Charger Active Shield v1.0 - User Guide

Noah Sickels, Adam Brannon, William Lochte – G12

Text

Description automatically generated with medium confidence

 [](https://www.python.org/downloads)  

## Table of Contents

* [Project Overview](#project-overview)
  + [System Overview Diagram](#_System_Overview_Diagram)
  + [Project Directory Structure](#project-directory-structure)
* [Project Prerequisites](#_Project_Prerequisites)
* [Testbed Configuration](#testbed-configuration)
  + [Network Configuration](#_Network_Configuration)
* [Fuzzing Workflow](#_Usage_&_Installation)
  + [Prerequisites](#_Prerequisites_–_Fuzzing)
  + [Installation](#_Installation_–_Fuzzing)
    - [Bash Script (Recommended)](#_Bash_Script_(Recommended))
    - [Dockerfile](#_Dockerfile_(WIP))
    - [Manual Installation](#_Manual_Installation)
  + [Usage](#_Usage_–_Fuzzing)
* [Active Defense Tool](#_Active_Defense_Tool)
  + [Prerequisites](#_Prerequisites_–_Active)
  + [Installation - Python Virtual Environments (Recommended)](#_Installation_–_Python_1)
  + [Usage – Active Defense Tool](#_Usage_–_Active)
  + [Key Features](#_Key_Features)
* [Demonstration Video](#_Demonstration_Video)
* [References](#references)

## Project Overview

Our senior design group is the second team working on the Charger Active Defense project. This project aims to create a fuzzing workflow that effectively tests the networking aspects of selected target applications. These target applications include Medusa, Masscan, and three different attack tools generated using the Phind and GitHub Copilot AI models. 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 involves the selection of fuzzing tools, two attack tools to fuzz, and the development of a fuzzing workflow. The active defense tool phase involves the development of a tool that can detect and respond to attacks on the network and send the fuzzed responses back to the adversary's tool.

At the sponsor's request, we have expanded the second phase to include generating three attack tools derived from the Phind and GitHub Copilot AI models. This addition will involve creating a banner-grabbing tool, an FTP password brute-force tool, and a simple, multi-threaded banner-grabbing tool from each model, comprising six tools.

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

Text

Description automatically generated

### System Overview Diagram

Diagram, schematic

Description automatically generated

A screenshot of a video game

AI-generated content may be incorrect.

### Project Directory Structure

#### **Highlights**

* **User\_Guide.docx:** User guide document for the project.
* **scripts/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.
* **AI\_Tools:** Contains the AI-generated attack tools.

## Project Prerequisites

*Combined list of all prerequisites required for the fuzzing workflow and active defense tool.*

* 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.
* Python 3.12 (or later)
* Apt Packages: clang, graphviz-dev, libcap-dev, git, make, gcc, autoconf, automake, libssl-dev, wget, curl, dos2unix, php-cli, wireshark, tshark, dos2unix, unzip.
* Pip Packages: rich, colorama, termcolor, pyshark, prompt\_toolkit, attrs, asyncio, nest\_asyncio, scapy.

## 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](https://www.kali.org/get-kali/#kali-virtual-machines).
* Rapid7 provides a pre-built Metasploitable2 VM from their website [here](https://www.rapid7.com/products/metasploit/metasploitable/).

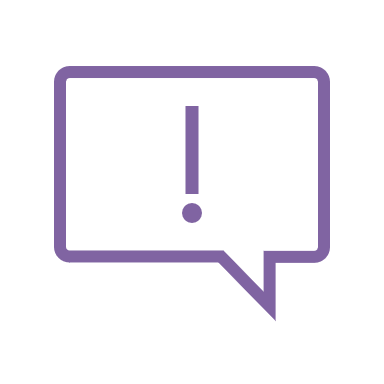
For detailed configuration information, please refer to the table below.

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 *(intent)* |
| 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 *(intent)* |
| Network Adapter 2 | *Optional* |
| 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 add` 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 add` 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.

## Fuzzing Workflow

The fuzzing workflow incorporates all the fuzz testing tools selected in the first phase of the project, specifically AFLnet and Radamsa. We designed this workflow to conduct comprehensive fuzz testing on network-based attack tools to identify any crashes or hangs that may occur, referred to as "active defense responses."

## Prerequisites – Fuzzing Workflow

* 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.
* Apt Packages: clang, graphviz-dev, libcap-dev, git, make, gcc, autoconf, automake, libssl-dev, wget, curl, dos2unix, php-cli, unzip.

## Installation – Fuzzing Workflow

We offer three methods for installing all the requirements for the fuzzing workflow: an automated bash script, Docker, and a manual installation process. We highly recommend using the automated bash script, as we have modified the source files of some attack tools to improve their integration with AFLnet. The script will automatically download the appropriate files from our repository.

### 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/chadv1.0/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.](https://www.docker.com/get-started/)



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

**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 – Fuzzing Workflow

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.

A screen shot of a computer

Description automatically generated

1. Next, start the PHP server using the command php -S localhost:8080 within the same www directory.
2. 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!*
3. 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.

A computer screen with text and numbers

Description automatically generated

1. 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`

A screenshot of a computer program

Description automatically generated

1. 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.

A screen shot of a computer

Description automatically generated

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 |

1. 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 |

1. 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.
2. 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 adapters available 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](#_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 |

1. 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> |

## Active Defense Tool

A screenshot of a computer

AI-generated content may be incorrect.

The active defense tool is a Python application that intercepts incoming network traffic from the attacking virtual machine (VM) to the victim and sends the appropriate active defense response back to the adversary. The program serves as a replay service to provide active defense capability to the victim.

### Prerequisites – Active Defense Tool

* Python 3.12 (or later)
* Apt Packages: wireshark, tshark.
* Pip Packages: rich, colorama, termcolor, pyshark, prompt\_toolkit, attrs, asyncio, nest\_asyncio, scapy.

### Installation – Python Virtual Environments (Recommended)

|  |
| --- |
| sudo apt install wireshark tshark  git clone <https://github.com/NCSickels/chadv1.0.git>  cd chadv1.0  python3 -m venv env  source env/bin/activate  pip install -r requirements.txt |

### Usage – Active Defense Tool

**Warning**

In order to properly use the live capture functionality on a network interface, you must run this with a user that has privileges to create raw packets, like ***root***. These capabilities can be granted by running the following command in your Python virtual environment: `*sudo setcap cap\_net\_raw=eip $(which python3)*`



|  |
| --- |
| # Ensure sudo user  sudo su  # (OPTIONAL-AS NEEDED) Grant raw packet privileges  sudo setcap cap\_net\_raw=eip $(which python3)  # Run the replay service in interactive mode  python3 chad.py -i  # OR run the replay service with command line arguments  python3 chad.py --start --interface *<INTERFACE\_NAME>* --target *<TARGET\_IP>* --port *<TARGET\_PORT>* |

## Key Features

* **Interactive Command Line Interface**: Custom interactive user interface that allows you to start/stop the packet capture, set target attributes (IP address, port, etc.), and view live packet details.
* **Live Packet Capture**: Utilizes the PyShark library to capture packets from the network interface and display them in real-time.
* **Logging**: Custom colored wrapper of the Python Logging library with user-defined message types for sent/received traffic and sent active defense responses. Logs packet details to a file for later analysis.
* **Active Defense Responses**: Automated responses to detected attacks, pulled from found crashes or hangs from AFLnet fuzz testing.

## 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](https://salsa.debian.org/pkg-security-team/medusa)
* [Masscan](https://github.com/robertdavidgraham/masscan.git)

### Fuzzing Tools

* [AFLnet](https://github.com/aflnet/aflnet)
* [Radamsa](https://gitlab.com/akihe/radamsa.git)

### Static Analysis Tools

* [LDRA Testbed](https://ldra.com)
* [Valgrind](https://valgrind.org)
* [Flawfinder](https://github.com/david-a-wheeler/flawfinder)

### Testbed Tools

* [VirtualBox](https://www.virtualbox.org/wiki/Downloads)
* [Docker Desktop](https://www.docker.com/get-started/)
* [Kali Linux 2023.4 Pre-built VMs](https://www.kali.org/get-kali/#kali-virtual-machines)
* [Ubuntu 20.04 LTS ISO](https://releases.ubuntu.com/focal/)
* [Metasploitable2 VM](https://www.rapid7.com/products/metasploit/metasploitable/)

### Miscellaneous Tools

* [dos2unix](https://dos2unix.sourceforge.io)
* [MD to DOCX Converter (Used to create User Guide from README)](https://cloudconvert.com/md-to-docx)