Low level attacks
Format string vulnerabilities
(part 1)

Mario Alviano

University of Calabria, Italy

A.Y. 2019/2020

Introduction

- Almost all C programs use `printf` or derivatives
- The format function is defined in the ANSI C standard
- It is variadic
  - First argument is the *format string*
  - Extra parameters follows

Introduction

- Almost all C programs use `printf` or derivatives
- The format function is defined in the ANSI C standard
- It is variadic
  - First argument is the *format string*
  - Extra parameters follows

Noncompliant code example

```
void foo(char* from_user) {
   printf(from_user);
}
```

Compliant code example

```
void foo(char* from_user) {
   printf("%s", from_user);
}
```

The format function family

- `printf`: print to STDOUT

The format function family

- `printf`: print to STDOUT
- `fprintf`: print to FILE

The format function family

- `printf`: print to STDOUT
- `fprintf`: print to FILE
- `sprintf`: print into a string

The format function family

- `printf`: print to STDOUT
- `fprintf`: print to FILE
- `sprintf`: print into a string
- `snprintf`: print into a string with length checking

The format function family

- `printf`: print to STDOUT
- `fprintf`: print to FILE
- `sprintf`: print into a string
- `snprintf`: print into a string with length checking
- `err*`: print errors

The format function family

- `printf`: print to STDOUT
- `fprintf`: print to FILE
- `sprintf`: print into a string
- `snprintf`: print into a string with length checking
- `err*`: print errors
- `warn*`: print warnings

Format string

- May include format parameters
  - `%d`: decimal (`int`)
  - `%u`: unsigned decimal (`unsigned int`)
  - `%x`: hexadecimal (`unsigned int`)
  - `%s`: string (`const char*`)
  - `%n`: number of bytes printed so far (`int*`)

Format string

- May include format parameters
  - `%d`: decimal (`int`)
  - `%u`: unsigned decimal (`unsigned int`)
  - `%x`: hexadecimal (`unsigned int`)
  - `%s`: string (`const char*`)
  - `%n`: number of bytes printed so far (`int*`)
- After `%` there may be a width
- The width may be preceded by a filler (eg. `0`)
- Extra parameters are passed via the stack

Format string

- May include format parameters
  - `%d`: decimal (`int`)
  - `%u`: unsigned decimal (`unsigned int`)
  - `%x`: hexadecimal (`unsigned int`)
  - `%s`: string (`const char*`)
  - `%n`: number of bytes printed so far (`int*`)
- After `%` there may be a width
- The width may be preceded by a filler (eg. `0`)
- Extra parameters are passed via the stack

Alert

We have format parameters to read and modify the stack!

Reading the stack

- Use (`%08x.`)+ to nicely print the stack

Reading the stack

- Use (`%08x.`)+ to nicely print the stack

Example

- Try `printf.c`

Reading the stack

- Use (`%08x.`)+ to nicely print the stack

Example

- Try `printf.c`
- Use `gdb`
- What information is leaked?

Reading the stack

- Use (`%08x.`)+ to nicely print the stack

Example

- Try `printf.c`
- Use `gdb`
- What information is leaked?
- Is the format string itself in the stack?

Write via format strings

- We can use `%n` to write into memory

Write via format strings

- We can use `%n` to write into memory
- We can use format parameters to increase the number of printed bytes

Write via format strings

- We can use `%n` to write into memory
- We can use format parameters to increase the number of printed bytes
- Try `printf_n.c`

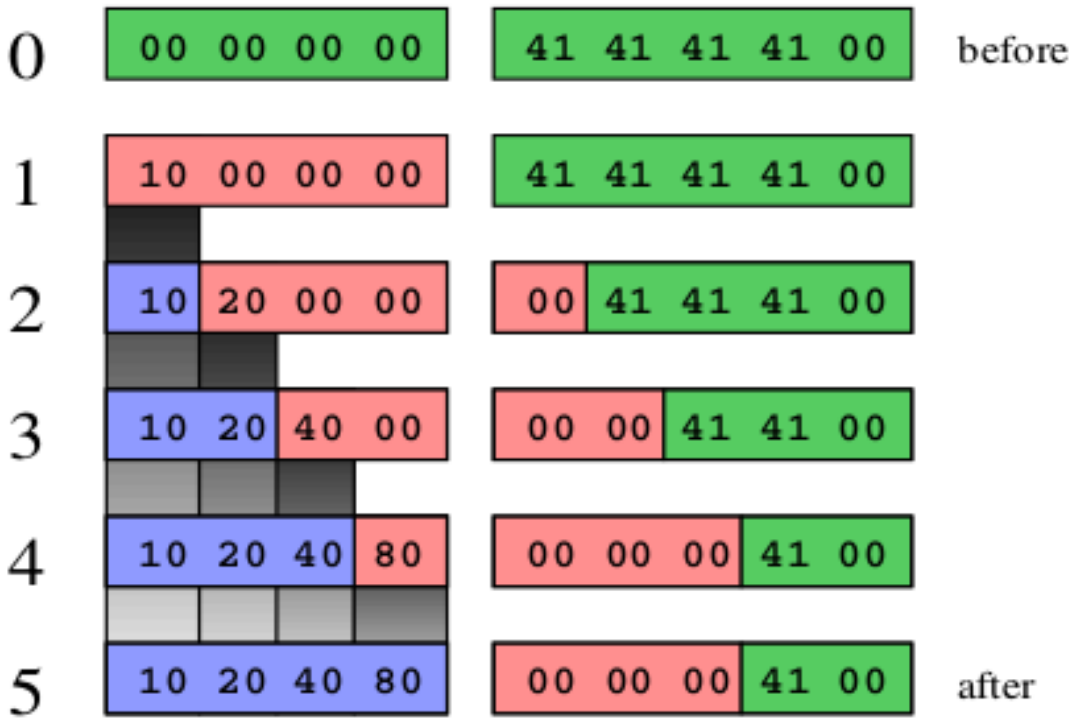
Write via format strings

- We can use `%n` to write into memory
- We can use format parameters to increase the number of printed bytes
- Try `printf_n.c`
- Remember that our machines are little-endian

Write via format strings

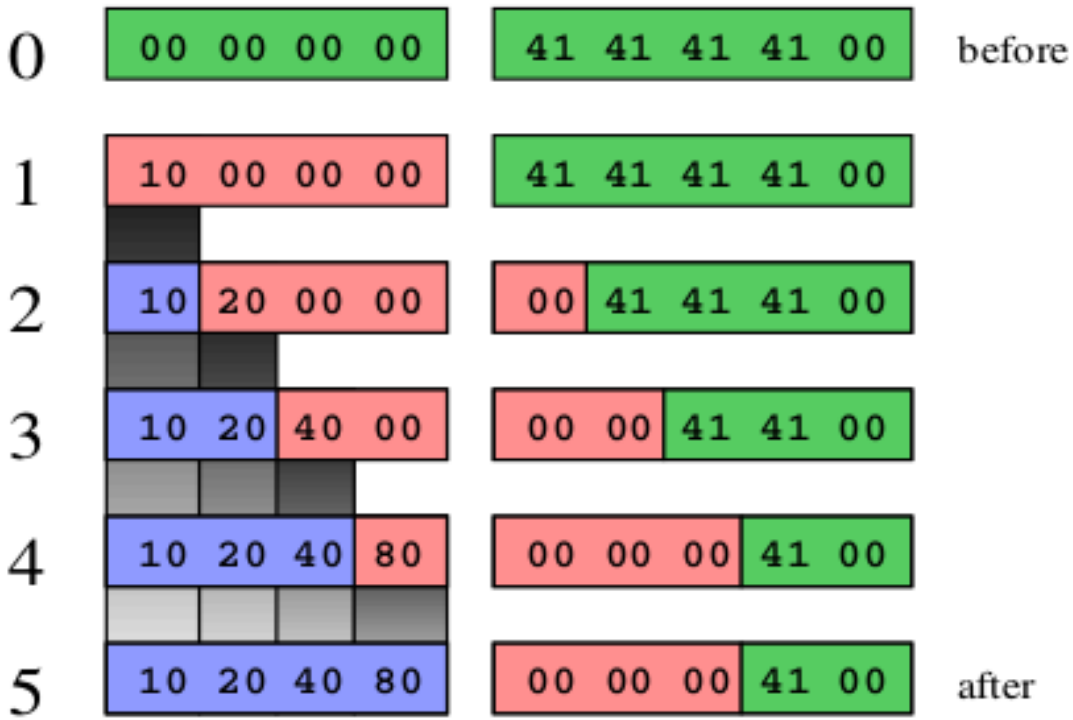
- We can use `%n` to write into memory
- We can use format parameters to increase the number of printed bytes
- Try `printf_n.c`
- Remember that our machines are little-endian
- Check the value of variable `n` with `gdb`

Four steps writing (1)



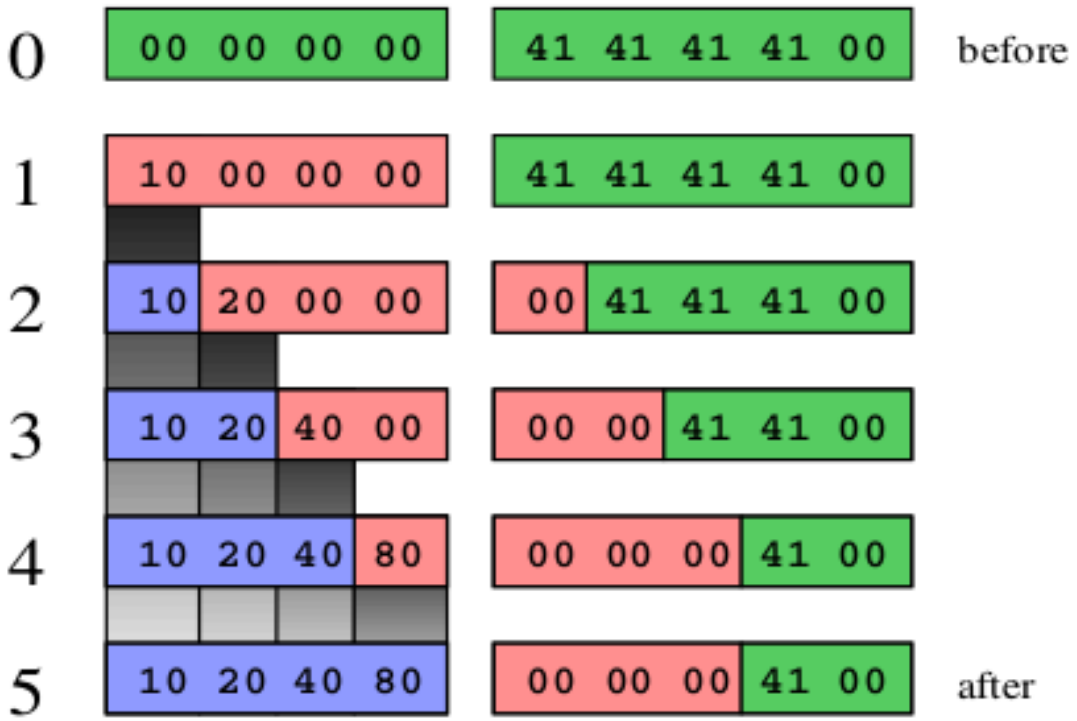
- Try `printf_write.c`

Four steps writing (1)



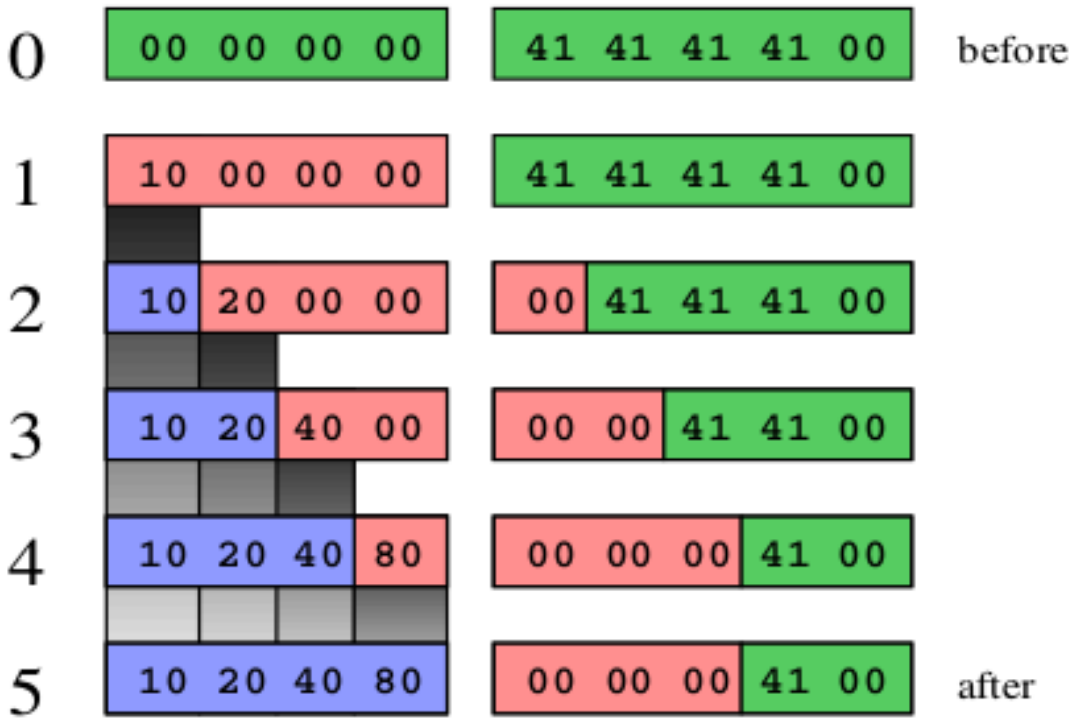
- Try `printf_write.c`
- Variable `foo` is written one byte at time

Four steps writing (1)



- Try `printf_write.c`
- Variable `foo` is written one byte at time
- The memory after the variable is also altered

Four steps writing (1)



- Try `printf_write.c`
- Variable `foo` is written one byte at time
- The memory after the variable is also altered
- If that memory is not important, it is OK!

Four steps writing (2)

- For exploitation, we have to use a single format string

Four steps writing (2)

- For exploitation, we have to use a single format string
- Try `printf_write2.c`

Four steps writing (2)

- For exploitation, we have to use a single format string
- Try `printf_write2.c`
- What if we want to write `80402010`?

Four steps writing (2)

- For exploitation, we have to use a single format string
- Try `printf_write2.c`
- What if we want to write `80402010`?
- It's little-endian! Overflow the least significant byte

Four steps writing (2)

- For exploitation, we have to use a single format string
- Try `printf_write2.c`
- What if we want to write `80402010`?
- It's little-endian! Overflow the least significant byte
- Try `printf_write3.c` and `printf_write4.c`

Four steps writing (3)

- How to compute those numbers?

Four steps writing (3)

- How to compute those numbers?

```
write_byte += 0x100;
already_written %= 0x100;
padding = (write_byte - already_written) % 0x100;
if (padding < 10)
        padding += 0x100;
```

Four steps writing (3)

- How to compute those numbers?

```
write_byte += 0x100;
already_written %= 0x100;
padding = (write_byte - already_written) % 0x100;
if (padding < 10)
        padding += 0x100;
```

- The biggest 32-bits unsigned integer is 4294967295

Four steps writing (3)

- How to compute those numbers?

```
write_byte += 0x100;
already_written %= 0x100;
padding = (write_byte - already_written) % 0x100;
if (padding < 10)
        padding += 0x100;
```

- The biggest 32-bits unsigned integer is 4294967295
- It is 10 ciphers

Four steps writing (3)

- How to compute those numbers?

```
write_byte += 0x100;
already_written %= 0x100;
padding = (write_byte - already_written) % 0x100;
if (padding < 10)
        padding += 0x100;
```

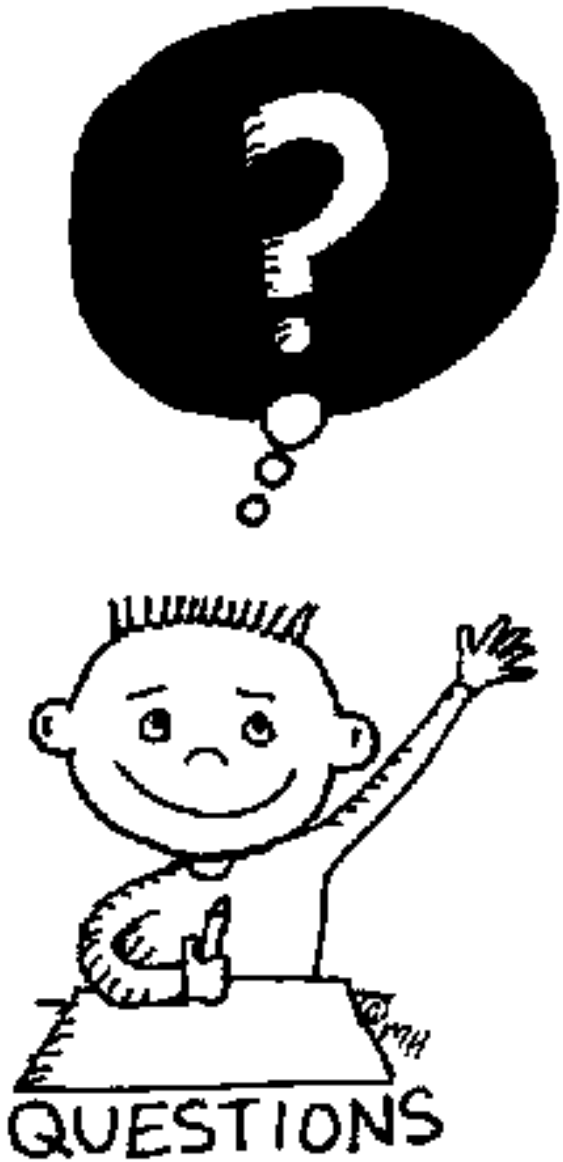
- The biggest 32-bits unsigned integer is 4294967295
- It is 10 ciphers
- Hence, the padding is at least 10 bytes

Four steps writing (3)

- How to compute those numbers?

```
write_byte += 0x100;
already_written %= 0x100;
padding = (write_byte - already_written) % 0x100;
if (padding < 10)
        padding += 0x100;
```

- The biggest 32-bits unsigned integer is 4294967295
- It is 10 ciphers
- Hence, the padding is at least 10 bytes
- Try `printf_write5.c`



END OF THE LECTURE