Format String Vulnerability

Lecture 17

Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

Acknowledgment: Adapted partially from course materials from Dr. Wenliang Du at Syracuse University, Dr. Fengwei Zhang at Southern University of Science and Technology, and Dr. Steven M. Bellovin at Columbia University.

Countermeasures: Developer

- Format string are used by many other functions
  - e.g., *fprintf()*, *springf()*, *snprintf()*, *vprintf()*, *vfprintf()*, *vsprintf()*, and *vsnprintf()*
  - those are C functions; other languages have similar functions that use format strings
- *Good program habit*: ***avoid using*** *untrusted user inputs for format strings* in functions like *printf, sprintf, fprintf, vprintf, scanf, vfscanf*

```
// Vulnerable version (user inputs become part of the format string):
      sprintf(format, "%s %s", user_input, ": %d");
      printf(format, program_data);


// Safe version (user inputs are not part of the format string):
      strcpy(format, "%s: %d");
      printf(format, user_input, program_data);
```

  - *ask users for* ***data input****, but not for* ***code***

Countermeasures: Compiler

- Compilers can detect potential format string vulnerabilities

```
#include <stdio.h>

int main()
{
   char *format = "Hello  %x%x%x\n";

   printf("Hello %x%x%x\n", 5, 4);     ①
   printf(format, 5, 4);               ②

   return 0;
}
```

- use two compilers to compile the program: *gcc* and *clang*
- we can see that there is a mismatch in the format string (line ①)
- none of them report line ②

```
$ gcc test_compiler.c
test_compiler.c: In function main:
test_compiler.c:7:4: warning: format %x expects a matching unsigned
    int argument [-Wformat]

$ clang test_compiler.c
test_compiler.c:7:23: warning: more '%' conversions than data
    arguments
      [-Wformat]
   printf("Hello %x%x%x\n", 5, 4);
                      ~^
1 warning generated.
```

- with default settings, both compilers gave warning for the first *printf()*
- no warning was given out for the second *printf()*

Countermeasures: Compiler

- Compilers can detect potential format string vulnerabilities

```
#include <stdio.h>

int main()
{
   char *format = "Hello  %x%x%x\n";

   printf("Hello %x%x%x\n", 5, 4);     ①
   printf(format, 5, 4);               ②

   return 0;
}
```

- use two compilers to compile the program: *gcc* and *clang*
- we can see that there is a mismatch in the format string (line ①)

```
$ gcc -Wformat=2 test_compiler.c
test_compiler.c:7:4: ... (omitted, same as before)
test_compiler.c:8:4: warning: format not a string literal, argument
    types not checked
[-Wformat-nonliteral]

$ clang -Wformat=2 test_compiler.c
test_compiler.c:7:23: ... (omitted, same as before)
test_compiler.c:8:11: warning: format string is not a string literal
      [-Wformat-nonliteral]
   printf(format, 5, 4);
          ^~~~~~
2 warnings generated.
```

- if we attach –Wformat=2 option in compiler command, both of them warm the developer
  - format string vulnerability

Countermeasures: Address Randomization

- If a program contains a vulnerable *printf()*, to access or modify the program's state, attackers still need to know the address of the targeted memory
- Turning on address randomization on a Linux system can make the task difficult for attackers, as it is more difficult to guess the right address
- ***Address Randomization***: randomly arranging the memory addresses of key data areas in a process, such as the stack, heap, and libraries, making it harder for attackers to predict and exploit them