Format String Vulnerability

Wei Wang
Format String Vulnerability Basics

- The semantic of printf:
  - `printf( char *fmt, ...)

- The parameters on the stack for printf:
  - last value to print
  - ... (omitted)
  - second value to print
  - first value to print
  - address of format string
Format String Vulnerability Basics cont'd

• Normally we would do:
  – `printf("%d", 1234);`
  – The number of variables to print should match the number of fields in the format string

• What happens if these two numbers do not match?
Format String Vulnerability Definition

- Consider the following code

```c
int main(int argc, char argv[]){
  char buff[256];

  strcpy(buff, argv[1], 200);
  printf(buff);

  return 0;
}
```

- What happens if the argument, argv[1], is just “%d”?

- Format string vulnerability, a.k.a. uncontrolled format string, happens when unchecked user-input is directly used as format string passed to printf, allowing the reading and writing of arbitrary memory.
Exploit Format String Vulnerability

Case 1: Program Crash

- Format “%s”: prints the string pointed at by an pointer
- The attack:

```c
buff="%s%s";
printf(buff);
```
Exploit Format String Vulnerability
Case 1: Program Crash con'td

- The attack:
  ```
  buff="%s%s";
  printf(buff);
  ```

- The stack when printf is called:

- The temporary variables generated by gcc are less likely can be used as valid memory address (string addresses)

- The program will crash if it tries to address some invalid addresses
Exploit Format String Vulnerability
Case 2: View Stack

- Format “%x”: prints the hexadecimal values of
  the memory after format string

- The attack:

```c
buff="%x%x";
printf(buff);
```
Exploit Format String Vulnerability
Case 2: View Stack con'td

- The attack:
  ```c
  buff="%x%x";
  printf(buff);
  ```

- The stack when printf is called:

- Two gcc temporary values will be printed
Exploit Format String Vulnerability

Case 2: View Stack cont'd

- If we want to view the 8th element on the stack, we need to use format string
  - "%x%x%x%x%x%x%x%x"
  - It is heavy handed.
  - What happens if we want to view the 1024th value?

- We can use the field specifier, e.g.,

  ```c
  printf("%2$x, %1$x", 1, 2);
  ```
Exploit Format String Vulnerability
Case 2: View Stack cont'd

• We can use the field specifier, e.g.,

```c
printf("%2$x, %1$x", 1, 2);
```

prints 2 first, then 1

• In general “%m$x” tells printf to print the m'th value
  – e.g., for the 1024'th value, we can use “%1024$x”
Exploit Format String Vulnerability
Case 2: View Stack cont'd

- This exploit allows the attacker to view the stack.
- This exploit can be used by attacker to determine the important addresses of stack objects, such as return addresses or saved EBP.
Exploit Format String Vulnerability
Case 3: View Arbitrary Memory

• We will keep using “%s” in this exploit
• The trick is to put an address on to the stack
• For example, if there is a string at address 0x80485e0, the format string can be:

```c
buff="\xe0\x85\x04\x08%x%x%x%x%s";
printf(buff);
```
How to Input Non-ASCII Characters?

- How to input a ASCII character “0xe0”?
- Two methods to generate the attack string
  - Write a C program
    
    E.g., char a[] = “\xe0\x85”;
  - Use perl
    
    E.g., perl -e 'print "\xe0\x85";'
- How to pass the attack string to the program?
  - Save the attack string into a file, and use bash stdin redirect (Google bash redirect if you don't know)
  - Command substitution – $(command) or `command` (note it is not single quote)
Exploit Format String Vuln. Case 3: View Arbitrary Memory con'td

The stack when printf is called:

Because the format string starts with hex number 0x080485e0, this number is first printed

Then three %x prints prints the temp vals

At last %s prints the string at address 0x080485f0
Exploit Format String Vulnerability
Case 4: Write Arbitrary Memory

• Format “%n”: writes the number of characters printed to a variable

• E.g., the following code writes 4 to i

```c
int i;
printf("ABCD%n", &i);
```
Exploit Format String Vuln. Case 4: Write Arbitrary Memory cont'd

- Similar to the view arbitrary memory exploit, and address has to be push onto stack for writing
- For example, the follow code write to address 0x804a030

```c
buff="\x30\xa0\x04\x08%x%x%x%n";
printf(buff);
```
Exploit Format String Vuln. Case 4: Write Arbitrary Memory cont'd

- The stack when printf is called:
- Because the format string starts with hex number 0x0804a030, this number is first printed
- Then three %x prints prints the temp vals
- At last %n prints the string at address 0x0804a028
Exploit Format String Vuln. Case 4: Write Arbitrary Memory cont'd

• Because the value written is the number of characters printed, it is slightly difficult to give a specific value.

• Attackers can control the number of characters printed with precision specifier or field with specifier.
Summary and Defense

- Format String Vulnerability is a bug that allows user to control the format string passed into printf() family functions.

  ```
  printf(user_input);
  ```

- The correct way to write is (essentially do not let user input be your format string)

  ```
  printf("%s", user_input);
  ```

- Compiler (e.g., gcc) provides check for unsafe use of format strings.
Summary of Format String Vulnerability

- Attackers can exploit this vulnerability to crash a program, view its stack, or view/write arbitrary memory locations.

- Format string vulnerability allows attackers to determine the values of important stack objects, which can be used for buffer overflow attacks.

- Format string vulnerability sometimes is more dangerous than stack buffer overflow because it allows attackers to write to arbitrary memory locations.