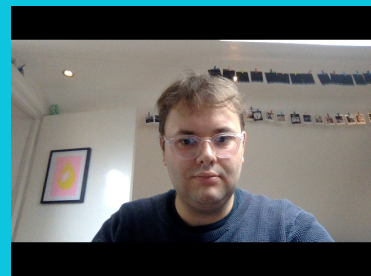
University of BRISTOL

Format String Vulnerability

Joseph Hallett



What is it?

A vulnerability in C-style print functions…
Allows an attacker to read from the stack
…and other places
Allows an attacker to write to any memory address on the stack

First spotted in the wild in 1999 (proftpd)
https://seclists.org/bugtraq/1999/Sep/328

Format string functions

```
int printf(const char * restrict format, ...);
int fprintf(FILE * restrict stream, const char * restrict format, ...);
int sprintf(char * restrict str, const char * restrict format, ...);
int snprintf(char * restrict str, size_t size, const char * restrict format, ...);
int asprintf(char **ret, const char *format, ...);
int dprintf(int fd, const char * restrict format, ...);
```


How are they normally used?

```
#include <stdio.h>

int main(void) {
        printf("Hello %s!\n", "Class");
        return 0;
}
```

The program will print *“Hello Class”*

What happens if there aren't enough arguments?

```
#include <stdio.h>

int main(void) {
        printf("Hello %s!\n");
        return 0;
}
```

```
cc     test.c   -o test
test.c:4:17: warning: more '%' conversions than data arguments [-
Wformat-insufficient-args]
        printf("Hello %s!\n");
                       ~^
```

bristol.ac.uk

What happens if the compiler can't know for sure?

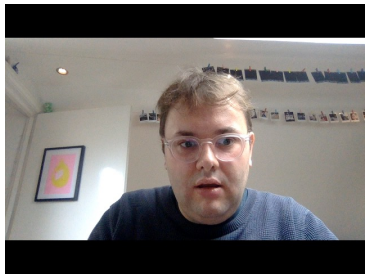
```
int main(void) {
        char format_string[80];
        fgets(format_string, sizeof format_string, stdin);
        printf(format_string);
        return 0;
}

cc     test.c   -o test
test.c:6:9: warning: format string is not a string literal
(potentially insecure) [-Wformat-security]
        printf(format_string);
               ^~~~~~~~~~~~~
```


Why is this dangerous?

What happens if the user inputs a string with a format specifier in it?

For example: `%08x %08x %08x %08x ?`
The print function will assume they are in the usual place (according to the calling convention)



For example…

```
% ./test
%08x %08x %08x %08x
000120a8 00000000 1d541eac 806a5f48
```

These are values taken from the stack…

Uh-oh…

From `man 3 printf`:

> **n** The number of characters written so far is stored into the integer indicated by the int * (or variant) pointer argument. No argument is converted. The format argument must be in write-protected memory if this specifier is used; see SECURITY CONSIDERATIONS below.

Fixes?

Listen to your compiler warnings!

Some modern systems remove %n (Windows)
Others log its use (OpenBSD)

But legacy code still uses it…
Try setting your phone name to `%08x` ;-)

Another Example to try at home

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

int pin=12345;

int check(int upin, int spin) {
    if (2*upin == spin)
        return 1;
    else return -1;
}

int main(int argc, char *argv[]) {
    char welcome[50];
    char name[40];
    int upin,auth;
    printf("Enter you name followed by your pin:\n");
    scanf("%39s%d",name, &upin);
    sprintf(welcome,"Hello %s",name);

    auth=check(upin,pin);
    printf(welcome);
    if(auth==-1) {
        printf("Sorry, try again...\n");
        exit(0);
    }
    return 0;
}
```