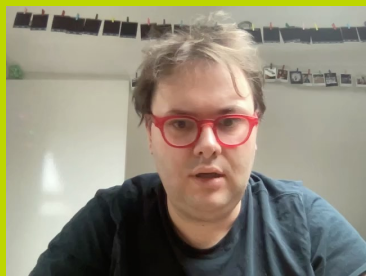
University of BRISTOL

COMS20012: Format Strings

Joseph Hallett



bristol.ac.uk

What is all this about then?

```
$ cat example3.c && make example3
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("This program is called: ");
  printf(argv[0]);
  printf("\n");

  return 0;
}
cc  -I/opt/homebrew/opt/openjdk/include   example3.c   -o example3
example3.c:5:10: warning: format string is not a string literal (potentially insecure) [-Wformat-security]
  printf(argv[0]);
         ^~~~~~~
example3.c:5:10: note: treat the string as an argument to avoid this
  printf(argv[0]);
         ^
         "%s",
1 warning generated.
$
```

man 3 printf

BUGS top

Because **sprintf**() and **vsprintf**() assume an arbitrarily long string, callers must be careful not to overflow the actual space; this is often impossible to assure. Note that the length of the strings produced is locale-dependent and difficult to predict. Use **snprintf**() and **vsnprintf**() instead (or asprintf(3) and vasprintf(3)).

Code such as **printf**(*foo*); often indicates a bug, since *foo* may contain a % character. If *foo* comes from untrusted user input, it may contain **%n**, causing the **printf**() call to write to memory and creating a security hole.

%n?

- LibC's *printf* function handles formatted output…
  - %s prints a string…
  - %d or %i prints a decimal integer…

n The number of characters written so far is stored into the integer pointed to by the corresponding argument. That argument shall be an *int* *, or variant whose size matches the (optionally) supplied integer length modifier. No argument is converted. (This specifier is not supported by the bionic C library.) The behavior is undefined if the conversion specification includes any flags, a field width, or a precision.

Say you wanted to do columnar output (and were really weird)

- Say we have an address book we want to print like the following

```
Joseph Hallett: 3.16 MVB, Bristol
                01234 567890
                Born 1987-11-08
```

```c
#include <stdio.h>

void print_address_book(char *name,
                        char *data[]) {
  int align;
  printf("%s: %n", name, &align);
  if (data != NULL)
    do {
      printf("%s\n", *data);
      for (int i = 0; i < align; i++)
        putchar(' ');
    } while (*(++data) != NULL);
  putchar('\n');
}
```

So why is this dangerous?

- Say you control the format string…
- Format string arguments typically passed via the stack
  - (once you've printed a few… if not using cdecl)

- What happens if you print more arguments than you have?

```
#include <stdio.h>
int main(void) {
  int target = 0x31337;
  char *args =
    "01: %p\n02: %p\n03: %p\n04: %p\n"
    "05: %p\n06: %p\n07: %p\n08: %p\n"
    "09: %p\n0a: %p\n0b: %p\n0c: %p\n"
    "0d: %p\n0e: %p\n0f: %p\n10: %p\n";
  printf(args);
  return 0;
}
```

Hmm!

```
$ ./example5
01: 0x16f147700
02: 0xd6eb3c
03: 0x100cbbf44
04: 0x31337
05: 0x16f147850
06: 0x100d690f4
07: 0x0
08: 0x0
09: 0x0
0a: 0x0
0b: 0x0
0c: 0x0
0d: 0x100dc8138
0e: 0x0
0f: 0x4d55545a
10: 0x20a000000000
```

```c
#include <stdio.h>
int main(void) {
  int target = 0x31337;
  char *args =
    "01: %p\n02: %p\n03: %p\n04: %p\n"
    "05: %p\n06: %p\n07: %p\n08: %p\n"
    "09: %p\n0a: %p\n0b: %p\n0c: %p\n"
    "0d: %p\n0e: %p\n0f: %p\n10: %p\n";
  printf(args);
  return 0;
}
```

So why is this dangerous?

- By careful choice of format string we can write to arbitrary addresses somewhere after the stack pointer…

- This could be a local variable…
  - Data corruption
- This could be return address…
  - Control flow corruption and arbitrary code execution

Going further…

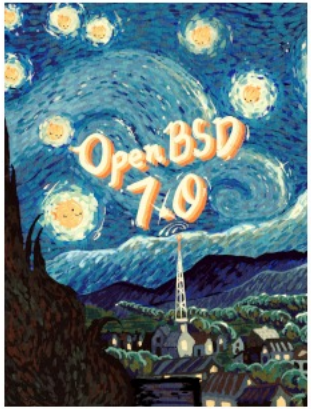
- See *Exploiting format string vulnerabilities* by scut/team teso
- See *Exploiting a format string bug in Solaris CDE* (Phrack Magazine, Volume 0x16, Issue 0x46) by Marco Ivaldi

- *(or take Systems and Software Security in Year 4 ;-))*

Why is this even still a thing?

- We don't need dangerous cruft from the 1970s in our programming languages just to save a call to `strlen()` anymore.
  - We didn't really back then either…

OpenBSD 7.0



Released Oct 14, 2021. (51st OpenBSD release)

7.0 Song: "The Style Hymn".
Artwork by Natasha Allegri.

- See the information on the FTP page for a list of mirror machines.
- Go to the pub/OpenBSD/7.0/ directory on one of the mirror sites.
- Have a look at the 7.0 errata page for a list of bugs and workarounds.
- See a detailed log of changes between the 6.9 and 7.0 releases.

- signify(1) pubkeys for this release:

```
openbsd-70-base.pub:    RWR3KL+gSr4QZ5mOvKhcOOgGe61ogHp5PyBOj2RrmyCpqchk9A7NVPzh
openbsd-70-fw.pub:      RWS8nd7vy+I+fRHtnpxVBeX+P+9rBqJMPvSU6z8LYyAv5p73WcdFXs3B
openbsd-70-pkg.pub:     RWR3iauEtA8/bLN/zfIQhOc5ramL/fARX72S6xw8BwAUebxik7KioCvL
openbsd-70-syspatch.pub:RWSD33kMDKsQH8j0Q8FzfYk+vsgTKiP8Q5DcrkQQtrZoWg48yxUQgLxU
```

What's New

This is a partial list of new features and systems included in OpenBSD 7.0. For a comprehensive list, see the changelog leading to 7.0.

- Security improvements:
  - Moved objcopy to base set to allow KARL to work on all installs.
  - Added unveil(2) calls to xterm in the case where there are no exec-formatted or exec-selected resources
  - Changed usage of %n from a syslog warning to syslog and abort for printf(3) (and associated variants).
  - Made kernel stop all threads when terminating via pledge_fail().