Format String Problems

Software Security

Overview

- The issue starts with unvalidated user-supplied data
  - Typical… lots of bugs start this way
- Can allow an attacker to write to memory
  - Arbitrary code execution
- Most severe in C/C++
  - Other languages might not lead to code execution

Format String Review

- printf, scanf
  - Print formatted data, read data according to format
- `printf("The answer is %d\n", num);`
  - Format string: `The answer is %d\n`
    - Format specifier: `%d`
  - Additional argument: `num`
- Format specifiers generally used to print data in a variable in a specific format

The problem

- Taking user input as a format string

- Might seem harmless, but format strings can be written to do…stuff…
  - Bad stuff

Example 1

- In this example…
- Printf expects there were two values pushed onto the stack
  - Even though there weren't any
  - That's what it's printing out

```c
#include <stdio.h>

int main(int argc, char *argv[])
{

    if(argc > 1)
        printf(argv[1]);

    return 0;
}
```

- Top two values on the stack as hex
  - e6832b78
  - e6832b90

```
Cody-MBP13:Desktop cody$ ./a.out "%x %x"
e6832b78 e6832b90Cody-MBP13:Desktop cody$
```


Example 2

- Dump a variable

Example 3

- Write to memory with printf

- %n - stores how many characters were written

Example 4

- Changing a variable of interest

Example 5

- More specifically changing a variable of interest

What to look for

- `printf(user_input);`
- Never put user input directly in a printf alone!
- Better solution:
  - `printf("%s", user_input);`

Compilers can help!

- Follow their advice!

```
test.c:9:10: warning: format string is not a string literal
      (potentially insecure) [-Wformat-security]
                printf(argv[1]);
                       ^~~~~~~
test.c:9:10: note: treat the string as an argument to avoid this
                printf(argv[1]);
                       ^
                       "%s",
1 warning generated.
```