EDURange Student's Manual

September 14, 2015

A Introduction

This document will be updated as changes are made. EDURange is both a collection of interactive, collaborative cybersecurity exercises and a framework for creating these exercises. Currently, we have six exercises that are ready to use: Recon, ELF Infection, strace, fuzzing, scapy hunt, and DDOS. A DNS exercise will be released soon. ROP needs better documentation, and the Firewall exercise is being implemented. They can be done in any order, although scapy hunt and ELF infection are more advanced. Most of the exercises require a minimal level of understanding of some standard Linux tools.

Each of the exercises was designed to be expanded to have more levels.

- **Recon** is an exercise to examine how network protocols such as TCP, UDP, and ICMP can be used to reveal information about a network. Recon focuses on reconnaissance to determine hosts in an unknown network. The student can explore tradeoffs between speed and stealth when using tools such as nmap. More advanced levels will address intrusion detection and network monitoring.

- **Strace** (dynamic analysis of binaries) poses the challenge of understanding what a process is doing based on its system calls. Students learn to filter large amounts of data to distinguish between normal and anomalous behavior.

- **Elf infection** assesses the student's understanding of the structure of an executable file. The goal is to teach the student, having identified that a program is doing something malicious, where that code has been injected and how it works. This is a reverse engineering problem and can use a range of tools, including readelf, objdump, gdb, strace and netstat.

- **Fuzzing** is an attack-defense exercise where the defender must correctly implement a user interface that filters malformed expressions. In the simplest version, the interface is for a calculator, and the defender must implement both the interface and the application.

- **Scapy Hunt** poses the challenge of analyzing network traffic to understand who is communicating with whom and how. The player is trying to get data from an ftp server which is not on the same subnet, but one of the hosts on its network is communicating with it. By default the player can only see packets sent to the server and must craft packets to get them routed to the target and get a response back.

- **ROP** needs better documentation. It teaches return-oriented programming, and requires a decent understanding of x86 assembly language. The student will learn how the PLT (procedure lookup table) and the GOT (global offset table) work and how to find their addresses in an executable file.

- **Firewall** is still being implemented. It requires students to understand the interactions between multiple firewalls which have different rule sets.

The most important learning goal is developing the analytical skills of understanding complex systems and complex data, i.e. understand how the relevant system works. For example, the standard tool for Recon is nmap, and while it is necessary to learn how to use that tool in order to do this exercise, that is not the most important learning goal. Similarly, the Elf Infection exercise uses standard tools such as netstat but requires that students reason about the behavior of a complex system to discover which binary is infected and what it is doing. Scapy hunt is about listening passively to discover hosts on the local network and which other hosts they are talking to. The strace exercise uses the strace utility, but it is about how to look at the system calls that a process makes in order to understand what the process might be doing.

Using EDURange: AWS

Your instructor will give you an access code that you can use to register for EDURange. You will register at `http://cloud.edurange.org`. You will choose a password for your account on cloud.edurange.org. When your instructor creates an exercise and places you in a group, you should receive an e-mail with a temporary username and password for that exercise. You can use that to connect via ssh to the public IP address of a gateway for that exercise. In some cases, you may need to ssh from the gateway to a team VM using the same credentials.

B Exercises

B.1 strace

Background

One of the important skills that is part of cyber security is being able to analyze malware. These skills overlap with debugging, except that the problems can be more subtle. This exercise focuses on dynamic analysis of programs, i.e. analyzing what a program does while it is running. It turns out that in order to do anything, a program or process relies heavily on the operating system. The system call (syscalls) can reveal a lot about what is program is doing. One of the tools for examining the syscalls is strace. You should first figure out where the strace binary is located and what some of the options are (look at the man pages). You will start with whitebox testing of some programs for which you have the source code. Then, you can move to blackbox testing using trace files. The trace files that you will analyze are in the /tmp/ directory on the AWS Linux VM that you are assigned to.

Then, you will read through the traces, read the man pages for many of the system calls, and try to figure out which utility corresponds to each trace you examine. You can verify your guess by stracing a given utility. The last strace in this example has two executables running. If you are working in a group, think about how you can divide up the work in an efficient way.

Laboratory Assignment and Questions

Q1: Your home directory contains various files that will be used in this scenario. One is the file empty.c, whose contents are:

```
int main () {}
```

Compile this program as follows:

```
gcc -o empty empty.c
```

Now run strace to execute the empty program:

```
strace ./empty
```

What do you think the output of strace indicates in this case? How many different syscall functions do you see?

Q2: The -o option of strace writes its output to a file. Do the following:

```
strace -o empty1 ./empty
strace -o empty2 ./empty
diff empty1 empty2
```

Explain the differences reported between traces empty1 and empty2.

Q3: Study the program copy.c.

```
# include <stdio.h>
# include <stdlib.h>

int main (int argc, char** argv) {
  char c;
  FILE* inFile;
  FILE* outFile;
  char outFileName[256];
  if (argc != 3) {
    printf("program usage: ./copy <infile> <outfile>\n");
    exit(1);
  }
  snprintf(outFileName, sizeof(outFileName), "%s/%s", getenv("HOME"), argv[2]);
  inFile = fopen(argv[1], "r");
  outFile = fopen(outFileName, "w");
  printf("Copying ``%s to %s\n", argv[1], outFileName);
  while ((c = fgetc(inFile)) != EOF) {
fprintf(outFile, "%c", c);
  }
  fclose(inFile);
  fclose(outFile);
}
```

Compile it to an executable named copy, and use strace to execute it as follows:

```
strace ./copy tiger.txt mytiger.txt
```

Explain the non-boilerplate parts of the trace by associating them with specific lines in copy.c.

Q4: The file strace-identify was created by calling strace on a command. The first line of the trace has been deleted to make it harder to identify. Determine the command on which strace was called to produce this trace.

Q5: The file mystery is an executable whose source code is not available. Use strace to explain what the program does in the context of the following examples:

```
./mystery foo abc
./mystery foo def
./mystery baz ghi
```

Q6. Create a one-line “secret” file. Heres an example, though of course you choose something different as your secret:

```
echo "My phone number is 123-456-7890" > secret
```

Now display the secret to yourself using cat:

```
cat secret
```

Is your file really secret? How much do you trust the cat program? Run strace on cat secret to determine what its actually doing Can other students read your secret? Can you read the secrets of other students? Explain everything you observe.

Q7. Here is a simple shell script in script.sh:

```
#!/bin/bash
echo "a" > foo.txt
echo "bc" >> foo.txt
echo `id -urn` >> foo.txt
chmod 750 foo.txt
cat foo.txt | wc}
```

Compare the outputs of the following calls to strace involving this script. Explain what you see in the traces in terms of the commands in the script.

```
strace ./script.sh
strace -f ./script.sh
```

Q8. Sometimes strace prints out an overwhelming amount of output. One way to filter through the output is to save the trace to a file and search through the file with grep. But strace is equipped with some options that can do some summarization and filtering. To see some of these, try the following, and explain the results:

```
strace find /etc/pki
strace -c find /etc/pki
strace -e trace=file find /etc/pki
strace -e trace=open,close,read,write find /etc/pki
```

Discussion Questions

1. what are the major types of syscalls? Which ones would you look for when black box testing?
2. Explain how you would diguise a rootkit that copies a file to a hidden directory.
3. Explain how you would disguise a rootkit that opens a reverse shell.

B.2 Recon I

Background

What are TCP and UDP?

In order to understand this exercise, you should be familiar with the 3-way handshake for TCP. You should also know something about ICMP and UDP. This exercise is not designed to teach you all of the details of those protocols, but it will teach you why some of those details are important. You can read about them in any standard networking textbook.

You will learn in this exercise, how these and other protocols can be used to discover hosts on a network, which ports on those hosts are open, and what applications are running on them. In practice, each message that is sent over the Internet uses multiple protocols, which are divided into five layers: physical layer, link layer, network layer, transport layer and application layer. For example, the physical layer handles what is encoded as a 0 or 1. The link layer handles communication on local area networks (LANs). The network layer handles routing on wide area networks (WANs), e.g. IP. The transport layer handles ports and processes, e.g. TCP, UDP, ICMP. The application layer handles applications communicating with each other, e.g. http, ftp, by nesting packets inside of packets. In general, these packets correspond to layers of functionality: TCP is connection-oriented and is responsible for a number of things including reliably conveying messages between the application layers on two hosts. The three-way handshake establishes this pairing with the following sequence: SYN, SYN-ACK, and ACK You can get a summary of the important protocols and their layers in: Chapter 4 of *Hacking: The Art of Exploitation* (Erickson)[1] or Chapter 2 of *Counter Hack Reloaded* [2]. *Network Security* by Kaufman, Perlman, Speciner [3]

Learning Outcomes for Recon I

Answer the following questions (after completing the lab):

1. what is the 3-way handshake?
2. what does the SYN flag do?
3. What does 10.1.1.0/17 mean? how many IP addresses does that include?
4. What are the options for nmap and what are their differences in terms of time, stealth and protocols?
5. which ports does nmap -sT scan?
6. which ports does nmap -sU scan?

Laboratory Assignment

1. The first goal is to locate hosts on the target network using a TCP scan. What flags do you set to do this? Use tcpdump or tshark to list the SYN and SYN-ACK packets. What ports are scanned? Is there a pattern? How many packets are sent? What is the scan time?
2. How many IP addresses are there in the target network? How would you divide up the address space into two equal parts? Divide the recon task into 2 parts and have a different person scan each part.
3. Try a ping scan. What flags did you use? Do the same hosts show up? How many ports are scanned? What is the scan time?

4. Try a UDP scan. What flags did you use? Do the same hosts show up? How many ports are scanned? What is the scan time?
5. Make your scan faster. What flags did you use?
6. Make your scan more stealthy. What flags did you use?
7. How can you increase the speed of your scan?

Diagram of Recon I network

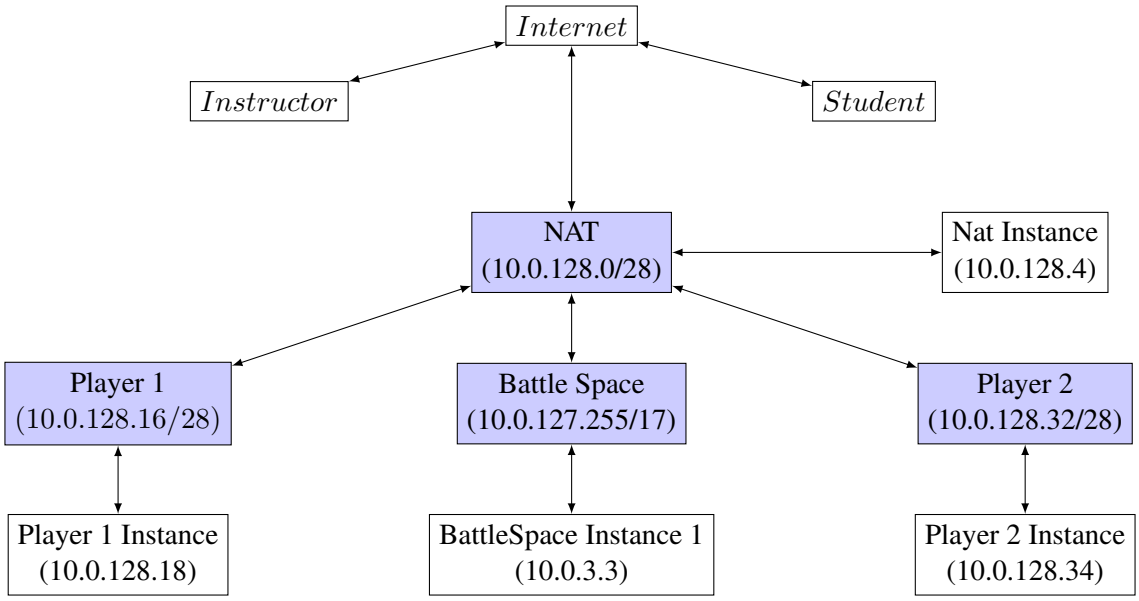

Figure 1: Conceptual diagram of the Recon I game. Note subnets are blue, and IP addresses are just as an example.

Discussion Questions

1. What are some ways to maintain stealth?

B.3 Elf Infection

The Elf infection exercise only uses a single VM for each team.

B.4 Scapy Hunt

C References

[1] Hacking: The Art of Exploitation(Chapter..)
[2] Counterhack Reloaded (Chapter 2)
[3] man pages for nmap

contributors

Stefan Boesen, Erin Davis, Michael Locasto, Jens Mache, Lyn Turbak, Richard Weiss,