EDURange Student's Manual

March 6, 2017

Table of Contents

Introduction

EDURange is an NSF-funded project that is both a collection of interactive, collaborative cybersecurity exercises and a framework for creating these exercises. This suite of exercises is intended to help supplement classroom lectures, labs, and other activities.

We believe EDURange has a potential to significantly advance the integration of cybersecurity into the undergraduate computer science curriculum. By providing interactive, competitive exercises, it will enhance the quality of instructional material, and increase active learning for students. The ease of use for instructors will encourage them to integrate cybersecurity into the current core curriculum. These exercises will also provide rapid feedback to students and faculty, which will aid in assessment of student learning.

A key question we ask ourselves in our design process is: *What kind of analysis skill do we expect the students to acquire from running (and re-running) this scenario*?
Our central and very intense focus is on creating exercises that support and nurture the development of analysis skills rather than memorized scripts, recipes, or standard command line and GUI settings for a particular tool. Though some exercises revolve around using a specific tool, the main learning goal is the development of the analytical skills and understanding of the complex system which that tool acts upon.
By keeping this focus, EDURange helps students buy into the process of sharpening their information security analysis skills and makes them a partner in evaluating and understanding the limits of those skills.

Our cybersecurity exercises cover the topics of Network Analysis and Reconnaissance, Malware Detection and Analyzation, Network Traffic Analysis and Defense, Social Engineering, and Web Security. Though our exercises can be done in any order, some can be good building blocks that work towards the more advanced ones. Most of the exercises require a minimal level of understanding of some standard Linux tools. We have provided some basic tutorials for Linux use at the end of this document.

Using EDURange

Your instructor will give you an access code that you can use to register for EDURange. You will register at http://cloud.edurange.org. When your instructor creates an exercise and places you in a group, it will be accessible to you in the scenarios section of your EDURange home page. From there you will be able to see your login credentials as well as your initial instructions. You can use that to connect via ssh to the public IP address of a gateway for that exercise. Some exercises will also have questions to be answered via the EDURange scenario page. Others have questions listed in this manual that your instructor might have separate instructions for.

Exercises

Strace

Description

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

Background

One of the important skills 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 the 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. When reading through the traces, you will first need to figure out what the system calls are doing. The system calls also have man pages. 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.

Learning Objectives

- Know how to analyze the sequence of sys calls and recognize patterns
- Be able to determine if a program is behaving as expected
- Recognize when a process forks another process
- Recognize when a process opens a file or socket
- Recognize when a process deletes a file
- Recognize which system calls introduce threats and how that happens

Instructions

Connect to the VM via your instructor’s directions, or as displayed on your EDURange account. Follow the Lab Assignments and Questions section below.

It may be helpful to look over the Grep and Piping and Redirecting portions of the Student Tutorials section. They can help you filter through the results of strace.

Lab Assignments and Question

1. 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?

2. The file hello.c contains this simple program:

```
# include <stdio.h>
int main () {
printf("hello\n");
}
```

Compile hello.c to hello and execute it with strace:

```
gcc –o hello hello.c
strace ./hello
```

Compare the output of strace for empty and for hello. Which part of the strace output is boilerplate, and which part has to do with the specific program?

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

```
strace -o empty-trace ./empty
strace -o hello-trace ./hello
diff empty-trace hello-trace
```

Explain the differences reported between traces empty-trace and hello-trace. (Colordiff is installed as well.)

4. 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:

```
gcc -o copy copy.c
strace ./copy tiger.txt mytiger.txt
```

Explain the non-boilerplate parts of the trace by associating them with specific lines in copy.c. Are there any lines from the program that you expect to show up in the trace that don't?

5. 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.

6. 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:

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

7. 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
chmod 644 foo.txt
```

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
```

8. 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
```

9. Create a one-line “secret.txt” file. Here’s an example, though of course you should choose something different as your secret:

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

Now display the secret to yourself using cat:

```
cat secret.txt
My phone number is 123-456-7890
```

Is your secret really secret? How much do you trust the cat program? Start by running strace on cat secret.txt to determine what it's actually doing. Based on this and subsequent experiments, determine answers to the following questions:

- The cat program in the strace scenario does more than display the contents of a file? Exactly what else does it do?
- How can you display the contents of a file without the extra actions reported above?
- Can anyone else read your secret?
- Can you read the secrets of anyone else?
- How do you think the trojaned cat program was implemented? How do you think it was installed? Justify your explanations

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 disguise a rootkit that copies a file to a hidden directory.
3. Explain how you would disguise a rootkit that opens a reverse shell.

Ssh Inception

Description

Ssh Inception teaches the basics of ssh, a secure program for logging into a remote machine. You will use ssh along with some other tools to navigate through a series of network checkpoints.

Background

Logging into a remote machine is one of the most basic computer networking tasks. Knowing the different methods and options for doing so is essential. This exercise will also introduce you to some other helpful tools as you navigate through the checkpoints, including grep, ifconfig, nmap and ftp. Though you will only use them briefly here, they are each powerful tools on their own that you should investigate further. Reading the man pages of these tools is the first place to start if you are unsure how to tackle a problem. For nmap, be sure that you only use it to scan the local network, which starts with 10.0.0.

For grep, you can search through a large number of files with one command:

```
grep 1234 *.txt
```

Learning Objectives

- Understand what is happening when you ssh
- Understand key pairs and why they provide more protection than passwords
- Have a basic familiarity with a linux system

Instructions

Connect to the VM via your instructor’s directions, or as displayed on your EDURange account. Instructions will be displayed upon logging in and at each new checkpoint.

Lab Assignments and Question

Questions can be found upon logging into your EDURange account.

Discussion Questions

1. Why did you see the following message when you used ssh the first time to connect to the NAT?

> The authenticity of host [IP address] can't be established.
> ECDSA key fingerprint is SHA256:[hash value].

2. What were some of the different ways each network limited or allowed a user to login?
3. How were you able to get access to the loose ftp server? Was there another way to gain access? What could be done to secure the ftp server?
4. How was the file ‘betcha_cant_read_me’ decoded? What are some other similar methods?
5. The final problem in this exercise is a bit challenging and will require some creative thinking. Why was it difficult to stay in Satan’s Palace? There are multiple ways to get what you are looking for. Share with classmates how you achieved your goal. Do you understand why these different methods worked?

Total Recon

Description

Total Recon is a progressive, story-based game designed to teach how network protocols such as TCP, UDP, and ICMP can be used to reveal information about a network. Total Recon focuses on reconnaissance to determine hosts in an unknown network. You will explore tradeoffs between speed and stealth when using tools such as nmap.

Background

Whether you’re doing a large-scale security audit, inventorying a network, or analyzing network response times, nmap is a powerful tool to help you complete your task. In order to understand this exercise, you should be familiar with the 3-way handshake for TCP. A basic understanding of ICMP and UDP will also be helpful. This exercise is not designed to teach you all of the details of those protocols, but rather to show you how they can be used for network exploring. You will learn how to discover hosts on a network, determine 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 Objectives

- Understand how the networking protocols (TCP, UDP, ICMP) can be exploited for recon
- Know how to use nmap to find hosts and open ports on a network
- Recognize the standard common ports (e.g. SSH, FTP, HTTP, SMTP, IMAP)
- Understand the TCP flags and how they can be used for different types of scans
- Understand CIDR network configuration and how to subdivide a network IP range

Instructions

Connect to the VM via your instructor's directions, or as displayed on your EDURange account. Instructions will be displayed upon logging in and at each new checkpoint.

Lab Assignments and Question

Questions can be found upon logging into your EDURange account.

Discussion Questions

1. What is the 3-way handshake?
2. What does 10.1.1.0/17 mean? how many IP addresses does that include?
3. What does the SYN flag do? What does the FIN flag do?
4. What are the options for nmap and what are their differences in terms of time, stealth and protocols?
5. Which methods did you use to speed up your scans? What else could you have done?

ELF Infection

Description

ELF Infection is an exercise to assess your understanding of the structure of an executable file. The goal is to teach you, 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.

Background

One of the first things most people think of when it comes to digital security are viruses! Being able to detect and disable malware is a never ending job as the digital world continues to

expand. The first step towards recognizing viruses is being able to understand what is normal behavior for a program. If you are unfamiliar with strace, and reading system calls, you might want to do our strace exercise before attempting ELF Infection.

As the title suggests, you will be examining the different ways ELF files can be infected. The files in this exercise were infected with a method called injection. 'Injection' is the process of inserting and smuggling a malicious payload into an ELF executable without breaking the executable's integrity.

Learning Objectives

- Know the capabilities of `readelf` and how to use the basic options
- Know the format of an ELF file header
- Know which system calls do the following -
    - Make a new name for a file
    - Execute a process
    - Terminate the calling process
    - Create message buffer and read from the message queue
    - Assign the local IP address and port for a socket
- Be familiar with the general classes of system calls
- Be able to read a system trace and know what is normal vs abnormal
- Be able to make a system call in C
- Be able to make a system call in x86 assembly
- Understand how the kernel handles system calls
- Understand how some system calls introduce threats
- Understand how errors are handled

Instructions

Connect to the VM via your instructor’s directions, or as displayed on your EDURange account.

Lab Assignments and Question

UNDER CONSTRUCTION

Discussion Questions

UNDER CONSTRUCTION

Treasure Hunt

Description

Treasure Hunt is an exercise that teaches about permissions and other security loopholes in Linux. In this virtual machine there are 16 imaginary users. Somewhere in his/her home directory, each of these imaginary users has a “secret” file named username-secret.<ext> (where <ext> is a file extension) whose contents are intended to be private (readable only by the user and no one else). However, each of their secret files can actually be read by other users who are both determined and clever. Your goal is to collect the contents of as many of the sixteen secret files as you can.

Background

There are often multiple users on the same system or network. Given this case, how does a system determine who is able to access specific files? Linux system of file access permissions are used to control who is able to read, write and execute certain files. This is used both to keep user files private as well as to protect critical system files. In order to obtain many of the secrets in this exercise, you will need to understand the read, write, and execute permissions as well as how permissions are applied to the owner, group owner, and every user. If you are unfamiliar with linux permissions, see the section on Linux File Permissions in the Student Tutorials section below.

This exercise also utilizes password cracking for a few users. That password cracking method that you will work with utilizes linux password hashes. This exercise is not intended to teach about hashes and password security techniques. If you are unfamiliar with the general idea of them, a quick web search should catch you up with the basics. The files that contain the password hashes are not publicly available on linux systems, but we have made them so for this exercise and will show where to find them. Hopefully, this will give you an idea if the passwords you use are secure or not!

You will also run into the .htaccess file in this exercise. This is a configuration file for Apache Web Server. It is used for many things but here it is only used from user authorization. You should be able to figure it out when you come across it. If not, a simple web search will help you out again.

Learning Objectives

- Know the difference between read, write, and execute permissions and how this affects directories and files
- Understand linux groups
- Understand what Set User ID and Set Group ID do
- Know how to find a file's permissions and interpret this and similar lines '-rw**s**r-xr-x'
- Be able to create a symbolic link and know what it does
- Recognize what sorts of passwords are easily cracked from known password hashes
- Have a moderate understanding of some basic linux tools and how to use them

Instructions

Connect to the VM via your instructor's directions, or as displayed on your EDURange account.

Once logged in, it is your goal to find the secrets of the following 16 fake users:

Alice Wan (awan)
Bob Duomo (bduomo)
Cathy Dry (cdry)
Debbie Shi (dshi)
Ellen Quintus (equintus)
Fred Sexon (fsexon)
George Hepta (ghepta)

Helen Ochoa (hochoa)
Inna Nunez (inunez)
Jack Dekka (jdekka)
Karen Elva (kelva)
Loretta Douzette (ldouzette)
Patricia Kaideka (pkaideka)
Pyotr Theodore Radessime (pradessime)
Quinn Sanera (qsanera)
Tudor Daforth (tdaforth)

Each secret is contained somewhere in that user's home directory. All fake users belong to a group named student, a fact that is important for some of the attacks. There are other significant groups as well that some of these users are in.

There is no strict sequential order for finding the secrets, though some you will only be able to get after gaining access to another user's account. Password cracking is a great place to start. We will walk you through that below.

Accessing some secret files will require that you make changes to certain files/directories in the accounts of the fake users. Once you determine the secret, be sure to undo any changes that you make so that you leave the system exactly in the same state that you found it. Otherwise, you could (1) make it very easy for others to access the information you worked so hard to get or (2) make it impossible for others to access the information you found (this is unacceptable in this exercise, though not in the real world).
Since some of your changes may be hard for you to undo, you can use the resetFakeUsers command to resets all fake user accounts to their initial states and also resets other parts of the system (e.g. deletes all files in the /tmp directory). Executing this command should solve all reseting issues; if it does not, please let us know. By calling resetFakeUsers frequently, you could cause a denial of service attack against your classmates; please do not do this!
(Note: One case has been found where resetFakeUsers does not work. This is after finding a particular secret, so you should be able to figure out what is necessary to make it work again.)

Password Cracking:
For password cracking download John the Ripper from http://www.openwall.com/john/ onto a local computer. John the Ripper is not on the Treasure Hunt VM, and you won't be able to install it there. If you only have access to a Windows computer for your local machine, John the Ripper suggests HashSuite; though we won't provide you with instructions on how to use that program.

On the Treasure Hunt machine, gain access to the file /etc/shadow. (See hints below if stuck). You will need a copy of /etc/shadow and /etc/passwd on your machine running John the Ripper. Use John’s unshadow command to combine /etc/passwd and /etc/shadow into a single password file (e.g. 'unshadow passwd shadow > mypasswd').
Manually edit 'mypasswd' to exclude all accounts other than the 16 fake users for this problem – otherwise you’re wasting processing time in your password cracker. When you find the secret of

a fake user, removing that user from the unshadowed file will help speed up future attempts. You do not want to waste processing time trying to crack passwords you don't need!
Run John on 'mypasswd' (e.g. 'john passwords'). The basic john command uses the default wordlist run/password.lst, which should be able to fairly quickly crack two user passwords. There is one more password that can be cracked, but you will need to feed john a custom word list. Maybe if you knew more about fake users...

User Web Pages:
Each user has at least one web page in a public html directory. Some of these pages contain information relevant to finding their secret. Although many of the user web pages are publicly readable by any user on the THVM, some can only be read via a web browser. Since you are logged in via ssh, you might be wondering how you can view these web pages. Lynx is a text-based web browser that we have provided for your use. Typing 'lynx localhost/~awan/' will let you view awan's homepage. The same format can be used to view the other 15 user's pages. Though you can see the public html pages in each user's directory, due to the permissions of any private files, you will need to use Lynx to uncover some of the secrets. See Lynx's man page for specific instructions. (w3m might also be installed, but the default version might not help you with every secret).

Hints:
- It may be helpful to export certain files from the THVM to your local computer (or vice versa). You can use scp or ftp from your local computer to do this.
- Having trouble gaining access to /etc/shadow? Look in /bin/ and see if you can find something to help you.
- Access to web directories can be controlled by a .htaccess file. See http://www.javascriptkit.com/howto/htaccess.shtml for documentation on .htaccess files.
- The web server runs as user/group www-data. Including www-data in a group gives the web server whatever permissions are given to the group. There is a group named apache whose only member is www-data.
- In an HTML file, text between <!-- and > is a comment that is not displayed by the web browser.
- The ghostview suite is installed. The gv command can be used to display .pdf files. (Be patient; it is very slow when displaying a window remotely. Alternatively, you may want to export relevant .pdf files from the THVM to your local computer and view them there.)
- It is possible to convert .pdf files to other file formats, and there are programs in the THVM for doing this. But saying exactly what those programs are would make one secret too easy too find. So you might want to research how to convert .pdf files to other formats in Linux.
- .docx format is a zipped (compressed) directory of XML files; it can be uncompressed with the unzip command. There are many ways to obtain that secret though.
- If you want to add a directory dir to the front of your PATH variable, a good way to do this is with the following command - export PATH= dir :$PATH (e.g. 'export PATH=/tmp/:$PATH')
- The strings command could be helpful for some secrets. As well as a hex viewer.

Lab Assignments and Question

UNDER CONSTRUCTION

Discussion Questions

UNDER CONSTRUCTION

Scapy Hunt

Description

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.

Background

UNDER CONSTRUCTION

Learning Objectives

UNDER CONSTRUCTION

Instructions

UNDER CONSTRUCTION

Lab Assignments and Question

UNDER CONSTRUCTION

Discussion Questions

UNDER CONSTRUCTION

Student Tutorials

UNDER CONSTRUCTION