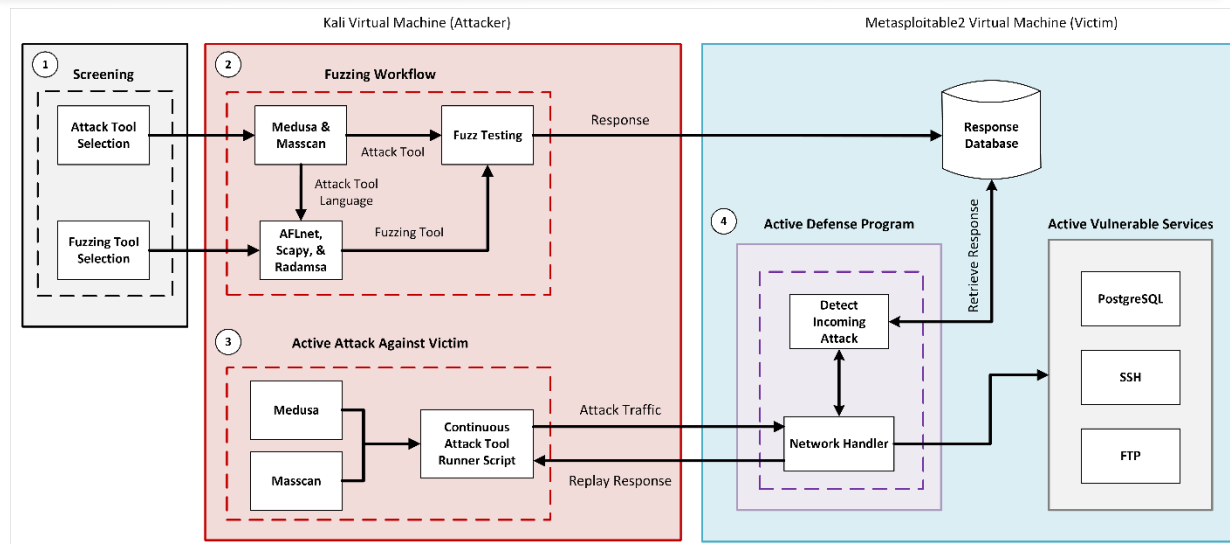
This project is divided into two main phases - the fuzzing workflow and the active defense tool. The fuzzing workflow phase consists of the selection of fuzzing tools, two attack tools to fuzz, and the development of a fuzzing workflow. The active defense tool phase consists of developing a tool that can detect and respond to attacks on the network and send the fuzzed responses back to the adversary's tool.

You can find the sponsor's project proposal slide below.

No classified or proprietary information on this page		DATE:										
Charger Active Defense – ChAD v1.0												
<p>Overview</p> <ul style="list-style-type: none"> Cyberattacks may utilize commonly available open-source tools in various stages of the cyber kill chain such as nmap for port scanning or hydra for brute force login attacks These tools exchange information with exposed services running on the targeted hosts Many of these tools are written, in part, in memory-unsafe languages such as C or C++ <ul style="list-style-type: none"> Nmap – 38.0% C / 17.3% C++ Hydra – 94.5% C Such tools may make the attacker's computer vulnerable to an active defensive response 	<p>Statement of Need or Problem</p> <ul style="list-style-type: none"> What is needed is an active defense mechanism that can, once an attack is detected, supply a response that neutralizes the attacking app (and in a nation-state scenario might be a precursor to infecting the attacker's system with malware) The goal of this project is to develop and demonstrate proof-of-concept responses that will cause the attacking applications to crash or hang on the attacker's computer 											
<p>Project Objective and Deliverables</p> <ul style="list-style-type: none"> Identify a cyberattack tool implemented with a significant percentage of C/C++ code, preferably a tool WITHOUT publicly documented fuzz test history Employ static analysis to assess suitability of tool selected Perform extensive fuzz testing using AFL / AFL++ derived tools to identify any host responses that cause the attacking tool to crash or hang – these are potential active responses Employ and report on the use of AI / LLM to generate active responses that crash or hang the application Implement a fake network service application running in a virtual machine that may be attached to any TCP or UDP port Use the fake service app to evaluate every active response to identify those that crash or hang the attacking application executing on a separate Kali Linux virtual machine Submit a conference/journal paper summarizing all results 	<table border="1"> <tr> <td>University:</td> <td>Govt Client:</td> </tr> <tr> <td>Team Lead:</td> <td>SME:</td> </tr> <tr> <td>Faculty Advisor: David Coe</td> <td>Alternates:</td> </tr> <tr> <td colspan="2">Capstone Coordinator (at university):</td> </tr> <tr> <td colspan="2">Budget:</td> </tr> </table>		University:	Govt Client:	Team Lead:	SME:	Faculty Advisor: David Coe	Alternates:	Capstone Coordinator (at university):		Budget:	
University:	Govt Client:											
Team Lead:	SME:											
Faculty Advisor: David Coe	Alternates:											
Capstone Coordinator (at university):												
Budget:												
<p>Research Topic Numeric Designator:</p>												

7

System Overview Diagram



Project Directory Structure

Highlights

- **User_Guide.docx:** User guide document for the project.
- **workflow.sh:** Bash script for installing and building the attack and fuzzing tools for the workflow.
- **Dockerfile:** WIP Dockerfile for fuzzing workflow.
- **Makefile:** Makefile for building and running the Docker container.
- **Background Screening:** Contains CVEs, LDRA Static Analysis test results, and Valgrind test results for all attack tool candidates.
- **Config:** Contains configuration files for the testbed.
- **Deliverables:** Contains project deliverables, including reports, briefings, design review, and final report.
 - **G12_attack_tool_selection_report.docx:** Attack tool selection report.
 - **G12_fuzz_tool_selection_report.docx:** Fuzz tool selection report.
 - **G12_fuzzing_results_analysis.docx:** Fuzzing tools compatibility testing results and analysis report.
- **Fuzzing:** Contains fuzzing-related files, including attack tool commands and fuzzing tool files
 - **Attack_Tool_Commands.md:** Commands used for the attack tools.
 - **password_list.txt:** Password list used for testing.\
 - **repeat_medusa.sh:** Script to repeatedly run Medusa.

Directory Tree

Charger Active Defense v1.0 - Senior Design Project

```
.
├── README.md
├── config
│   ├── Metasploitable2_Running_Services.txt
│   └── Testbed_Config.md
├── deliverables
│   ├── Conference-template-A4.doc
│   ├── G12_attack_tool_selection_report.docx
│   └── G12_fuzz_tool_selection_report.docx
```

- G12_fuzzing_results_analysis.docx
- G12_updated_milestones.docx
- G12_updated_timeline.png
- Project_Timeline_v2.gan
- briefings
 - brief_1
 - G12_briefing_1_progress_report.docx
 - brief_2
 - G12_briefing_2_progress_report.docx
 - fuzzowski_medusa_telnet.pcap
 - brief_3
 - G12_briefing_3_progress_report.docx
- design_review
 - 495_488_design_review_template.pptx
 - Behavioral_Decomposition.vsd
 - Functional_Decomposition.vsd
 - G12_design_review_presentation.pptx
 - G12_level_of_effort.docx
 - G12_marketing_requirements.docx
 - Updated_Behavioral_Decomposition.png
 - Updated_Functional_Decomposition.png
 - individual_level_of_effort.md
- final_report
- proposal
 - Project-Proposal-Submission.pdf
- timeline_and_milestones
 - initial
 - Project_Timeline_Proposal.gan
 - milestone_analysis.md
- fuzzing
 - afl-qemu-trace
 - fuzzowski.medusa.ftp
 - ftp.py
 - fuzzshark
 - ~src
 - icmp.masscan
 - fuzz_ping.sh
 - grammer.bnf
 - internet_checksum.py
 - requirements.txt
 - send_icmp.py
 - medusa.postgresql.afl_1
 - cmdline
 - fuzz_bitmap
 - fuzzer_setup
 - fuzzer_stats
 - ~hangs
 - init_attempt
 - medusa_config.txt
 - wrapper.c

- └─ wrapper.sh
 - └─ ~plot_data
 - └─ ~queue
- └─ peach_fuzz
 - └─ network_fuzzing.xml
 - └─ peachfuzzer.dockerfile
- └─ radamsa
 - └─ Radamsa_Instructions.md
 - └─ ~img
- └─ randbytes
 - └─ ftp_server.py
 - └─ pcap_parsing.py
- └─ randpkt
 - └─ ~src
- └─ scapy.radamsa
 - └─ radamsa_scapy_pcap_fuzzing.py
- └─ misc
 - └─ Attack_Tool_Commands.md
 - └─ Attack_Tool_Info.md
 - └─ password_list.txt
 - └─ repeat_medusa.sh
- └─ pcaps
 - └─ baseline
 - └─ masscan.pcap
 - └─ medusa_ftp.pcap
 - └─ medusa_postgresql.pcap
 - └─ medusa_ssh.pcap
 - └─ scapy
 - └─ ftp_login_packets.pcap
 - └─ fuzz_test_1.pcap
 - └─ medusa_ftp_brute_force.pcap
 - └─ medusa_ftp_fail.pcap
 - └─ nmap_ftp_scan.pcap
- └─ project_overview.png
- └─ research
 - └─ CVEs.md
 - └─ Fuzzing_Tools.md
 - └─ cmiller-csw-2010.pdf
- └─ background_screening
 - └─ ldra
 - └─ aircrack-ng
 - └─ aircrack-ng.mts.htm
 - └─ masscan
 - └─ masscan.mts.htm
 - └─ medusa
 - └─ medusa.mts.htm
 - └─ netdiscover
 - └─ netdiscover.mts.htm
 - └─ reaver
 - └─ reaver.mts.htm

```

├── yersinia
│   └── yersinia.mts.htm
└── valgrind
    ├── commands.txt
    ├── masscan.txt
    ├── medusa_ftp.txt
    ├── medusa_postgres.txt
    ├── medusa_ssh.txt
    └── netdiscover.txt

```

Full Explanation

- **README.md:** This file.
- **project_overview.png:** Image of the project overview.
- **config:** Contains configuration files.
 - **Testbed_Config.md:** Configuration details for the testbed.
- **deliverables:** Contains project deliverables, including the tool reports, proposal presentation slides, briefings, design review, and final report.
 - **G12_attack_tool_selection_report.docx:** Attack tool selection report.
 - **G12_fuzz_tool_selection_report.docx:** Fuzz tool selection report.
 - **G12_fuzzing_results_analysis.docx:** Fuzzing results analysis.
 - **G12_updated_milestones.docx:** Updated milestones.
 - **G12_updated_timeline.png:** Updated timeline.
 - **Project_Timeline_v2.gan:** Gantt chart file for the project timeline.
 - **briefings:** Contains briefing files.
 - **brief_1:** Briefing 1 files.
 - **G12_briefing_1_progress_report.docx:** Briefing 1 progress report.
 - **brief_2:** Briefing 2 files.
 - **G12_briefing_2_progress_report.docx:** Briefing 2 progress report.
 - **fuzzowski_medusa_telnet.pcap:** Fuzzowski Medusa Telnet PCAP file.
 - **brief_3:** Briefing 3 files.
 - **G12_briefing_3_progress_report.docx:** Briefing 3 progress report.
 - **design_review:** Contains design review files.
 - **495_488_design_review_template.pptx:** Design review template.
 - **Behavioral_Decomposition.vsd:** Behavioral decomposition Visio diagram.
 - **Functional_Decomposition.vsd:** Functional decomposition Visio diagram.
 - **G12_design_review_presentation.pptx:** Design review presentation.
 - **G12_level_of_effort.docx:** Level of effort document.
 - **G12_marketing_requirements.docx:** Marketing requirements document.
 - **Updated_Behavioral_Decomposition.png:** Updated behavioral decomposition diagram image.
 - **Updated_Functional_Decomposition.png:** Updated functional decomposition diagram image.

- **individual_level_of_effort.md**: Individual level of effort document.
- **final_report**: Final report files.
- **proposal**: Proposal files.
 - **Project-Proposal-Submission.pdf**: Project proposal presentation slides.
- **timeline_and_milestones**: Contains timeline and milestones files.
 - **initial**: Initial timeline and milestones.
 - **Project_Timeline_Proposal.gan**: Initial project timeline proposal.
 - **milestone_analysis.md**: Milestone analysis.
- **fuzzing**: Contains fuzzing-related files.
 - **afl-qemu-trace**: AFL QEMU trace binary.
 - **fuzzowski.medusa.ftp**: Fuzzowski Medusa FTP files.
 - **ftp.py**: FTP file for Fuzzowski Medusa.
 - **fuzzshark**: Fuzzshark files.
 - **medusa.postgresql.afl_1**: Medusa PostgreSQL AFL files.
 - **init_attempt**: Initial attempts with AFLnet.
 - **medusa_config.txt**: Medusa configuration file for wrapper.
 - **wrapper.c**: Custom wrapper source file.
 - **wrapper.sh**: Custom wrapper script.
 - **peach_fuzz**: Peach Fuzz files.
 - **network_fuzzing.xml**: Network fuzzing XML model file.
 - **peachfuzzer.dockerfile**: Peach Fuzzer Dockerfile.
 - **radamsa**: Radamsa files.
 - **Radamsa_Instructions.md**: Radamsa testing instructions.
 - **randbytes**: Randbytes files.
 - **ftp_server.py**: FTP server file.
 - **pcap_parsing.py**: PCAP parsing file with Scapy.
 - **randpkt**: Randpkt files.
 - **scapy.radamsa**: Scapy Radamsa files.
 - **radamsa_scapy_pcap_fuzzing.py**: Radamsa & Scapy PCAP fuzzing Python script.
- **misc**: Miscellaneous files.
 - **Attack_Tool_Commands.md**: Commands for attack tools used during compatibility testing.
 - **Attack_Tool_Info.md**: Information about attack tools.
 - **password_list.txt**: Password list used for testing.
 - **repeat_medusa.sh**: Script to repeatedly run Medusa.
- **pcaps**: Contains PCAP files.
 - **baseline**: Baseline PCAP files.
 - **scapy**: Scapy PCAP files.
- **background_screening**: Contains test-related files.
 - **ldra**: LDRA test files.
 - **aircrack-ng/aircrack-ng.mts.htm**: Aircrack-ng LDRA test files.

- **masscan/masscan.mts.htm:** Masscan LDRA test files.
- **medusa/medusa.mts.htm** Medusa LDRA test files.
- **netdiscover/netdiscover.mts.htm:** Netdiscover LDRA test report.
- **reaver/reaver.mts.htm:** Reaver LDRA test report.
- **yersinia/yersinia.mts.htm:** Yersinia LDRA test report.
- **valgrind:** Valgrind test results for each attack tool candidate.
 - **commands.txt:** Commands used for running the Valgrind tests.
 - **masscan.txt:** Masscan Valgrind test results file.
 - **medusa_ftp.txt:** Medusa FTP Valgrind test results file.
 - **medusa_postgres.txt:** Medusa PostgreSQL Valgrind test results file.
 - **medusa_ssh.txt:** Medusa SSH Valgrind test results file.
 - **netdiscover.txt:** Netdiscover Valgrind test results file.
- **research:** Contains research-related files.
 - **CVEs.md:** List of CVEs from all attack tool candidates.
 - **Fuzzing_Tools.md:** Background research on possible fuzzing tools.
 - **cmiller-csw-2010.pdf:** Research paper on general fuzzing and fuzzing tools.

Prerequisites

- VirtualBox 7.1.0 (or later)
- Kali Linux 2023.4 (or later) or Ubuntu 20.04 (or later)
- Wi-Fi/Ethernet Adapter that supports promiscuous mode.
- Packages: clang, graphviz-dev, libcap-dev, git, make, gcc, autoconf, automake, libssl-dev, wget, curl, dos2unix, php-cli.

Testbed Configuration

The Chad workflow testbed comprises two virtual machines: Kali Linux 2023.4 (or newer) and Metasploitable2.

- You can download a pre-built Kali Linux VM from their website [here](#).
- Rapid7 provides a pre-built Metasploitable2 VM from their website [here](#).

For detailed configuration information, please refer to the table below.

Table 1: VirtualBox VM Configurations

Component	Configuration	
Hypervisor	VirtualBox	7.1.0
Virtual Machine 1 (Host)	OS	Kali Linux 2024.3
	Kernel Version	Linux kali 6.10.11-amd64
	GCC Version	14.2.0 (Debian 14.2.0-3)
	Network Adapter 1	NAT
	Network Adapter 2	Internal Network (<i>intent</i>)
	IP Address	192.168.1.99 /24
	Miscellaneous	16,384 MB RAM, 2 CPU cores, 80 GB HDD
Virtual Machine 2 (Target)	OS	Metasploitable2
	Kernel Version	Linux Metasploitable 2.6.24-16-server
	GCC Version	4.2.4 (Ubuntu 4.2.4-lubuntu4)
	Network Adapter 1	Internal Network (<i>intent</i>)
	Network Adapter 2	<i>Optional</i>
	IP Address	192.168.1.100 /24
	Miscellaneous	2,048 MB RAM, 1 CPU core, 8 GB HDD

Important

“*Virtual Machine 1 (Host)*” refers to the attacking virtual machine running Kali, which runs Medusa and Masscan against the target VM.

“*Virtual Machine 2 (Target)*” refers to the virtual machine running Metasploitable2, which has vulnerable services active.

Network Configuration

For both VMs to communicate with each other, you will need to configure the network adapters in VirtualBox and on the VMs' network interfaces.

You can use either a physical network adapter that supports promiscuous mode or virtual adapters through your hypervisor; however, we recommend using the virtual adapters as shown below.

VirtualBox Network Adapter Settings Configuration

Kali Virtual Machine

1. Open VirtualBox and select the Kali VM.
2. Click on the *Settings* icon and choose the *Network* tab.
3. Under *Adapter 1*, select *Attached to: NAT*.
4. Now select the *Adapter 2* tab.
5. Check the box for *Enable Network Adapter*.
6. Select *Attached to: Internal Network*.
7. In the *Name* field, enter `intent`
8. Click *Ok* to save the settings and close out of the window.

Metasploitable2 Virtual Machine

1. Open VirtualBox and select the Metasploitable2 VM.
2. Click on the *Settings* icon and navigate to the *Network* tab.
3. Under *Adapter 1*, select *Attached to: Internal Network*.
4. In the *Name* field, enter `intent`.
5. Click *Ok* to save the settings and close out of the window.

Host Machine Network Configuration

Once the virtual machines are configured, start both VMs and configure the network interfaces on each VM. Note that the network interface names may vary depending on the operating system and version. We will use the default network interface names for the examples below. The IP addresses used are default private IP addresses, but you can use any IP address within the same subnet.

Kali Host Machine

Open terminal and run the following command(s) to view the network interfaces:

```
sudo ip addr
# or
sudo ifconfig
```

You should see the network interfaces listed. If you enabled network adapter 1, you should see your internet-facing network interface named `eth0` or `enp0s3`. If you enabled network adapter 2, you should see an interface named `eth1` or `enp0s8`. Otherwise, you may need to manually configure the network interfaces on your system.

We will use `eth1` for the internal network communication as an example for the commands below.

Set the IP address for the internal network interface `eth1` (*requires root privileges*):

```
sudo ip addr add 192.168.1.99 dev eth1
# or
sudo ifconfig eth1 192.168.1.99
```

You can verify the IP address is set correctly by running `sudo ip addr` or `sudo ifconfig` again.

Metasploitable2 Target Machine

By default, the Metasploitable2 VM has no GUI and should boot into a terminal window. Verify the available network interfaces by running the following command(s):

```
sudo ip addr  
  
# or  
  
sudo ifconfig
```

You should see the network interface `eth0` listed. Set the IP address for the internal network interface `eth0` (*requires root privileges*):

```
sudo ip addr add 192.168.1.100 dev eth0  
  
# or  
  
sudo ifconfig eth1 192.168.1.100
```

You can verify the IP address is set correctly by running `sudo ip addr` or `sudo ifconfig` again.

Verify Network Connection

Once the network interfaces have been configured on both VMs, you can test the network connection between the two VMs by running the following commands:

```
# On the Kali VM  
ping 192.168.1.100  
  
# On the Metasploitable2 VM  
ping 192.168.1.99
```

If the network connection is successful, you should see the ping responses from the target VM (e.g., `64 bytes from 192.168.1.100: icmp_seq=1 ttl=64 time=0.171ms`).

If the connection is unsuccessful, restart the VMs and verify the network configurations.

Network adapters may reset at times, so we recommend checking the network adapter IP addresses to ensure they are still set correctly.

If they are not set, you can reconfigure them by repeating the steps above.

Usage & Installation

There are three ways to install and use the tools necessary for the Chadv1.0 workflow: using the Bash script, the Dockerfile, or manually.

Bash Script (Recommended)

To install the attack tools and fuzzing tools, you can use the provided Bash script as shown below (***requires root privileges***):

```
# Download the workflow script through curl or manually
from the repository
curl -O https://raw.githubusercontent.com/NCSickels/chad
v1.0/main/scripts/workflow.sh

# Make the script executable
chmod u+x workflow.sh

# Using the workflow script
sudo ./workflow.sh --help

# To install all tools (attack and fuzzing)
sudo ./workflow.sh install

# To build all tools (attack and fuzzing)
sudo ./workflow.sh build
```

Once the script finishes, you should see a new directory created `chadv1.0` where you will find a `fuzzing_tools` folder and `attack_tools` folder. Inside you will find AFLnet, Radamsa, Medusa, and Masscan artifacts, respectively.

Note

If you encounter the error: `-bash: ./workflow.sh: /bin/bash^M: bad interpreter: No such file or directory`, it is most likely due to the script being in DOS format for a UNIX system. To fix this, you can use the `dos2unix` command to convert the script to UNIX format. You can install it through Apt package manager using the command `sudo apt install dos2unix`.

Dockerfile (WIP)

⚠ Warning

Requires Docker and Make to be installed on the host machine. Docker Desktop is available [here](#).

The Chad workflow Docker implementation utilizes a Makefile and the `make` utility to build and run the Dockerfile image in a streamlined manner.

- Build the Chadv1.0 Workflow Docker image: `make build`
- Run the Chadv1.0 Workflow Docker container: `make run`

Optionally, you can build the Docker image and run the container manually using the commands below.

Build the Docker Image

```
# Build the Docker image
docker build -t workflow .
```

Run the Docker Container

```
# Run the Docker container
docker run --rm -it --name workflow -v . workflow /bin/bash
```

Manual Installation (Recommended)

Cloning the Repositories

```
# Clone the attack tool repositories
git clone https://salsa.debian.org/pkg-security-team/medusa.git
git clone https://github.com/robertdavidgraham/masscan.git

# Clone the fuzzing tool repositories
git clone https://github.com/aflnet/aflnet.git
git clone https://gitlab.com/akihe/radamsa.git
```

Install Necessary Dependencies

```
sudo apt install -y clang graphviz-dev libcap-dev git make \
gcc autoconf automake libssl-dev wget curl
```

Build the Attack Tools

```
# Build Medusa
cd medusa
./configure
make
make install
cd ..

# Build Masscan
cd masscan
# Optionally, can run `make -j` for faster compilation
make
make install
cd ..
```

Build the Fuzzing Tools

```
# Build AFLnet
cd aflnet
make clean all

cd llvm_mode
make
# If this command does not work, it most likely means that
```



```
llvm-config is not in your PATH. If so, you can add it manually  
as shown below.  
# It should be named something like llvm-config-6.0 in  
/usr/bin/  
export LLVM_CONFIG=$(ls /usr/bin/llvm-config-* 2>/dev/null |  
head -n 1)  
cd ../../  
export AFLNET=$(pwd)/aflnet  
export WORKDIR=$(pwd)  
export PATH=$PATH:$AFLNET  
export AFL_PATH=$AFLNET  
cd ..  
  
# Build Radamsa  
cd radamsa  
make  
sudo make install  
cd ..
```

This will install the necessary tools for the Chadv1.0 fuzzing workflow, including AFLnet, Radamsa, Medusa, and Masscan.

Usage

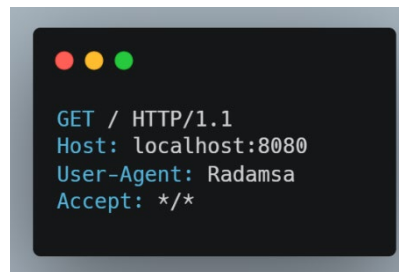
After all tools are installed and configured, you can run AFLnet or Radamsa alongside Medusa and Masscan. Due to resource limitations, we decided to pair one fuzzing tool with one attack tool. Specifically, AFLnet will be used to test Masscan, while Radamsa will be used to test Medusa.

Radamsa and Medusa

Radamsa provides two methods for fuzzing network services, allowing it to operate as either a TCP client or server. When used as a TCP server, it can intercept web traffic and fuzz it with random data before relaying it back to the specified IP address and port.

For demonstration purposes, we set up a simple PHP HTTP web server on the local host, operating on TCP port 8080, by following the steps below.

1. Create a directory named ``www`` and `cd` into it.
2. Create two separate files within the ``www`` directory, `index.html` and `http-request.txt`.
3. In the `index.html` file, use your text editor of choice and add the line: `<h1> Radamsa Test </h1>`. You will use this file to test the PHP server and ensure it is running and visible on the network.
4. In the `http-request.txt` file, add the HTTP header as shown below.



```
GET / HTTP/1.1
Host: localhost:8080
User-Agent: Radamsa
Accept: */*
```

5. Next, start the PHP server using the command `php -S localhost:8080` within the same `www` directory.
6. Open a separate terminal session in your home directory and run the following command(s): `curl localhost:8080`. Do **NOT** close the first session running the PHP server!
7. In your first terminal session window, you should see the contents of the `index.html` file in the PHP server logs. This output verifies that the PHP server is up and running.

```
(kali㉿kali)-[~/www]
$ curl localhost:8080
<h1> Radamsa Network Service Test </h1>
```

8. In the same terminal where you ran the `curl` command, we will now use Radamsa and the known, good output from the `http-request.txt` file to send back to the PHP server. Use the command: ``radamsa -o 127.0.0.1:8080 http-request.txt -n inf``

```
(kali㉿kali)-[~/www]
$ radamsa -v -o 127.0.0.1:8080 http-request.txt -n 4
Random seed: 230460012069687009359010
- 127.0.0.1:8080/1: 71b
- 127.0.0.1:8080/2: 95b
- 127.0.0.1:8080/3: 103b
- 127.0.0.1:8080/4: 79b
```

9. Finally, view the output of the PHP server logs, and you'll see that it received the requests, but it will most likely not process them, as they were invalid/malformed.

```
(kali㉿kali)-[~/www]
$ php -S 127.0.0.1:8080
[Tue Nov 12 17:44:42 2024] PHP 8.2.24 Development Server (http://127.0.0.1:8080)
started
[Tue Nov 12 17:44:53 2024] 127.0.0.1:34280 Accepted
[Tue Nov 12 17:44:53 2024] 127.0.0.1:34280 [200]: GET /
[Tue Nov 12 17:44:53 2024] 127.0.0.1:34280 Closing
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54474 Accepted
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54474 Invalid request (Unexpected EOF)
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54474 Closing
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54490 Accepted
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54490 Invalid request (Malformed HTTP request)
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54490 Closing
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54502 Accepted
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54502 Invalid request (Malformed HTTP request)
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54502 Closing
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54516 Accepted
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54516 Invalid request (Malformed HTTP request)
[Tue Nov 12 17:45:03 2024] 127.0.0.1:54516 Closing
```

Radamsa can also be used to fuzz network client applications by intercepting responses from a network service and modifying them before the client receives them. The steps below will guide you in setting up Radamsa and Medusa in this manner.

1. First, you must acquire sample output from Medusa as input data. Use the command below to run Medusa through PostgreSQL against a target and save the response to a text

file. You can find the `password_list.txt` file under ``chadv1.0/fuzzing/password_list.txt``.

```
medusa -h 192.168.1.100 -u postgres -P password_list.txt -M
postgres -n 5432 > medusa_output.txt
```

2. Next, open a separate terminal session and run the command below. This command will set up Radamsa as a server that will send the fuzzed versions of Medusa's output in response.

```
radamsa -o :5432 medusa_output.txt -n inf
```

3. Finally, using the repetition script (``chadv1.0/fuzzing/repeat_medusa.sh``), run Medusa against the Metasploitable2 target VM. The script allows Medusa to run while Radamsa is running against it continually; Radamsa offers an "infinite" flag (`inf`), but Medusa does not.
4. The responses will be sample data fuzzed by Radamsa.

AFLnet and Masscan

The process for running AFLnet and Masscan is far more streamlined than the initial Radamsa and Medusa setup, as we do most of the configuration in the workflow script.

1. After running the workflow script, you should see a folder named ``chadv1.0``. In it, you'll see two folders - `attack_tools` and `fuzzing_tools`.
2. Navigate to the `fuzzing_tools/aflnet` directory: `cd fuzzing_tools/aflnet`.
3. Next, in the `aflnet` directory, we will create two folders: ``in`` and ``out``. Use the command(s): `mkdir in` and `mkdir out`. These folders are required for AFLnet to store known good commands for Masscan and the output of our fuzz testing results.
4. Next, you will need to get the MAC address of your ethernet/wi-fi adapter on your VirtualBox system. Type the command ``ip addr``. You should see a list of your available adapters on your system. Use the same adapter you configured previously with the Internal Network. You will find your MAC address to the right of the line starting with `link/ether`. If you have more than one and are unsure, refer to the [Network Configuration](#) section above.
5. Next, follow the commands below to put the Masscan commands in respective tests files in the `in` directory. *You will need to replace the `<MAC_ADDRESS>` with your adapter's MAC address!*

```
# Scan 1 File
echo "masscan -p21-8180 192.168.1.100 --banners --packet-
trace --source-mac <YOUR_MAC_ADDRESS>" > scan1.txt

# Scan 2 File
echo "masscan -p80,443 192.168.1.100 -banners" > scan2.txt

# Scan 3 File
echo "masscan -p1-65535 192.168.1.100 --rate=1000" >
scan3.txt
```

6. Finally, navigate to AFLnet's root directory (chadv1.0/fuzzing_tools/aflnet) and run the command below. This command will start AFLnet fuzzing on Masscan. *You will need to replace the <MAC_ADDRESS> with your adapter's MAC address!*

```
./afl-fuzz -t 1200 -i in -o out -N tcp://192.168.1.100/22 -P
SSH masscan -p21-8180 192.168.1.100 --banners --packet-trace
--source-mac <YOUR_MAC_ADDRESS>
```

Demonstration Video

You can find a demonstration video for the full Chad workflow, including the bash script and docker setup and installation and usage of each tool in the link below.



References

Attack Tools

- [Medusa](#)
- [Masscan](#)

Fuzzing Tools

- [AFLnet](#)
- [Radamsa](#)

Testbed Tools

- [VirtualBox](#)
- [Docker Desktop](#)
- [Kali Linux 2023.4 Pre-built VMs](#)
- [Ubuntu 20.04 LTS ISO](#)
- [Metasploitable2 VM](#)

Miscellaneous Tools

- [dos2unix](#)
- [MD to DOCX Converter \(Used to create User Guide from README\)](#)