

# CITS3006 Group Report - Part 1

Group: Web Warriors

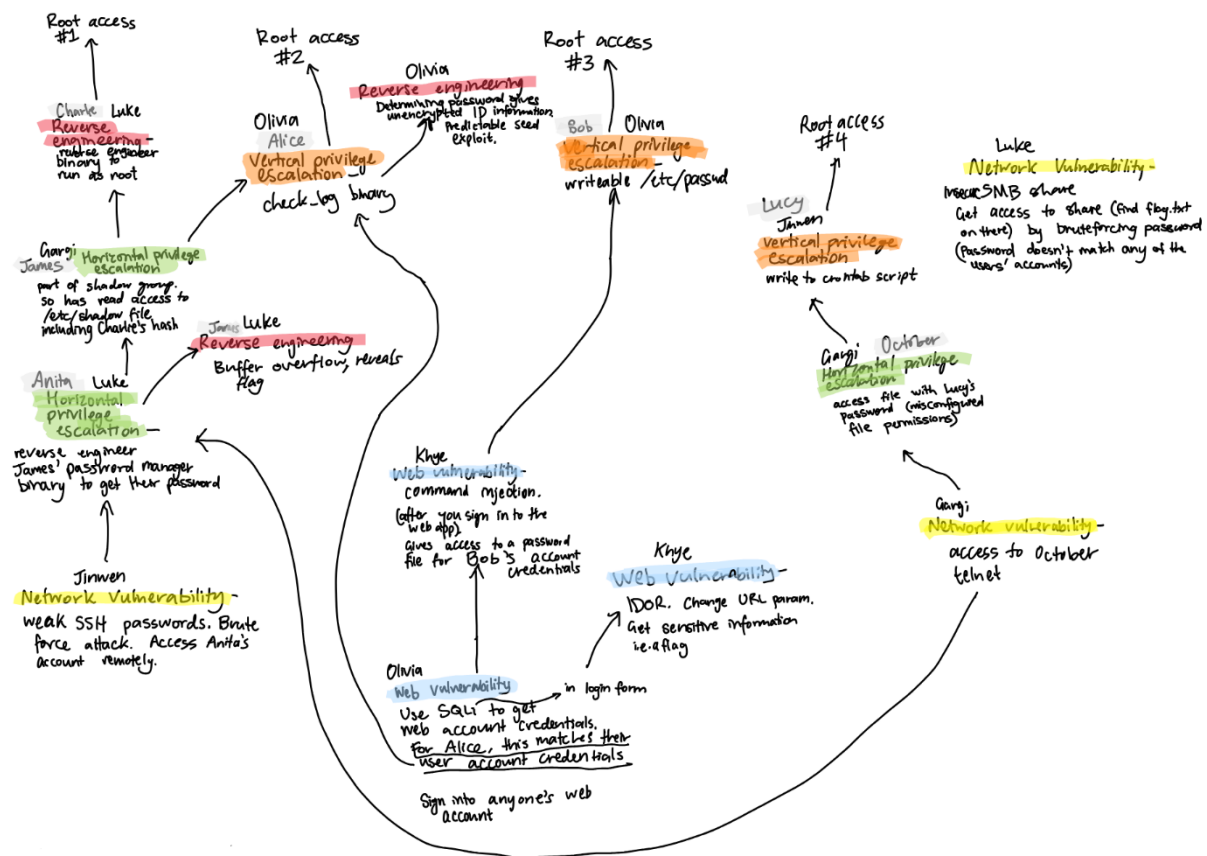
Members:

Olivia Hanly (23331483), Luke Waters (23487767), Khye Goh (22886485), Jinwen Liu (23736736), Gargi Garg (23887876)

## Introduction

Attackers will begin with access to the local network and the local machine. They won't be able to sign into any accounts, so the initial attack surface is only through the network. By chaining various exploits added to the machine, attackers will eventually be able to access all accounts and gain root access.

The following diagram summarises the machine's vulnerabilities and intended attack chain:



## Exploits

### **Reverse Engineering #1 – Buffer Overflow (Luke Waters)**

Buffer overflows are a common vulnerability found in many applications, and usually involves user input being written to a fixed-length buffer. The buffer bounds aren't checked, and an attacker can input something longer than the buffer can hold. This allows the attacker to control the memory after the buffer, which can be used to compromise the application or system.

For the exploit, the application takes the first command-line argument and copies it to a buffer that is only 20 bytes long. This is stored on the stack, and an integer variable is next on the stack. This variable is set to -1, and to reveal the flag it is compared to 0x4B434148.

```
void __fastcall bufferOverflow(char *input)
{
    char shortBuffer[20]; // [rsp+20h] [rbp+0]
    int getsOverwritten; // [rsp+38h] [rbp+18]
    getsOverwritten = -1;
    j_strcpy_0(shortBuffer, input);
    if ( getsOverwritten == 0x4B434148 )
        printFlag(0x4B434148);
}
```

Without a buffer overflow, the flag is never revealed as the comparison will fail. To exploit the buffer overflow, the attacker needs to give a command-line argument long enough that it will overrun the buffer and also set the integer variable to 0x4B434148. An example of what input can be used is "1111111111111111111111111111HACK". This will overrun the buffer, the 8-byte stack padding and the integer variable with "HACK". While this converts to 0x4841434B in hexadecimal, integers are stored as little-endian so it will be read as 0x4B434148 during runtime. If this is successfully exploited, it will reveal the flag "flag={deadbeef}".

This flag is also encrypted using XOR encryption, and as an extra security layer the key is 0x4B434148 so it would be much easier to exploit the buffer overflow instead of trying to decrypt the flag.

### **Reverse Engineering #2 – Return-Oriented-Programming Buffer Overflow (Luke Waters)**

This exploit is similar to the Reverse Engineering Exploit #1, but more complex. When executed, the program asks for a hexadecimal string input. This string is converted into its byte array equivalent and copied to a buffer that's 20 bytes long. The code doesn't check if the input is larger than the buffer, so a large enough input will overrun the buffer. The buffer is stored on the stack, and an integer variable is next on the stack. This variable is set to -1, and to continue execution it must be equal to 0x4C495645 (EVIL backwards

as integers are stored as little-endian). A successful buffer overflow will cause the program to print “try again :(” as there is more to the exploit.

```
void __fastcall bufferOverflow(char *input, size_t size)
{
    char shortBuffer[20]; // [rsp+28h] [rbp+8h] BYREF
    int getsOverwritten; // [rsp+54h] [rbp+34h]

    getsOverwritten = -1;
    j_memcpy_0(shortBuffer, input, size);
    if ( getsOverwritten == 0x4C495645 )
        printIncorrectFlag(0x4C495645);
}
```

[illegible]

On the stack, beyond the buffer and integer variables, there is the stack pointer and the return pointer. The return pointer is used to return after a function has been called, and the stack pointer controls the stack when a function is called. These variables are what need to be exploited next to reveal the root password. If an attacker analyses the program further, there's an unused function called "printCorrectFlag" that will print the root password.

To successfully exploit the program and show the root password, the return pointer must be set to a pointer to “printCorrectFlag”. To do this, a debugger needs to be attached to the program first as the address of the program is randomised each time it is executed due to ASLR (Address Space Layout Randomisation). Using the debugger, we can get the address of “printCorrectFlag” and a valid stack pointer and start crafting the buffer overflow:

Buffer overflow and integer overwrite:

[illegible]

Stack and return pointer overwrite:

- Both pointers are 8 bytes long and need to be reversed due to the x64 architecture being little endian.
- For example, in one run the stack pointer was 00007FFE6C341A70 and the function pointer was 000058355D1D81E9. The stack pointer is before the return pointer and both need to be reversed which gives:
- 701A346CFE7F0000E9811D5D35580000

[illegible]

In Alice's home directory, in a folder called Personal, there is an executable called id. When run, the executable will prompt the user for a password to access Alice's ID information. If the correct password is given, the ID information is printed in plaintext, but if the wrong password is given, the printed information is incomprehensible.

1. The password inputted is read using scanf into variable called local\_64. Navigating to the scanf format specifier address DAT\_00102033 shows that password is read in as an octal value (%o).

DAT_00102033				
00102033	25	??	25h	%
00102034	6f	??	6Fh	0
00102035	00	??	00h	

- ```
int gen_num(int param_1)
{
    return (param_1 * 3) % 0x28;
}
```

After narrowing down the possible passwords, the correct password can be bruteforced (40 possibilities is low enough that it can be done manually, or a script could be written to automate the process). The correct number is decimal 33 (actually, many values are possible e.g. 73, 113 etc.) so 41 (octal value of decimal 33) is the correct password that will reveal the sensitive information (passport number and social security number) in plaintext.

## **Horizontal Privilege Escalation #1 – Misconfigured File Permissions (Gargi Garg)**

Horizontal privilege escalation occurs when a user gains access to another user's account and resources at the same privilege level.

Lucy has a hidden folder in her home directory, intended for storing sensitive files. On Linux-based systems, hidden directories (denoted by a preceding period, e.g., /home/lucy/.hidden\_files) are used to store files that should not be easily visible. However, even though directories are hidden, they are not automatically secured unless appropriate permissions are set. Unfortunately, due to incorrect usage of the chmod command, the permissions on this folder allow October to access it. This opens the door for exploitation.

Using a command like the following, October can quickly find readable files across the system, including Lucy's hidden files:

```
find / -readable ! -user october ! -group october -type f -exec ls -al {} \; 2>/dev/null | grep -v '/proc/*\|/sys/*'
```

This command lists all files that October does not own but can read, including hidden directories. This method reveals Lucy's home directory files. Normally, these files should be protected by restrictive permissions, such that only the owner (Lucy) or the root user can access them. However, incorrect permissions on the folder and files allow other users like October to read the contents, including Lucy's password.

Unauthorized Access: Once October retrieves Lucy's password from the password.txt file, October can log in to Lucy's account using her credentials. This gives October the same level of access as Lucy, enabling her to:

- Access Lucy's Private Data: October can now read, modify, or delete Lucy's files, emails, or any other data Lucy has stored.
- Perform Actions as Lucy: October can execute commands under Lucy's user privileges, potentially accessing sensitive work documents, private correspondence, or even other user accounts if Lucy has elevated access to shared resources (e.g., via SSH or shared drives).
- Escalate Further Attacks: By gaining access to Lucy's account, access other systems Lucy has permissions for, or even spread the attack to other machines or services. October could also launch further attacks on the system, including lateral movement, to compromise other parts of the network.

## **Horizontal Privilege Escalation #2 – Readable /etc/shadow file (Gargi Garg)**

Horizontal privilege escalation occurs when a user exploits a misconfiguration to gain access to sensitive information or other user accounts without elevating their own privileges. In this scenario, James, a user with the same privilege level as others, is

mistakenly added to the shadow group, giving him read access to the system's password hashes stored in `/etc/shadow`.

The `/etc/shadow` file contains hashed passwords for all users in a Linux system. Normally, this file is only accessible by the root user and certain processes that need to verify user logins. However, due to a misconfiguration, the user James has been added to the shadow group, granting him read access to this critical file.

By using the `id` command, James can verify that he is indeed a member of the shadow group. The command output confirms that James has the necessary permissions to access `/etc/shadow`. By verifying the file permissions using the command: `ls -l /etc/shadow`, the system returns:

```
-rw-r----- 1 root shadow 3424 Sep 23 18:10 /etc/shadow
```

This output shows that members of the shadow group have read access to the shadow file. James can then display the contents of the file using the following command: `cat /etc/shadow`

This allows James to view the password hashes of all users on the system. James can download the contents of the shadow file and use widely available password-cracking tools like John the Ripper or Hashcat to run offline attacks against these hashes.

If a user, such as Charlie, has a weak or easily guessable password, these tools will allow James to crack the hash quickly. For instance, by comparing Charlie's password hash with common passwords from a wordlist like `rockyou.txt`, James can deduce Charlie's actual password and gain access to his account.

### **Horizontal Privilege Escalation #3 – Reverse Engineer Password Manager (Luke Waters)**

This exploit is contained within a password manager, where the correct password needs to be inputted to reveal the usernames and passwords contained in the password manager.

The correct password is encrypted using Vigenère encryption, which is decrypted at runtime and compared to the password input given by the attacker. The cipher-text is "NRMZEEBXAXSUSJHYDYYP", the key is "CANTHACKTHIS", and any attacker who knows basic encryption methods would recognise the key and ciphertext are Vigenère encryption due to the key and ciphertext both using alphabet-only characters. They are both hardcoded in the executable, so attackers could reverse-engineer it to find the key and ciphertext and use an online decryption tool to find the correct password.

While the password is encrypted using Vigenère encryption, the usernames and passwords in the password manager are encrypted using XOR encryption. This could also be reverse-engineered but would be much harder than the Vigenère encryption.

Once the correct password “LEETSECRETPASSWORDWOW” is inputted, the password “hardpassword1” for the local account “James” is displayed. This is how James’ password is found, and it leads to another exploit.

## Vertical Privilege Escalation #1 – Find Command Injection with Elevated Permissions (Olivia Hanly)

In Alice’s home directory there is a text file with the following message.

```
Hi Alice,

I know you often need to check if each employee has updated their
worked_hours_log.ods in the past week so that it's ready to send to
you at the end of the week. This program will check that for you so
you can chase them up about it if they haven't.

Simply open Terminal and run:

check_log

Our ICT security specialist was away this week so we just had our
intern whip this program up - hopefully there are no issues.

Cheers,
Anita
```

When the check\_log command is run, Alice is prompted for the name of a user. If another user’s name is given e.g. lucy, the command works as expected and either the modified date of the file is reported, or a file not found message is printed.

If we type in any text that is not recognised as a user, e.g. qweh, the following error message appears: “find: ‘/home/qweh’: No such file or directory”.

This reveals that the binary works by using the find command, and takes the input of the name of the user Alice wants to search, to construct the starting directory as the user’s home directory by appending the input to ‘/home/'. This also indicates that the executable runs with elevated privileges in order to search other users’ home directories (This can be checked by running find /home/anita from the command line which gives a permission denied message). This means that a carefully crafted input may be able to perform command injection to execute commands with root privileges. Using the exec flag with find is a powerful way for arbitrary commands to be run as root, such as the following:

```
alice -name worked_hours_log.ods -exec bash -c 'sudo adduser alice sudo' \;
```

This works because the find command searches Alice’s home directory for a file called worked\_hours\_log.ods, which it finds (this can of course be changed to anything, you would just want the find command to match with at least one file so the next command will be executed). In a separate bash shell, the command ‘sudo adduser alice sudo’ is then performed (with root permissions), allowing the attacker to gain root access (assuming the user has Alice’s password at this point, which they are intended to have according to our intended attack chain).

A note on the implementation of this vulnerability: this vulnerability has two compiled bash executables, one called `run_check_log` (which contains the actual functionality of the command), which Alice has the ability to run using `sudo` without a password. This executable is stored in root's home. Then another compiled executable that is called `check_log` and is stored in `/usr/local/bin`, and simply calls `sudo run_check_log` (but with the absolute path). Therefore, while anyone can run `check_log`, only Alice can run it correctly, as all other users will be prompted for a password from the `sudo` command.

## **Vertical Privilege Escalation #2 – Writeable /etc/passwd file (Olivia Hanly)**

File permissions can often be misconfigured, allowing users to access and modify files they shouldn't have access to. Using the `find` command can be useful to quickly find files that might have sensitive data or misconfigured file permissions.

In Bob's account, using the `id` command shows that he is part of the `pswd` group. The following command:

```
find / -group pswd -type f -exec ls -la {} \; 2>/dev/null | grep -v '/proc/*\|/sys/*'
```

looks for files that are owned by `pswd`, and lists out their permissions (ignoring uninteresting files found in `proc` and `sys`). This prints the following file:

```
-rw-rw-r-- 1 root pswd 3469 Sep 26 00:18 /etc/passwd
```

indicating that bob has write access to the `/etc/passwd` file.

The command:

```
find / -writable ! -user bob ! -group bob -type f -exec ls -al {} \; 2>/dev/null | grep -v '/proc/*\|/sys/*'
```

can also be used to find writeable files that are not owned by bob (as user or group), which reveals the same files.

While the `/etc/passwd` file no longer contains passwords and uses an `x` to indicate the hash is stored in the `/etc/shadow` file, the `/etc/passwd` still takes precedence. Therefore, an attacker can create a hash of a known password and edit the file to replace the `x` in root's line with the new hash, then switch user to root to gain root access. Another possibility is to add a new user with a known hash and with user id 0, to gain root privileges.

## **Vertical Privilege Escalation #3 – Writeable Crontab Script (Jinwen Liu)**

Crontab is a scheduled task scheduling tool in Linux system, which can execute tasks at set time intervals. We can adjust its execution frequency for system monitoring, data



backup and other operations. If crontab executes a script file and an attacker can edit the file, they can gain root permissions.

For this exploit, crontab will execute the script file “/cron/crontab\_log.sh” every minute.

```
lucy@bankwarriors-VirtualBox:~/Desktop$ ls -al /cron
total 12
drwxrwxrwt  2 root root 4096 Sep 26 01:30 .
drwxr-xr-x 24 root root 4096 Sep 26 01:05 ..
-rwxrw----  1 root log   69 Sep 26 01:30 crontab_log.sh
```

This file can be read/written to by lucy (who is a member of the group ‘log’), and they can exploit this to run privileged commands. Appending the lines “cp /bin/bash /tmp/rootbash” and “chmod +xs /tmp/rootbash” to the file will cause a bash shell with SUID permissions to be copied to the temp directory.

```
lucy@bankwarriors-VirtualBox:/tmp$ ls -al ./rootbash
-rwsr-sr-x 1 root root 1446024 Sep 27 14:00 ./rootbash
```

When it is added and crontab executes the script, they can Run “/tmp/rootbash -p” to get access to a root shell.

```
lucy@bankwarriors-VirtualBox:/tmp$ ./rootbash -p
rootbash-5.2# whoami
root
rootbash-5.2#
```

## Web #1 – Union-Based SQL Injection (Olivia Hanly)

The bankwarriors webapp directs clients to a login page, which has username and password inputs.

Testing with a quotation mark ‘ crashes the website indicating that it could be SQL-based and has poor sanitisation and error handling, so it is vulnerable to SQLi. Inputting: “a’ OR ‘1’ = ‘1” in username and anything in password reveals an error message saying that bobbyboy’s password is not correct. Clearly, if a user inputs a correct username but incorrect password, an error message saying that their password is incorrect is displayed. Because the website shows results from queries, Union-based SQLi can be used to leak data. Inputting: “a’ UNION SELECT 1,2,3—” will reveal that there are three columns, and a 2 will be where the username in the error message usually is, indicating that the second column on any queries will be displayed back to the attacker.

First, the name of tables and columns needs to be determined. This webapp uses SQLite, so pragma tables are used to store information about the database, and can be accessed using pragma commands, or SQL queries to the pragma tables. Attackers can discover that SQLite is used as testing other queries and commands e.g. MySQL or PostgreSQL will not work. A full list of pragma database commands and their associated tables can be found at <https://www.sqlite.org/pragma.html> which also outlines the column names.

To get the filename (filepath), indicating the name of the database is webapp.db:

- a' UNION SELECT seq, file, name from pragma\_database\_list---

To get the names of tables (and using the SQLite command GROUP\_CONCAT):

- a' UNION SELECT 1, GROUP\_CONCAT(name), 3 from pragma\_table\_list---

- Result:

sqlite\_sequence,webusers,sqlite\_schema,sqlite\_temp\_schema's password is not correct

We can guess that the webusers table will have the user information we want. We can use pragma\_table\_info to get the column names:

- A' UNION SELECT 1, GROUP\_CONCAT(name),3 FROM main.pragma\_table\_info('webusers')---

- Result:

Username  Password

id,username,password\_hash,first\_name,last\_name's password is not correct

To get the usernames and password hashes of the users, we use the commands:

- a' UNION SELECT 1,GROUP\_CONCAT(username),3 from webusers---
- a' UNION SELECT 1,GROUP\_CONCAT(password\_hash),3 from webusers---

Now that we know the username and password hash of the users, we can use an online password cracker to crack the hashes and sign in as the users. The user Alice, who has a web account and is also a user on the machine, uses the same password for both, giving attackers access to the machine through Alice's account.

## Web #2 – Command Injection (Khye Goh)

Command injection is a type of attack where an attacker can execute arbitrary commands on the host operating system through a vulnerable web application. It takes advantage of insufficient input sanitization, allowing malicious inputs to be passed through user input fields such as search bars or forms. For this project, I developed a bank-themed webpage to demonstrate this vulnerability. A search bar was implemented in the navigation bar, which could be exploited by attackers to execute unauthorized commands.



When a user enters a query into the search bar, a GET request is sent to the /search route, which is specifically designed to handle GET requests. The query is then processed in the

terminal shell using the subprocess module. Then, the result will be rendered in another page called search\_results.html.

```
@app.route('/search', methods=['GET'])
def search():
    query = request.args.get('query')

    # Vulnerable command execution
    if query:
        try:
            # Here, we use a shell command directly with the user input, which is unsafe
            command = f"echo {query}" # Replace with a vulnerable command as needed
            output = subprocess.check_output(command, shell=True, text=True)
            return render_template('search_results.html', output=output)
        except subprocess.CalledProcessError as e:
            return f"Error: {str(e)}"

    return "No query provided."
```

For example, if an attacker enters “;ls” into the search bar, it will execute the ls command and list the files in the current directory.

## Search Results

app.py  
confidential  
init\_db.py  
\_\_pycache\_\_  
schema.sql  
static  
templates  
venv  
webapp.db

Similarly, the attacker can use the “;cat” command to display the contents of files.

Attackers can then read “./confidential/sensitive.txt” to get the username and password of a local account (Bob).

## Web #3 – IDOR (Khye Goh)

IDOR (Insecure Direct Object Reference) is a web application security vulnerability that allows users to access or modify files or data they should not have access to by manipulating the parameters passed to the server. In this web application that I created, several bank accounts were made to demonstrate this vulnerability. Specifically, I used an identifier directly related to a database object—the user ID. When a logged-in user navigates to their account page, the URL displays the user ID at the end, such as `http://127.0.0.1:5000/show_account?id=6`.

127.0.0.1:5000/show\_account?id=6

Home

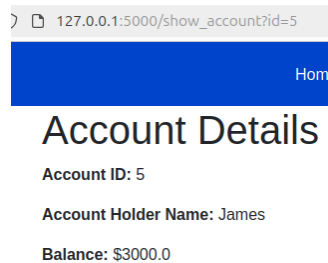
Account Details

Account ID: 6

Account Holder Name: Bob

Balance: \$1800.25

An attacker can exploit this URL by simply changing the last number, which represents the user ID, to another value, allowing them to view other users' account details. For example, Bob can access James' account information, even though he should not have permission, by changing the parameter to 5 in the URL.



## **Network #1 – Insecure SMB credentials (Luke Waters)**

Samba is software that implements many windows protocols to allow interoperability between linux and windows operating systems. This includes the SMB protocol, which allows a folder to be shared to the network and accessed by other computers on the network. These folders all have access control enabled, and a username and password is required to access their contents.

To successfully exploit the SMB share, attackers must determine the correct username and password for the share. There are no hints to what username is meant to be used, as samba requires that the username for the network share is the same as a local account. This narrows down the possible list of usernames, and I decided to use the local account username “svc”. The password can be anything, but I decided that for the SMB share to be exploitable the password must be insecure. To do this, I chose the password “graduatedsenior” as the password is from rockyou.txt and could be easily brute forced. This could be done using Metasploit or any other SMB brute force methods.

Once the correct username and password are found, the SMB share contains a file called “flag.txt”, and contained within it is a flag to signal that they successfully exploited the SMB share.

## **Network #2 – Insecure SSH (Jinwen Liu)**

Secure Shell is a protocol that provides secure encryption for network services, usually used for remote login and secure data transmission. It can support multiple authentication methods, such as passwords and public keys. Therefore, if there are vulnerabilities in the SSH service, it will be attacked by attackers. For example, weak password configuration, allowing password authentication, incorrect permissions, etc.

To find SSH vulnerabilities, the attacker must first scan the target port through the Nmap scanning system. Usually, SSH listens on port 22 by default, and then the attacker can use

relevant commands to obtain information. You can use `nmap -p` to view the status of various ports.

```
Starting Nmap 7.95 ( https://nmap.org ) at 2024-09-23 13:11 AWST
Nmap scan report for ubuntu-22.04.2.shared (10.211.55.5)
Host is up (0.91s latency).

PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    closed http
[2222/tcp closed EtherNetIP-1
```

Use `nmap --script ssh-auth-methods -p` to view the configuration of the SSH service.

```
Starting Nmap 7.95 ( https://nmap.org ) at 2024-09-23 13:16 AWST
Nmap scan report for ubuntu-22.04.2.shared (10.211.55.5)
Host is up (0.88s latency).

PORT      STATE SERVICE
22/tcp    open  ssh
| ssh-auth-methods:
|   Supported authentication methods:
|   publickey
|_  password

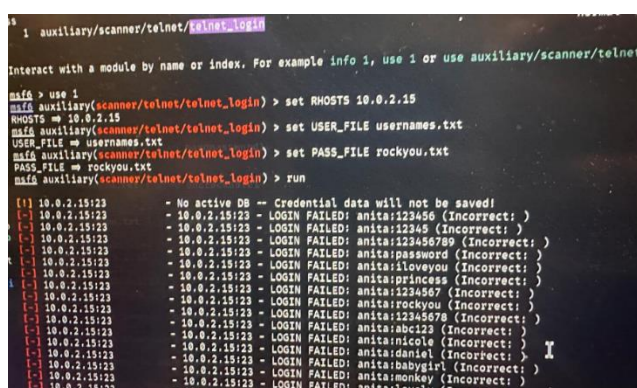
Nmap done: 1 IP address (1 host up) scanned in 1.20 seconds
```

This determines how we can crack the user. If password authentication is enabled, we can try to use Hydra for brute force. If password authentication is not enabled, we can try to use arpspoof to deceive the machine and capture the traffic, open Wireshark to capture the data. This method is used to crack our first user 'Anita'.

### Network #3 – Insecure telnet (Gargi Geng)

Telnet is an old network protocol used for remote communication, allowing text-based command input from a remote device. However, Telnet has significant security flaws, including the lack of encryption, making it vulnerable to interception and brute-force attacks. This scenario demonstrates how an attacker can exploit a misconfigured Telnet service to gain unauthorized access to a system, as the Telnet service was left open and vulnerable. Telnet need to be set up properly with the implementation and processing before attacking from another vm.

Brute-Force Attack: Using a tool like Metasploit, an attacker initiates a brute-force attack on the Telnet login using a wordlist (e.g., rockyou.txt).



```
1 auxiliary/scanner/telnet/telnet_login

Interact with a module by name or index. For example info 1, use 1 or use auxiliary/scanner/telnet

msf5 > use 1
msf5 auxiliary(scanner/telnet/telnet_login) > set RHOSTS 10.0.2.15
RHOSTS => 10.0.2.15
msf5 auxiliary(scanner/telnet/telnet_login) > set USER_FILE usernames.txt
USER_FILE => usernames.txt
msf5 auxiliary(scanner/telnet/telnet_login) > set PASS_FILE rockyou.txt
PASS_FILE => rockyou.txt
msf5 auxiliary(scanner/telnet/telnet_login) > run

[*] 10.0.2.15:23 - No active DB -- Credential data will not be saved!
[*] 10.0.2.15:23 - LOGIN FAILED: anita:123456 (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:12345 (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:123456789 (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:password (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:iloveyou (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:princess (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:1234567 (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:rockyou (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:12345678 (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:abc123 (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:nicole (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:daniel (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:babygirl (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:monkey (Incorrect: )
[*] 10.0.2.15:23 - LOGIN FAILED: anita:lovely (Incorrect: )
```

Once a valid username-password combination is found, the Telnet session is successfully established, allowing access to the target machine for users with insecure passwords, including October's.

The attack process is as follows:

- Identify the target IP address: The attack is aimed at the Telnet service running on the identified target IP, using tools such as nmap.
- Launch the Brute-Force Attack: The attacker configures Metasploit to try different username and password combinations from the wordlist. Given the insecure nature of Telnet, the target is vulnerable to a high volume of login attempts.
- Cracking Weak Passwords: As the target server uses weak passwords, the brute-force attack grants the attacker access to the system.
- Successful Exploit: The attacker can log into the target system remotely via Telnet. The Telnet session provides full access to the system under the compromised user account.

Impact: Unauthorized Access, Privilege Escalation

Prevention: Disable Telnet, Enforce Strong Password Policies, Use Encryption, Monitor and Limit Login Attempts, Regular Audits and Vulnerability Scans.

## **Appendix**

### **User Credentials**

bankwarriors: h@Kl10)ap%jWp

alice: sneaky88

james: hardpassword1

svc: \$j9am2UC8\*01Amch

anita: (hello126)

charlie: password1234

bob: mR232aA&s1ssid

october: secret123

lucy: +{rQlp3\$1mghE