

# EDURange Student Manual

Stefan Boesen  
Jens Mache

Erin Davis  
Lyn Turbak  
James Newman

Michael Locasto  
Richard Weiss

July 2019

# Contents

<b>Introduction</b>	<b>3</b>
<b>Using EDURange</b>	<b>4</b>
<b>Scenarios</b>	<b>5</b>
SSH Inception . . . . .	5
Description . . . . .	5
Background . . . . .	5
Learning Objectives . . . . .	5
Lab Assignments and Question . . . . .	6
Discussion Questions . . . . .	6
Strace . . . . .	6
Background . . . . .	6
Learning Objectives . . . . .	7
Instructions . . . . .	7
Lab Assignments and Question . . . . .	7
Discussion Questions . . . . .	9
Total Recon . . . . .	10
Background . . . . .	10
Learning Objectives . . . . .	10
Instructions . . . . .	11
Lab Assignments and Question . . . . .	11
Discussion Questions . . . . .	11
Getting Started . . . . .	11
Terminal . . . . .	12
Linux File System . . . . .	12
Commands . . . . .	14
Man Pages . . . . .	16
File Types . . . . .	17
Case Sensitivity / touch / echo / Angle Brackets . . . . .	18
vim, Regular expressions and find . . . . .	20
More Commands . . . . .	22
Final Mission . . . . .	23
Treasure Hunt . . . . .	23

	Description . . . . .	23
	Background . . . . .	24
	Learning Objectives . . . . .	24
	Instructions . . . . .	24
	Lab Assignments and Question . . . . .	27
	Discussion Questions . . . . .	27
ELF	Infection . . . . .	27
	Description . . . . .	27
	Background . . . . .	28
	Learning Objectives . . . . .	28
	Instructions . . . . .	28
	Lab Assignments and Question . . . . .	28
	Discussion Questions . . . . .	29
	Description . . . . .	29
	Background . . . . .	29
	Learning Objectives . . . . .	29
	Instructions . . . . .	29
	Lab Assignments and Question . . . . .	29
	Discussion Questions . . . . .	29

# 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.

# Scenarios

## 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?

## Strace

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: 

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

 Compile this program as follows: 

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

 Now run strace to execute the empty program: 

```
sh  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: 

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

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

```
sh  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:  
`sh 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`.  

```
“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");
}
```

```
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:sh 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: `sh 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: 

```
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.  

```
sh 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: 

```
sh ./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: 

```
sh 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.

## Total Recon

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?

## Getting Started



Figure 1: Welcome to CyberSec

Congratulations recruit and welcome to CyberSec. As you should know we provide security services to you, our communities, and small businesses. You've made it this far, I suppose that means you may be helpful. But first comes the

training. We need to strengthen your skills before you can truly be of use. Go through each tab to the left and at the end we will test what you have learned. Remember, this world is being controlled by the malicious corporations and individuals and we must learn to protect ourselves from their invasive attacks. We must protect our data. This is why you are here. To protect yourself, your company and your community. And it all starts here, at the beginning, with a terminal command.

## Terminal

When using your computer (Mac, Linux, Windows) you typically are using a GUI (Graphical User Interface). It's a pretty representation of how your computer works. To really get into the 'guts' of your computer and to really learn how to control it we will learn how to use the terminal. The terminal is a text based representation of your computer (rather than graphical). Learning to use the terminal will help you along your path to protecting your community and your self.

Why should you learn the command line?

- You gain greater control over the system (computer)
- A GUI interface just doesn't have the power needed to run repetitive tasks.
- Doing anything from a simple task of renaming a file, changing a user information and searching for files is faster and easier through command line once it is learned.
- Scalability
- Scriptability
- Simple design
- Simple interface
- Stable design

In summation it allows you to do stuff faster than GUI and provides an amazing automation support built in.

Open up terminal for the rest of this training. Or `ssh` into the server for the command line experience.

## Linux File System

### Hierarchy

Linux folders and files are arranged like an upside down tree, where the slash / is called the root, or beginning, of all your files in the entire computer. The root is the base of the tree and as you go down it keeps splitting into branches and leaves. The leaves would be a file and the branches are folders.



Figure 2: How to start terminal on the Ubuntu operating system.

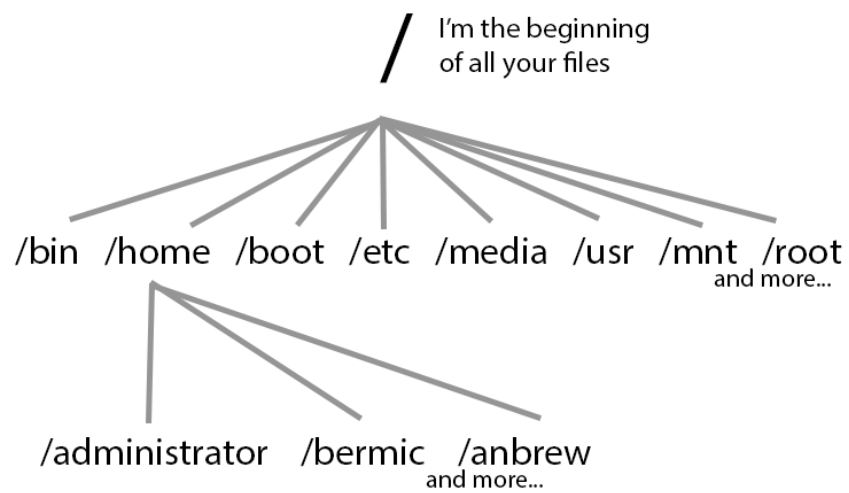


Figure 3: A visualization of the linux filesystem hierarchy

## **/ vs. logging in as root**

The root, signified by a / , is the beginning of your files. But you can also log in as the root user. When you do this, your home directory (where your files are typically saved) is in the folder /root not at, /. The /root folder is not to be confused with the slash (root) the beginning of all the files. Just like if you were logged in as bermic you would typically save your files in /home/bermic, whereas the root user saves their files in /root.

**Important**, a root user is someone who has access to everything on the computer. They could even delete everything in a computer. It is best practices to disable root or use a VERY strong password. For example using numbers, letters, capitals, special symbols and a random sampling of each, and no dictionary words.

## **Commands**

What is a command? A command is something you type into your terminal to make something happen. That's a bit vague but essentially the idea. You can do a lot of things with commands. Let's get started right away and use two different commands. Let's go to your home directory.

### **cd**

Type each of the following. One at a time. Hitting enter after each

```
~$ cd /  
~$ cd /root  
~$ cd  
~$ cd ../
```

The first command sends you to the root of your entire file system.

The second command sends you to the user root folder

The third command sends you to your home directory

The fourth sends you backwards (up) a level.

TASK: cd to /bin then cd back to your home directory.

### **pwd**

Now type in

```
~$ pwd
```

Hit enter, `pwd` is a command to print your working directory. In other words, it prints your location so you can see where you are. You should see something similar to `/home/yourusername`. To learn more type `man pwd`, then to get out of that page type `q`

`ls`

Type

```
~$ ls
```

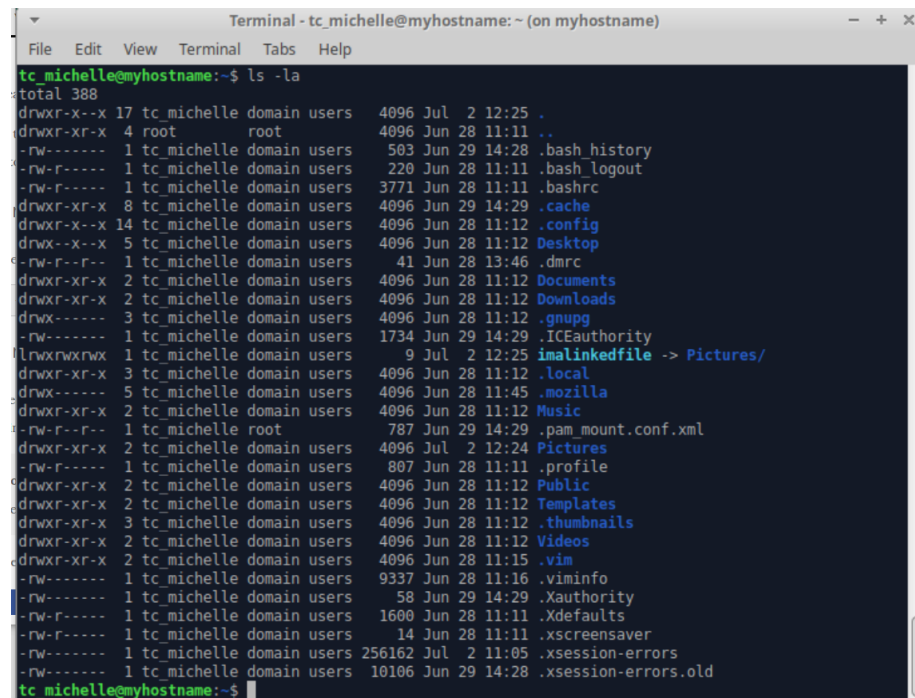
Hit enter, `ls` lists the files and directories of where you are now.

There are parameters and options you can give a command. What if you wanted to list the permissions of a file and find hidden files? (Yes there are hidden files!)

Now type

```
~$ ls -la
```

Then hit enter.

A terminal window titled "Terminal - tc\_michelle@myhostname: ~ (on myhostname)" showing the output of the command `ls -la`. The output lists files and directories with their permissions, owner, group, size, date, and name. The files are sorted by date. The output is as follows:

```
total 388
drwxr-x--x 17 tc_michelle domain users 4096 Jul  2 12:25 .
drwxr-xr-x  4 root                4096 Jun 28 11:11 ..
-rw-r----- 1 tc_michelle domain users  503 Jun 29 14:28 .bash_history
-rw-r----- 1 tc_michelle domain users  220 Jun 28 11:11 .bash_logout
-rw-r----- 1 tc_michelle domain users 3771 Jun 28 11:11 .bashrc
drwxr-xr-x  8 tc_michelle domain users 4096 Jun 29 14:29 .cache
drwxr-x--x 14 tc_michelle domain users 4096 Jun 28 11:12 .config
drwx--x--x  5 tc_michelle domain users 4096 Jun 28 11:12 Desktop
-rw-r--r--  1 tc_michelle domain users  41 Jun 28 13:46 .dmrc
drwxr-xr-x  2 tc_michelle domain users 4096 Jun 28 11:12 Documents
drwxr-xr-x  2 tc_michelle domain users 4096 Jun 28 11:12 Downloads
drwx----- 3 tc_michelle domain users 4096 Jun 28 11:12 .gnupg
-rw-r----- 1 tc_michelle domain users 1734 Jun 29 14:29 .ICEauthority
lrwxrwxrwx  1 tc_michelle domain users   9 Jul  2 12:25 imalinkedfile -> Pictures/
drwxr-xr-x  3 tc_michelle domain users 4096 Jun 28 11:12 .local
drwx----- 5 tc_michelle domain users 4096 Jun 28 11:45 .mozilla
drwxr-xr-x  2 tc_michelle domain users 4096 Jun 28 11:12 Music
-rw-r--r--  1 tc_michelle root        787 Jun 29 14:29 .pam_mount.conf.xml
drwxr-xr-x  2 tc_michelle domain users 4096 Jul  2 12:24 Pictures
-rw-r----- 1 tc_michelle domain users  807 Jun 28 11:11 .profile
drwxr-xr-x  2 tc_michelle domain users 4096 Jun 28 11:12 Public
drwxr-xr-x  2 tc_michelle domain users 4096 Jun 28 11:12 Templates
drwxr-xr-x  3 tc_michelle domain users 4096 Jun 28 11:12 .thumbnails
drwxr-xr-x  2 tc_michelle domain users 4096 Jun 28 11:12 Videos
drwxr-xr-x  2 tc_michelle domain users 4096 Jun 28 11:15 .vim
-rw-r----- 1 tc_michelle domain users 9337 Jun 28 11:16 .viminfo
-rw-r----- 1 tc_michelle domain users  58 Jun 29 14:29 .xauthority
-rw-r----- 1 tc_michelle domain users 1600 Jun 28 11:11 .Xdefaults
-rw-r----- 1 tc_michelle domain users  14 Jun 28 11:11 .xscreensaver
-rw-r----- 1 tc_michelle domain users 256162 Jul  2 11:05 .xsession-errors
-rw-r----- 1 tc_michelle domain users 10106 Jun 29 14:28 .xsession-errors.old
tc_michelle@myhostname:~$
```

Figure 4: An example of using `ls -la` to view detailed information about files in a directory.



That's a lot of info! What you see is all the files and folders in the folder you are at currently.

- The first column is the type of file followed by permissions. `-` means a regular file. `d` is a directory (folder). `l` is a link. `rw` are the permission for each file. `rw` stands for 7 so `rw-rw-rw` would be 777. These correspond to binary. There are 3 bits. 000 would stand for 0. 111 is 7. 101 is 5, etc. Each file has visible 3 permissions User, Group, Anyone.
- The next column is the number of links or directories in the folder
- The 3rd column is the user that owns the folder/file
- The 4th column is the group that owns the folder/file
- The 5th is the size of the file/folder
- The 6th is the month day and time it was last edited/touched
- And finally the file name

TASK: `cd` into your home directory and then type `ls` and you will see a directory called "follow\_Me". Travel as deep as that folder will go. When you get to the end there is a file whose name is a randomized number. Find that.

## sudo

Sometimes commands can only be run as a super user. This is when the command **sudo** comes to use (which stands for: superuser do). This gives unprivileged users access to privileged commands. If you try to do something and it says Permission denied, try again with **sudo** in front of the command.

## Man Pages

Man Pages is short for manual pages. These are text documents with lots of information on commands. Remember the command we did for listing our files? `ls`! Let's find that man page.

Type

```
~$ man ls
```

and hit enter

Remember we typed `ls -la`? Let's learn what `-l` and `-a` is!

`-l`

To search inside a man page, you use a `/`. Now type `/-l` and hit enter. You will probably see the page move to the first occurrence found and on top of that you should see that `-l` was highlighted\*. To move around your search keep hitting `n`

(stands for next) until you see `-l` highlighted to the left. This will be above `-L`. Long listing stands for listing the items in a row as seen in figure 2

**-a**

Type `/-a` and hit enter. Hit `n` till you can't go any further. Now hit `b` (stands for back) until you find the entry for `-a` which is above `-A`. The man page is telling you that files that start with `.` like `.bashrc` (which are typically hidden) are now going to be displayed.

**q**

When in a man page and you need to get out, just type `q`

TASK: Open the man page for “file”. Can you give a brief description of what the command “file” does?

- If you don't see it highlighted, you may have typed something by mistake or your console colors may not be optimized. If you typed something by mistake, just retype `/-l` etc.

## File Types

Not all files appear as they really are. Just because you see a file that says, `imanimage.png` does not mean that it is an image. It could be a text file or a harmful file if executed! So... how do you protect yourself? One way is with the `file` command!

**file**

To find out what a file really is regardless of its extension is `file`. Check out the man page. Give it a peruse by typing in `man file`. What type of options are there with `file`? Now let's test it. Type `q` to get out of the man page.

There are 2 files in your Linux box in a folder at your home directory called `/toLearn`. One is called `cat.jpg` and the other is `dog.jpg`

Both look like images to me! But if you type in `ls -l` you will notice that one is a lot larger in size than the other. One is about 25,000 bytes whereas the other is only about 20. Now let's see what is really going on.

Type

```
~$ file dog.jpg
```

You'll see something like, dog.jpg: ASCII text

Now type

```
~$ file cat.jpg
```

You'll see something like, cat.jpg: JPEG image data, Exif standard: TIFF image data, ... etc.

## **cat**

Now let's learn a new command, **cat**. **cat** prints out the text from a file.

Type

```
~$ cat dog.jpg
```

You should see something like:

```
meow I am a doggo
```

**TASK:** In your home directory there is a folder called **stuff**. Open that up and find out what file types are in there. One is a text file (ASCII). **cat** that and find the secret code inside.

## **Case Sensitivity / touch / echo / Angle Brackets**

### **Case Sensitivity**

Case sensitivity means that HoW yoU labEL yoUR files matters. If you search for a file called hiya.docx, it would not be the same as finding a file, hiyA.docX.

### **touch**

**touch** is a command that 'touches' a file. If the file exists it updates its modified date. If the file does not exist, then the file will be created with nothing in it. **man touch** to learn more.

### **echo**

**echo** will copy what you write to stdout (standard out, explained more later). You can use this in many different ways.

Type

```
~$ echo "This is echoed"
```

You will see that it was repeated back to you!

## Angle Brackets

Angle Brackets are > >> < << . They have many uses. > Will replace a file with what you input. If the file already existed > will delete **everything** in that file and replace it with what you sent it. In contrast if you use >> , this will append what you sent to the bottom of the file, leaving the rest of the file intact. Let's give it a try.

Type

```
~$ echo "This is cool" > newfile
~$ cat newfile
```

Now type

```
~$ echo "This is cool too" > newfile
~$ cat newfile
```

You can see that > will replace any text with what you send it. While >> will append to a file

Type

```
~$ echo "This is another thing" >> secondfile
~$ echo "Hello World" >> secondfile
~$ cat secondfile
```

Now let's combine the two files.

Type

```
~$ cat newfile >> secondfile
~$ cat secondfile
```

You can see the newfile appended to the end of secondfile whereas if you cat newfile it will still only have what we added to it earlier.

TASK: There is a folder in your home directory called textfiles. There are three files, append them all to a new file called, alltogether.txt in your personal directory. (Your home directory)

The tip below is NOT required but only if you want a harder task!

There are many ways to accomplish this in one line, here is a hint for one way, type

```
echo "one"; echo "two"; echo "three";
```

## vim, Regular expressions and find

### vim

**vim** is a program that is used to edit files, and will hopefully be your new best friend! There are different editors out there for example, **nano** and **emacs**. To create a file just type

```
~$ vim mynewfile.txt
```

Or

```
~$ vim thisisfun
```

To edit a file that is already created it's the same procedure, just make sure not to misspell it or you'll create a new file with that spelling.

Once you are in **vim** the main key strokes to editing a file are:

**i** - This puts you in edit mode to type and delete text like you normally would

**esc** - Hitting the escape key will take you out of edit mode

**:w** - These keystrokes will save the file. **w** stands for write.

**:q!** - To quit without saving. Did you edit a file and don't want to commit that change? These keystrokes will exit vim and NOT save your file.

**:q** - These keystrokes will quit the vim program. You can also do **:wq** to save the file and quit right away. **:q** will not work unless you have saved your file or you have made no changes what-so-ever.

**dd** - When you are NOT in edit mode this will delete an entire line. (Make sure your cursor is on the line you want to delete)

**0** - zero will take you to the beginning of a line

**\$** - will take you to the end of a line

There is a LOT that **vim** can do but we won't list it all here. Do a search on the internet to learn more! You can also check out a **vim** command cheat sheet, [here](#) and [here](#). But at the end of this lesson will be a couple more commands that you will find to be amazingly helpful! You can also check out **vimtutor**:

```
~$ vimtutor
```

**TASK:** In your home directory in a folder called **editme** there is a file called **editme.txt**, open that up in **vim**. Delete lines 4 and 5 and add 2 more lines of anything you would like at the end of the file. Don't forget to save.

## Wild Card and Regular expressions

Regular expressions are used to help you find something on your computer and can be used in programming to enhance your programs. There are a LOT of websites out there that teach you all about it but the gist is that you can use symbols like a `*` to mean something when parsing through text. For example the `*` is a wild card. Let's say I wanted to find a file with the word `spekter` in it. But there could be other text before and after the word `spekter`. So I could say search for, *spekter*. This means, search for `spekter` and I don't care if there is anything else before or after.

To learn about how powerful regular expressions are check out these sites:

<https://regexone.com/>

<https://regexr.com/>

## find

This leads us to `find`. `find` is a very helpful tool that you can use to find things on your computer. Take a moment to peruse the man page for `find`. (man `find`) Get an idea of how it is used.

An example as given earlier:

```
~$ find . -type f -iname *spekter*
```

- What you see is the command `find`. The next `'.'` is telling us where we want to find. It's the path. The dot means, search in this location where I am at. We could also type in `/Documents` or a full path from the root. Where ever you need to search.
- The parameter `-type` is telling `find` that you specifically want to find a type of object, in this case, `f` stands for a regular file.
- `-iname` is telling the name of the file we are looking for. You can also use `-name` but `-iname` is case insensitive. This means that my search will pull up, `SpeKter` as well.
- Lastly, the `*`'s on either side again tell `find` that I want everything with `spekter` in it, regardless of what is around it. If I did not add that, my `find` will pull up nothing.

TASK: Hidden throughout your home directory are image files with the name, `edurange`. Take your skills and find all 6. Create a new file in your home directory and put the location and type of each in that file. Remember there are many types of file images. `Png`, `jpg`, `jpeg`, and `gif` to name the widely used ones.

## More Commands

### **mv**

**mv** is used to move a file from one location to another, or to rename a file. Type **man mv** to learn more.

### **cp**

**cp** is used to copy a file to a new location. Type **man cp** to learn more.

### **less**

**less** is more. The command **less** is used to open larger files, page by page. It allows you more tools to read a file in an organized fashion.

### **cowsay**

**cowsay** is probably one of the best commands... ever. Well maybe not ever but it is fun!

```
~$ cowsay "this is fun"
```

### **fortune**

**fortune** gives you, well, a fortune! Go ahead give it a whirl!

```
~$ fortune
```

## Piping

Piping is a new concept but stick with me on this one. When you give input to a command it is considered a standard in (STDIN). In other words it is data that is fed to a program/command. Whereas when you see something printed out back to you it is using standard out (STDOUT). A pipe, also recognized as **|** is used when you want something that is a STDOUT to then be used as the STDIN. To understand this we will use a fun example.

Remember fortune? If we just type **fortune** we get a fortune back. That fortune we get back is a STDOUT. But when we use **cowsay** we type something for the cow to say, which is STDIN. So we can use a pipe to take our fortune and have the cow say it!

```
~$ fortune | cowsay
```

## More vim

There are a few more really useful commands I would like to teach you about vim.

If you want to open up multiple files at the same time you can do:

```
~$ vim fileone filetwo.c filethree.h filefour.cc
```

Or you can open up another file or create a new file while you are still in vim with:

```
:e anotherfile.js
```

**:b** - And you can jump from file to file with the **:b** keystrokes and using tab to go through which file you would like to edit next

## Final Mission

Here at CyberSec you have been well trained recruit, now let's put your training to test! This is a real mission of the utmost importance. Complete this and you are assured a spot on our team.

**TASK:** Our intelligence has told us that somewhere in your computer is an image that is vital to our company. This image has a weird name but we know for certain that it contains the word 'cowFJS' in it. Through our sources we also know that there is ANOTHER file in that folder. **COPY** both the image and second file to your home directory in a new folder, called final-mission. Then create a new file in the folder, final-mission, called 'cowsay.file' with the contents of the cowsay man file in it.

## 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 resetting 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 to 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

## 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

##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