Sistemas de Software Seguros
Segurança de Software
2024/2025
Class Project: Experiments with Format Strings

1. Format Strings

a) Start by creating a program with the following C code and store it in a file with name `myFormatStr.c`:

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

int main (int argc, char **argv)
{
    char secret[20];
    char buf[32];

    strncpy(secret, "secret!!", sizeof(secret));

    strncpy(buf, argv[1], sizeof(buf));
    printf(buf);

    printf("\n");
    printf("locations: buf = %lu secret = %lu\n",
             (unsigned long) buf, (unsigned long) secret);
    fflush(stdout);
}
```

b) Compile the program in the following way:

```
gcc -fno-stack-protector -fno-asynchronous-unwind-tables myFormatStr.c -o myFormatStr
```

c) Run the program in the following way:

```
./myFormatStr 10
```

d) Understand the output from the above execution and relate it with the code you wrote in point a).

e) Run the program again, but now print the contents of the *rsi* register when `printf(buf)` is called. Provide the command and explain it. (NOTE: you may want to have a look at the assembly code; and recall the format string conversion specifications)

f) Run the program again, but now print the contents of the first 8 bytes in the stack right above the `rip` that was stored when calling `printf(buf)`. Provide the command and explain it. (NOTE: you might need to use smaller format conversion specifications, like `%x` or `%lx`).

g) Run the program again, but now print enough of the stack to get the value of *argc* parameter of `main()`. Provide the command and explain it. (NOTE: if you have not done so, you might need to investigate the assembly code; use the gcc command from last class).

h) Run the program again, but now try to find a way to print the "secret!!" string stored in variable *secret*. Explain if you can achieve this objective **using the format string conversion specifications**. (NOTE: if you are using conversion specifiers that print the content of the stack in hexadecimal, you will need to use an ASCII table to see the string. For example, the letter "s" corresponds to 73 in hexadecimal).

Delivery of the Report

The output of the class project is a report answering all the questions and including the justifications for the responses. Each group should deliver the report either by submitting it in the course Moodle page or, if there is some difficulty with this method, by emailing it to the professor of the TP class. The file type should be a pdf.

Deadline: 4 November 2024 (there will be no extensions)

==========
The following exercise is optional!
==========

Challenge exercise: Format Strings + Buffer Overflow

a) Now create the following program and store it in a file with the name `overwrite.c`. This is a game that you win when you guess a number generated randomly:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void welcome(char *s)
{
    char buf[128];
    bzero(buf, sizeof(buf));
    snprintf(buf, sizeof(buf), "Welcome %s", s);
    printf(buf);
}

int main (int argc, char **argv)
{
    int  decision;
    char msg[30];
    int  len = sizeof(msg);

    welcome(argv[1]);

    srandom((unsigned int)time(NULL));
    decision = (random() == atoi(argv[2])) ? 1 : 0;
    snprintf(msg, len, "with value %s", argv[2]);

    if (decision)
        printf("\nYOU WON %s! GREAT!!!!!\n", msg);
    else printf("\nYou lose %s )-:\n", msg);

    printf("var len = (%d, %p), msg = %p, decision = %p\n",
        len, &len, msg, &decision);

}
```

b) Compile the program in the following way:

```
gcc -fno-stack-protector -fno-asynchronous-unwind-tables overwrite.c -o overwrite
```

c) Run the program in the following way:

```
./overwrite luis 1234
```

d) Understand the output from the above execution and relate it with the code you wrote in point a).

e) Now you should find a way to exploit the existing vulnerabilities in the program to “always win”! (i.e., the output should include something like “`YOU WON with value` ….”).

NOTE: You are free to attack the program in any way you think. Below, I give a few hints that might help you perform the attack.

- Notice that function `welcome` has a format string vulnerability. Use the first argument of the program to exploit the vulnerability to print the contents of the stack.
- Notice that the same vulnerability could be exploited to write a value in a specific place in memory (recall the "%n" conversion specifier of the format string). You could take advantage of this capability to increase the value of variable `len` to become higher than 30.
- Notice that if `len` is larger than the original value, you can perform a buffer overflow on `msg` with the second argument of the program. Maybe you can use this buffer overflow to change the value of `decision` and win the game!
- Also recall that if you want to write an address in the command line, you should use the following trick. Imagine that you want to write address "0x7fffffffddcc"

```
./overwrite  .......`printf "\xcc\xdd\xff\xff\xff\x7f"`  123
```

f) The game developers detected your attack. To prevent it, they have modified two lines of the above program so that they become:

```
decision = (random() == atoi(argv[2])) ? 100 : 0;

snprintf(msg, len, "with value %s", argv[2]);

if (decision == 100)
```

Can you alter your attack to ensure that you always win the game? Good work!