Happy Cybertines Day!

This is the official walkthrough for the Cybertines Day exercises. It is meant to help familiarize you with the Linux CLI (command line interface). If you are completely new to all things shell, linux, and bash, use this guide as a step-by-step walkthrough. If you want a little more of a challenge, see how much progress you can make without referring to this guide.

Table of Contents

Some Helpful Tips

In case you don't know, keep these in mind:

- Use `man <command>` or google to learn more about a command
- Use the `tab` key for auto complete
- To copy and paste in the terminal use `ctl+shift+c` and `ctl+shift+v`

If you're ever stuck or confused don't be afraid to ask for help so someone can explain it to you. Feel free to reach out in discord or to one of the board members.

First Date Basics: I'd Like to Introduce you to my Friend CLI

Using the terminal can be intimidating at first, but you will quickly find that it can be much more efficient and give you easier access to various tools and resources. To start, we will go over some basic commands.

You will use:

- pwd
- ls
- cd
- man
- file
- cat

pwd: Print working directory. Basically get the absolute path to where you are currently at in the file system:

```
pwd
```

All levels of this activity will be in folders/directories. We will be starting out with the **firstDateBasics**. However, how do we know what is available in the folder we are currently in? Our first command is `ls` which lists directory content. The terminal will show a list of the files in your current directory. You'll see one of the directories is named **firstDateBasics**. We want to explore this directory.

```
ls
```

To get into this folder, we must change directory, which can be accomplished with `cd` followed by the folder we wish to go into **firstDateBasics**.

```
cd firstDateBasics
```

Tip: If you start typing out the file, you can use `tab` key to autocomplete the filename

Now you can use `ls` to see the files in **firstDateBasics**

From the `ls` command, you'll see theheart.. but we don't exactly know what **theheart** is from the `ls` command. We can use `file` command to determine the file type. Similar to usage of `cd`, we will type `file` followed by **theheart** the file we wish to gain more information on.

```
file theheart
```

This gives us a description of **theheart** to be a directory. Thus, we must move change directories into **theheart**.

```
cd theheart
```

Now inside **theheart**, you can use the `ls` command. You'll see a file, **walls.txt**. You may use `file` command on the file to make sure it is a text file as the file extension (.txt) indicates. Since **walls.txt** is a text file, we want to read the contents. The command `cat` allows for concatenating files and print out the standard ouput. In otherwords, we will see the contents of the file in the terminal.

```
cat walls.txt
```

Note: you may continue to use tab.

Now, the text file tells us a bit of information. Unfortunately, it looks like we have not solved this puzzle yet; the file tells us to look for real clues.

To complete this challenge, we will need to understand the commands further. The `man` command allows us to read the manuals of different commands. Let's `man ls`. Now we see, that there is an option `-a, --all`, do not ignore entries starting with . and we can conclude that the `ls` command is hiding something from us. These options are added after the command.

```
ls -a
```

Ah hah! Now we see more files! Let's `cat` the new file. That must be a clue to a later challenge! (Note the name of the file).

Tip: When the terminal gets too cluttered, you can type `clear` for a clean slate. `ctrl + L` also works.

It's Not Me, It's You: Looking for Red Flags in Logs

This challenge is taken from the National Cyber League (NCL) and is meant to introduce you to log analysis using the command line.

Log analysis can be tedious, but when you know how to use the right tools to extract information and identify patterns, the real fun begins.

The Challenge: Answer the following questions

1. How many requests were logged?
2. How many unique status codes were returned by the server?
3. How large was the largest response body in bytes?
4. How many HTTP tunneling attempts were made?
5. How many entries have completely invalid request lines containing raw binary data?
6. Of those invalid entries, how many likely the result of an attempt to establish an SSL or TLS connection?
7. How many unique user agents were observed, excluding empty or missing user agents?
8. How many requests were made by Firefox?
9. How many attempts were made to exploit CVE-2020-8515?

You will use:

- | (pipe)
- head
- cat
- wc
- grep
- cut
- sort
- uniq

Before we begin we are about to use a bunch of commands together so it is important to understand the idea of pipes `|`. A pipe is going to take the output of one command and use it as the input for the next. For example, `command A | command B` will use the output of command A as the input of command B.

Let's start with the first question where we need to determine how many requests were logged. I like to use the `head` command to print out the first few lines and get an idea of what the log format looks like:

```
head looking_for_red_flags.log
```

Don't forget to make use of that `tab` auto complete feature.

From the output we can see there is one request per line, so to get the total number of requests, we count how many lines are in the file. We do this using the `wc` (word count) command, but we want the number of lines, not words so we add the `-l` flag. We will first use `cat` to output the file and then use a pipe `|` to redirect it as the input for `wc` command:

```
cat looking_for_red_flags.log | wc -l
```

To determine the number of unique status codes, we can extract the codes from each line, sort them, and then get the number of remaining unique ones. Using the `cut` command, we will use the `-d` flag to indicate the delimiter and the `-f` flag to decide which field(s) we want to display after performing the split.

`cut -d '"' -f 3` gives us the status code. We then use `sort` to put them in order and `uniq` to get rid of duplicates. Lastly, we use `wc -l` to count how many unique codes there are.

Before doing this entire command try it out in increments to check the output of the command as you go and better understand what's going on:

```
cat looking_for_red_flags.log | cut -d '"' -f 3 | cut -d ' ' -f 2 | sort |
uniq | wc -l
```

Again before doing this whole command right away maybe try:

```
cat looking_for_red_flags.log
cat looking_for_red_flags.log | cut -d '"' -f 3
cat looking_for_red_flags.log | cut -d '"' -f 3 | cut -d ' ' -f 2
cat looking_for_red_flags.log | cut -d '"' -f 3 | cut -d ' ' -f 2 | sort
cat looking_for_red_flags.log | cut -d '"' -f 3 | cut -d ' ' -f 2 | sort |
uniq
cat looking_for_red_flags.log | cut -d '"' -f 3 | cut -d ' ' -f 2 | sort |
uniq | wc -l
```

To determine the largest response body in bytes, we will use a similar approach where we will `cut` the response size and then use `sort -n`. Note: The `-n` flag indicates numeric sort:

```
cat looking_for_red_flags.log | cut -d '"' -f 3 | cut -d ' ' -f 3 | sort -n
```

To determine the amount of HTTP tunneling attempts, we first need to know that this means we need to look for how many `CONNECT` requests were made. Like any of these questions, there are multiple ways to solve. We will solve this by using `grep` to search for the word `CONNECT` and then use `wc` to get how many lines with `CONNECT` were found:

```
cat looking_for_red_flags.log | grep 'CONNECT' | wc -l
```

To determine how many invalid requests there were with binary data we can look for any lines that have `\x` because we know that will be the format of the binary data. Note that when we use the `grep` command for this, we use the `'` single quotes and an additional `\` to properly escape the other `\`. Otherwise, the `\x` will be interpreted as a special character.

```
cat looking_for_red_flags.log | grep '\\x' | wc -l
```

We now need to find out how many of those were from an attempt to establish an SSL or TLS connection. To do this, we will print out the binary data we found along with the line before the found lines. We do this by adding `-B 1` to our `grep` command, where the `-B` flag indicates the number of lines *before* the matching line to include in the output:

```
cat looking_for_red_flags.log | grep '\\x' -B 1
```

We can see that the requests that started with `\x16` are preceded by a `CONNECT` indicating they were from an attempt to establish a connection. So we can specifically look for the commands with `\x16`:

```
cat looking_for_red_flags.log | grep '\\x16' | wc -l
```

Now to find how many unique user agents were found, we can use `cut` to extract the field with the user agent:

```
cat looking_for_red_flags.log | cut -d '"' -f 6 | sort | uniq
```

Note that we do not want to include any emtpy or missing user agents, meaning we want to exclude the user agent fields with `-`, we can do this using the `-v` flag which inverts the search, excluding the specified value. Once we do this we can find out how many lines there are to determine the answer:

```

```

```
cat looking_for_red_flags.log | cut -d '"' -f 6 | sort | uniq | grep -v "-"
| wc -l
```

To find the requests made by Firefox, we can use `grep` and include the `-i` flag to ignore case-sensitivity and then get the line count:

```
cat looking_for_red_flags.log | cut -d '"' -f 6 | sort | grep -i firefox |
wc -l
```

Finally, to determine how many attempts were made to exploit CVE-2020-8515 we first need to know that this involves using the `POST` method and involves the `cgi-bin/mainfunction.cgi` script. So to start maybe we first look at the `POST` requests. Note: The `-i` flag is used to ignore case-sensitivity:

```
cat looking_for_red_flags.log | grep -i "post"
```

We can see that there are some requests that include that `cgi-bin/mainfunction.cgi` script. So now we can do:

```
cat looking_for_red_flags.log | grep -i "cgi-bin/mainfunction.cgi" | wc -l
```

Setting Up Boundaries & Sticking to Them: Firewalls

This is an introductory exercise to get you used to the idea of firewall rules and how you can use the command line to configure them.

The challenge:

- Using ufw: enable the firewall, add a rule, delete a rule, then disable the firewall.

You will use:

- ufw

`ufw` (uncomplicated firewall) is a simple firewall configuration tool. It is a frontend for another tool, `iptables`.

Check the status: First we will check the ufw status:

```
sudo ufw status
```

Enable the firewall: If it is disabled, enable `ufw` and then check the status again:

```
sudo ufw enable
sudo ufw status
```

Note the default ufw setting is to block all external access and ports, so if you are using `ssh` and enable ufw without having a rule that allows `ssh`, you will get disconnected.

Add a rule: Here we allow traffic on port 22 (which is the port `ssh` uses). In this example, we are creating a rule that allows traffic for a certain port. Note: Rules can both `allow` and `deny` traffic, and they do not just have to be a port number. They can also be an IP address, subnet, application profile, a combination of those, and more.

```
sudo ufw allow 22
```

Note that we see two rules we added and one says (v6). This is reffering to IPv4 and IPv6.

Display the rules:

```
sudo ufw status numbered
```

Here we specify `numbered`. This will come in handy shortly for deleting a rule.

Delete a rule: There are different ways you can delete a rule. You can specifiy the rule itself or its number. From the last command, we can see our rules are rule 1 and 2. Either of these commands will work:

```
sudo ufw delete allow 22
```

or

```
sudo ufw delete 1
sudo ufw delete 1
```

You repeat this command twice since two rules are added when you do allow 22 (one for IPv4 and one for IPv6). When you delete rule 1, the numbers shift so what was previously rule 2 is now rule 1.

Diable the firewall: Disable the firewall and check the status:

```
sudo ufw disable
sudo ufw status
```

Hacking My Way Into Your Heart: Password Cracking

This section will show you how to crack encrypted passwords.

The challenge:

- Practice solving passwords.

You will use:

- hashcat

Let's first check out the **hackingMyWayIntoYourHeart** file.

```
cat hackingMyWayIntoYourHeart
```

Hashcat is a password recovery tool. In order to crack this password and hack our way into the heart, we must construct an *effective* `hashcat` command. Thought brute forcing is technically an option, it is hardly the most efficient and should only be used as a last resort.

To construct a hashcat command, we must know the hash algorithm type, the attack mode, and the wordlist:

- The `-m` flag denotes the hash algorithm type
- The `-a` flag denotes the attack mode
- The `wordlist` is a file containing a list of strings that `hashcat` will use to crack the password.

```
hashcat -m <hash type> -a <attack mode> <wordlist>
```

There are many different algorithms hashing algorithms. Recall the hint for this challenge (MD5SUM). Note: MD5 is an older hashing algorithm that is known to be vulnerable.

Let's check the hashcat manual to find the hash type for MD5. Either of these commands will work:

```
hashcat --help
man hashcat
```

The hash type section tells us that the mode for MD5 is `0`. Thus, we know the first part of our hashcat prompt is `-m 0`.

Next, since we intend to use a wordlist, we will be using a 'straight' attack mode, which the manual tells us is `-a 0`.

For this challenge, we will use the `rockyou` wordlist, which is a list of (millions of) passwords from an old data breach. It is commonly used to crack weak passwords, and it will be more than sufficient for this challenge.

If you are on a Kali machine, `rockyou.txt` should be pre-installed, but you must use `gunzip` to access it:

```
gunzip /usr/share/wordlists/rockyou.txt.gz
```

Lastly, we use the `-o` flag to save the results to a file.

Let's construct our hashcat command:

```
hashcat -m 0 -a 0 /usr/share/wordlists/rockyou.txt -o secretpassword.txt
```

Congratulations, you've cracked the password!

Secret Admirer: Steganography & Hidden Messages

This section will show you various commands and tools you can use to learn and extract useful information from files.

The challenge:

- Extract hidden information from the two secret_admirer files.

You will use:

- file
- strings
- binwalk

Let's start with **secret_admirer_pt1.bin**. A good starting point is to use the `file` command, which gives us information about the type of file. So we do the command:

```
file secret_admirer_pt1.bin
```

The output: From the output we can see that this is meant to be a binary file (which we expected since it's a .bin file, but you never know with those pesky files acting like other file types...). So we didn't get much information from that, but it wasn't a big waste and it's good to keep this command in mind.

Of course since it's a binary file our human brain is not going to be able to understand and read it, but maybe there is some useful information in there that we need to find, so another good, quick, and easy command to try is `strings`, this will output any strings found in the file. Don't forget that you can check out the man page using `man strings` for other flag options too:

```
strings secret_admirer_pt1.bin
```

Don't forget to make use of that `tab` button

Wow would you look at that, this binary was wishing us a happy Cybertines day all along.

Now moving onto **secret_admirer_pt2.jpg**. We see it's a jpg file but we can't open it. Perhaps we investigate further with the `file` command:

```
file secret_admirer_pt2.jpg
```

From the output we see that this is supposed to be a `.zip` file, so let's go ahead and rename it with the .zip extension. We can do this manually through right clicking and renaming in our file system (no shame in doing that, let's be real it's just as fast and easy as using cli). Though if you are a bash beast then feel free to use this command:

```
mv secret_admirer_pt.jpg secret_admirer_pt2.zip
```

Now we have our zip file ready but when we try to open it, it still doesn't work. This is because I deleted the last 100 bytes of the file (using the `truncate` command) which is where zip files store their central directory (where zip files store information about the file structure and the metadata for the files in the archive).

So the zip file doesn't know the information it has in there but it is still in there and we can still extract it. We are going to do this using `binwalk` which is a tool that identifies files and code embedded in binary images. You will find that even though this isn't a .bin file it will still work with the .zip file because it has a binary header and file signature information that binwalk looks for.

Just using binwalk without any flags will give a summary of the embedded data it found and then we can use the `-e` flag to extract the files.

```
binwalk secret_admirer_pt2.zip
binwalk -e secret_admirer_pt2.zip
```

We now have a folder with all the extracted information.

Love is in the Shell: Some Fun Helpful Random Tools

There are countless command line tools for countless uses, here are a few more fun or useful tools to check out:

- tmux
- vim
- cbonsai (A bonsai tree in your terminal)
- cowsay (Make a cow say something)
- figlet (Large text using ASCII characters)
- lolcat (Rainbow text)

Farewell

We hope you had a wonderful and secure Cybertines day, remember Cybertines isn't just a day, it's a spirit and a season. Only you can make everyday feel like Cybertines.