Sécurité des logiciels

print? pwnd!
escape? pwnd!

Samuel Thibault <samuel.thibault@u-bordeaux.fr>

Printf is evil

Printf is evil

```
int main(void) {
  char buf[128];
  while (1) {
    fgets(buf, sizeof(buf), stdin);
    printf(buf);
  }
}
```

What is the bug?

Printf is evil

```
int main(void) {
  char buf[128];
  while (1) {
    fgets(buf, sizeof(buf), stdin);
    printf("%s", buf);
  }
}
```

What is the bug?

Printf is evil

```
int main(void) {
  char buf[128];
  while (1) {
    fgets(buf, sizeof(buf), stdin);
    printf(buf);
  }
}
```

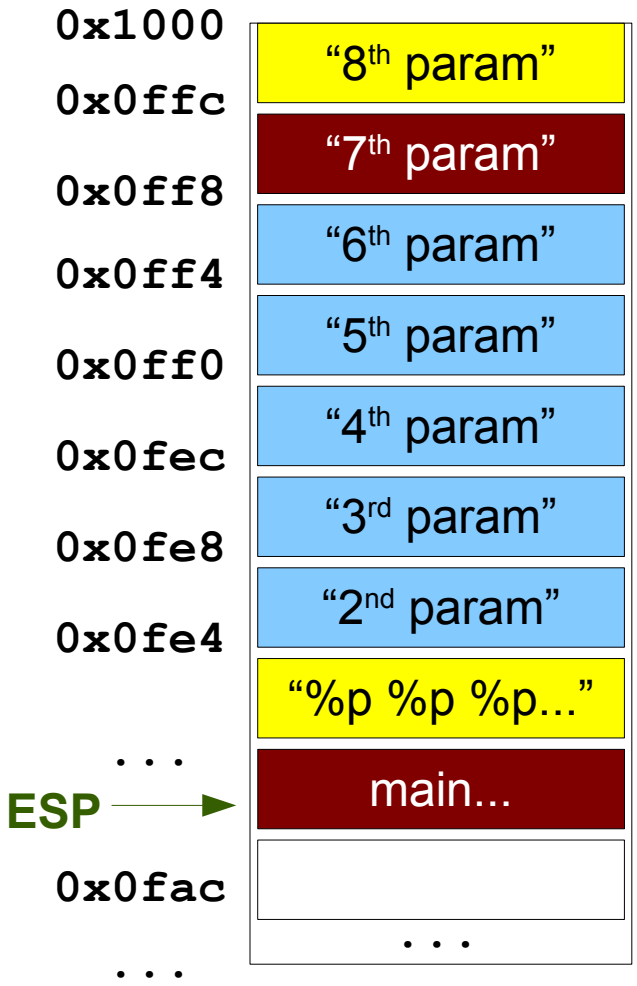
What is the bug?

Printf is evil

```
printf("%p %p %p %p %p\n");
```

Parameters do “exist”, even if none was passed!

Leaks whatever you want from the stack

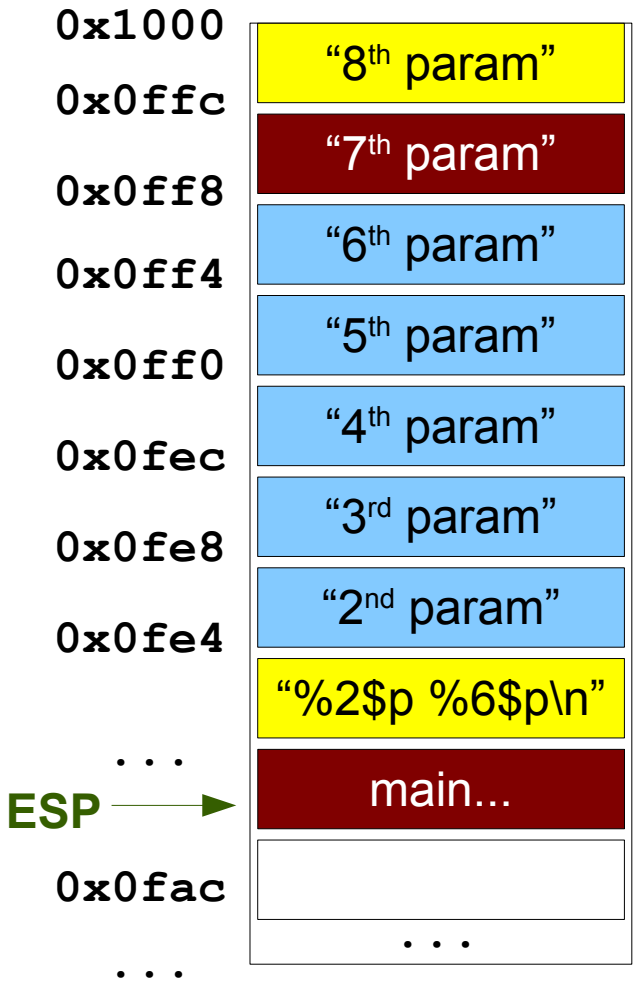


Printf is really evil

```
printf("%2$p %6$p\n");
```

You can even choose what to print!

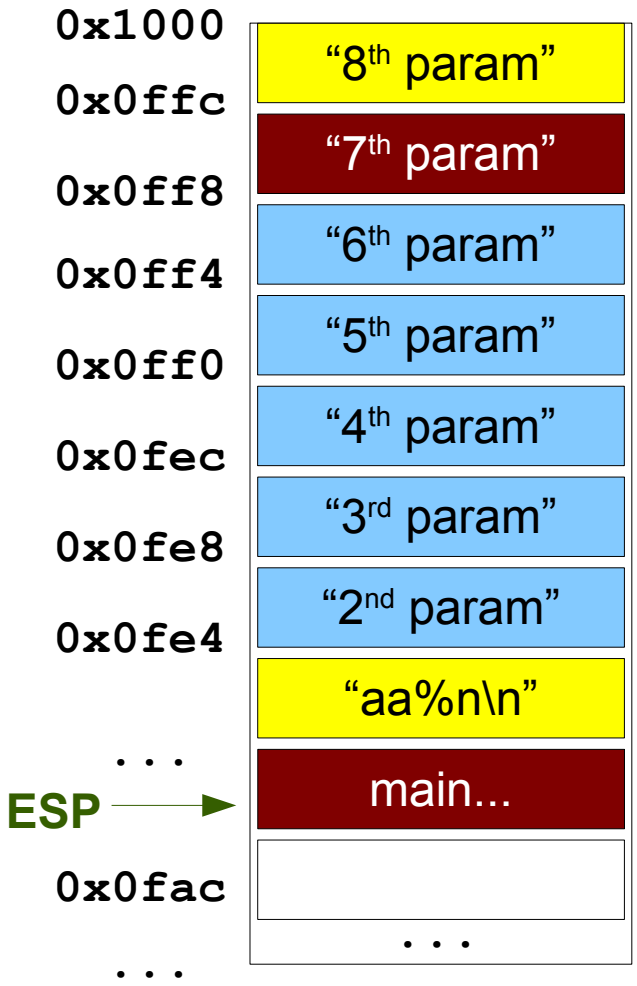
Note: `%2$p` means:
- the second parameter after the format,
- so here it is the third parameter of printf



Printf is awfully evil

```
printf("aa%n\n");
```

What the hell is that?



Printf is awfully evil

```
printf("aa%n\n");
```

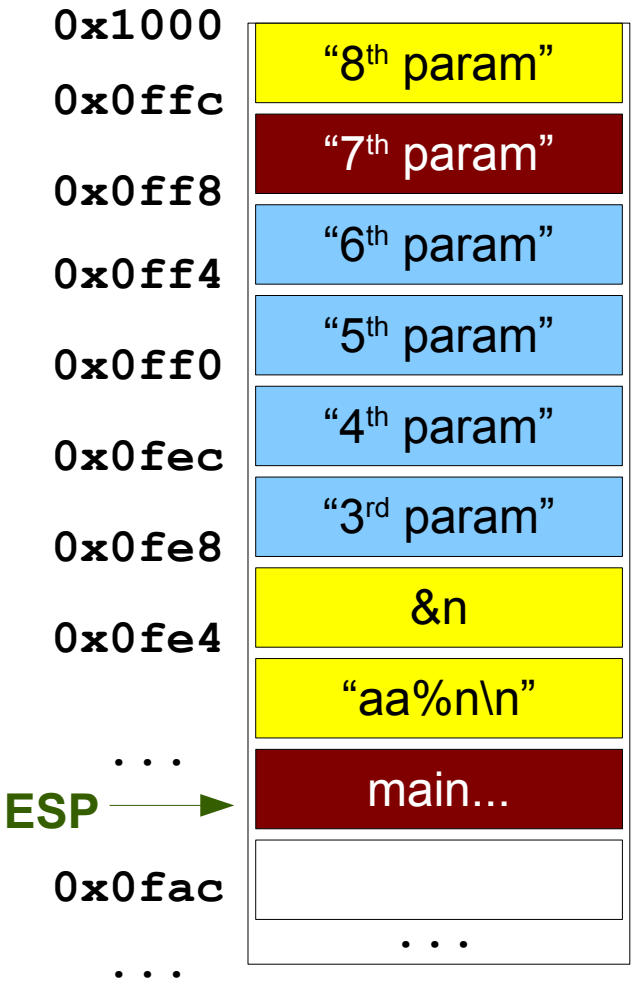
What the hell is that?

Normal use:

```
int n;
printf("aa%n\n", &n);
```

Writes in `n` the number of printed chars so far, i.e. 2 here (`aa`).

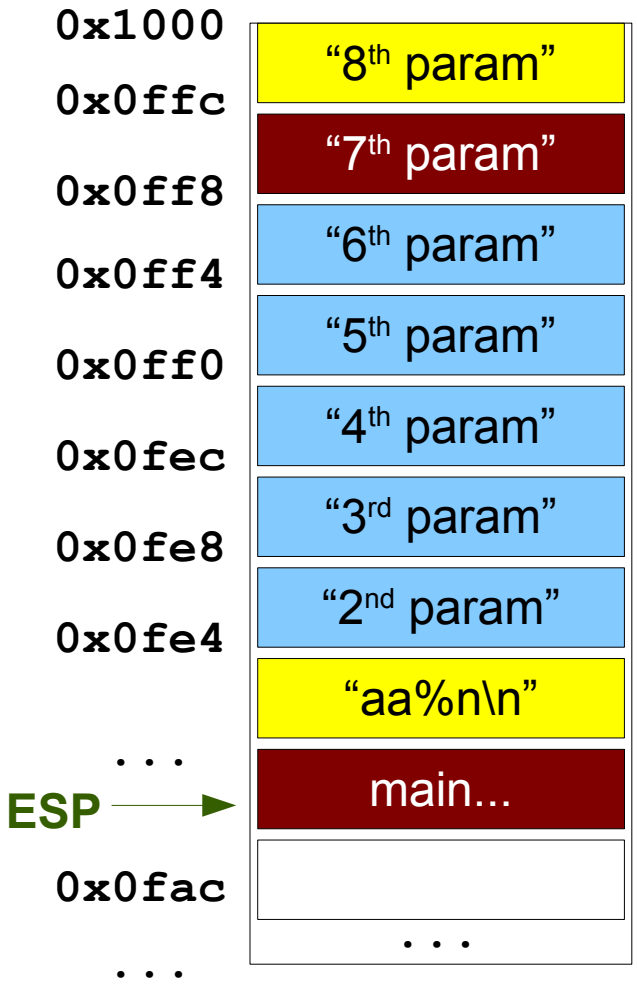
Yes, printf can write to memory



Printf is awfully evil

```
printf("aa%n\n");
```

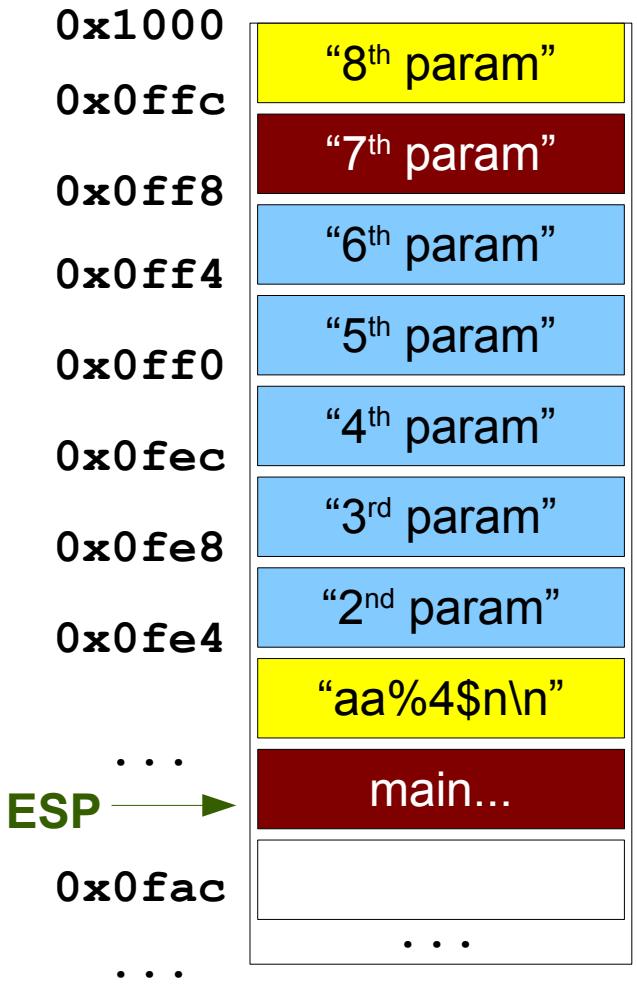
Uses 2nd parameter as address where to write 2.



Printf is awfully evil

```
printf("aa%4$n\n");
```

Uses 5th parameter as address where to write 2.

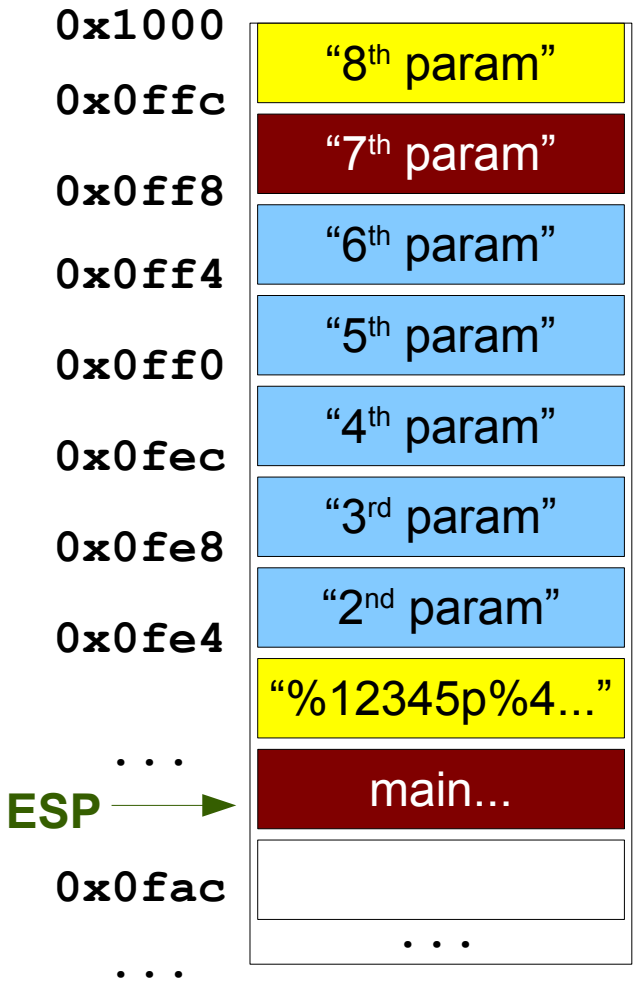


Printf is awfully evil

```
printf("%12345p%4$n\n");
```

Uses 5th parameter as address where to write 12345.

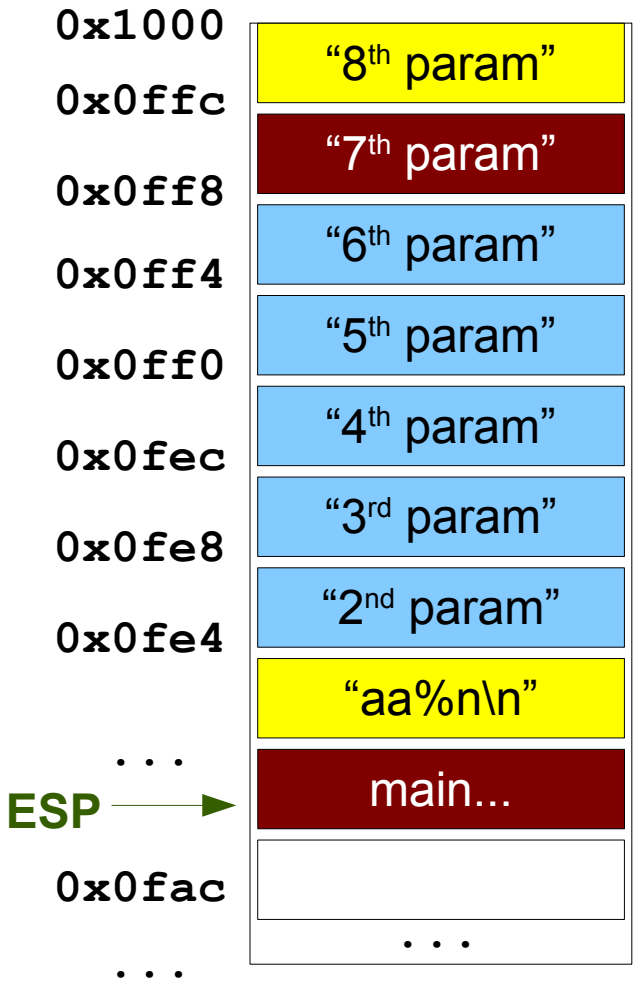
Smells really bad...



Printf is awfully evil

Remember that our victim code was

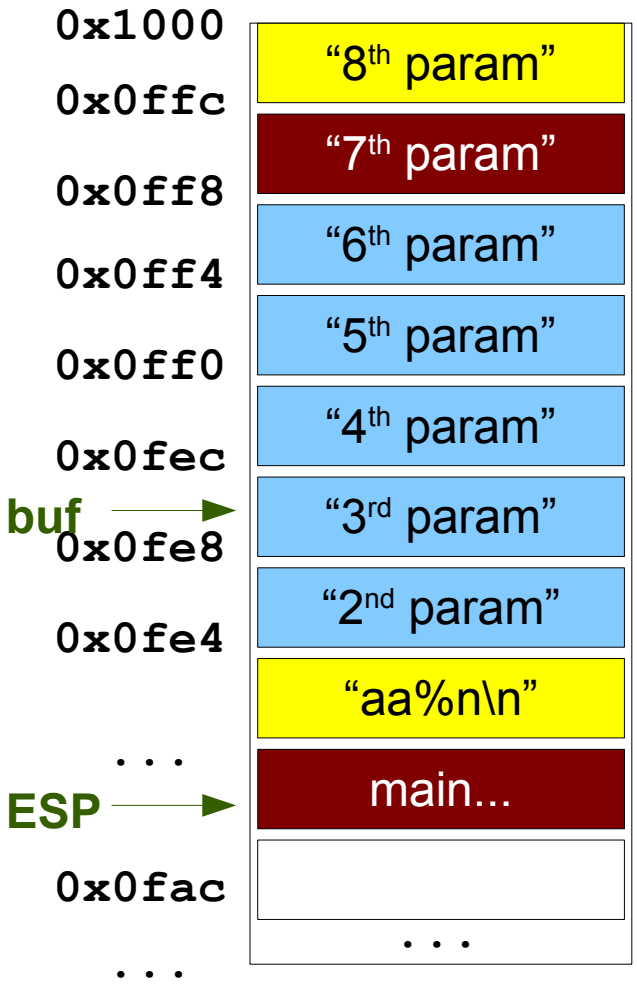
```
buf[128];
fgets(buf, sizeof(buf), stdin);
printf(buf);
```



Printf is awfully evil

Remember that our victim code was

```
buf[128];
fgets(buf, sizeof(buf), stdin);
printf(buf);
```

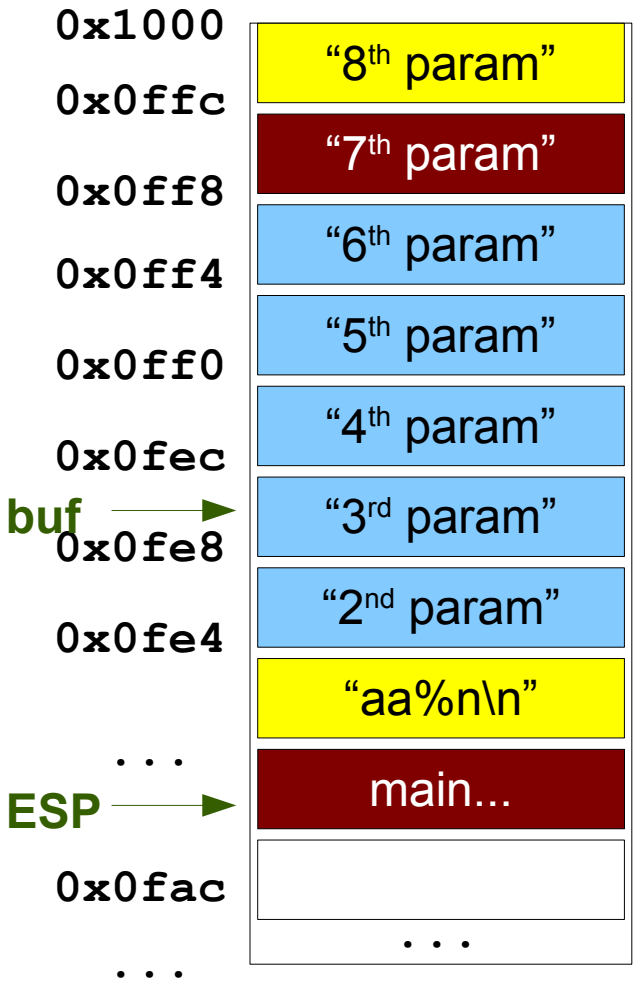


Printf is awfully evil

Remember that our victim code was

```
buf[128];
fgets(buf, sizeof(buf), stdin);
printf(buf);
```

I.e. we actually control what is on the stack...



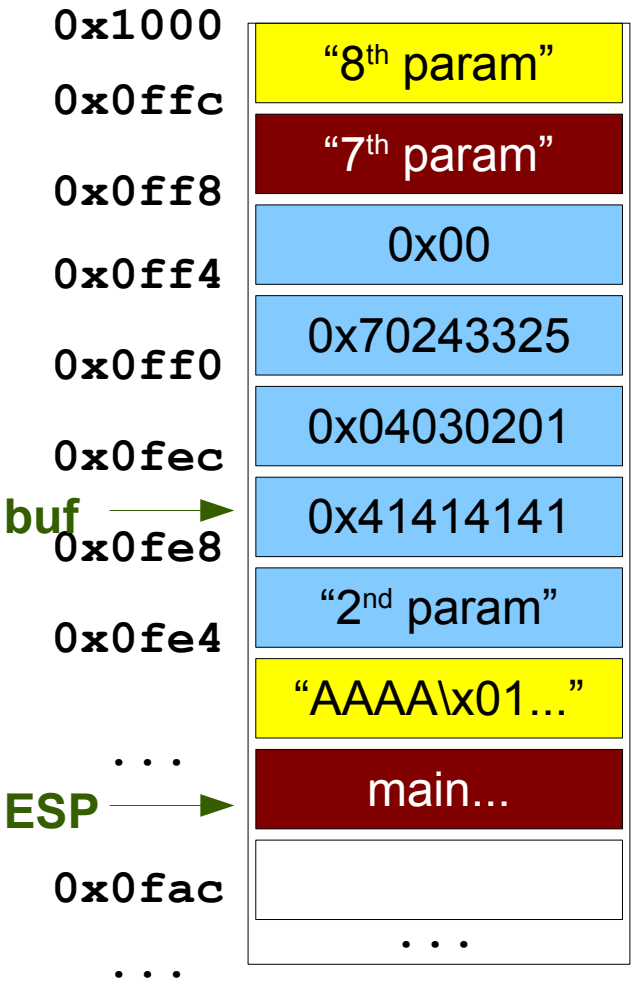
Printf is awfully evil

Remember that our victim code was

```
buf[128];
fgets(buf, sizeof(buf), stdin);
printf(buf);
```

I.e. we actually control what is on the stack...

Let's pass "AAAA\x01\x02\x03\x04%3$p"



Printf is awfully evil

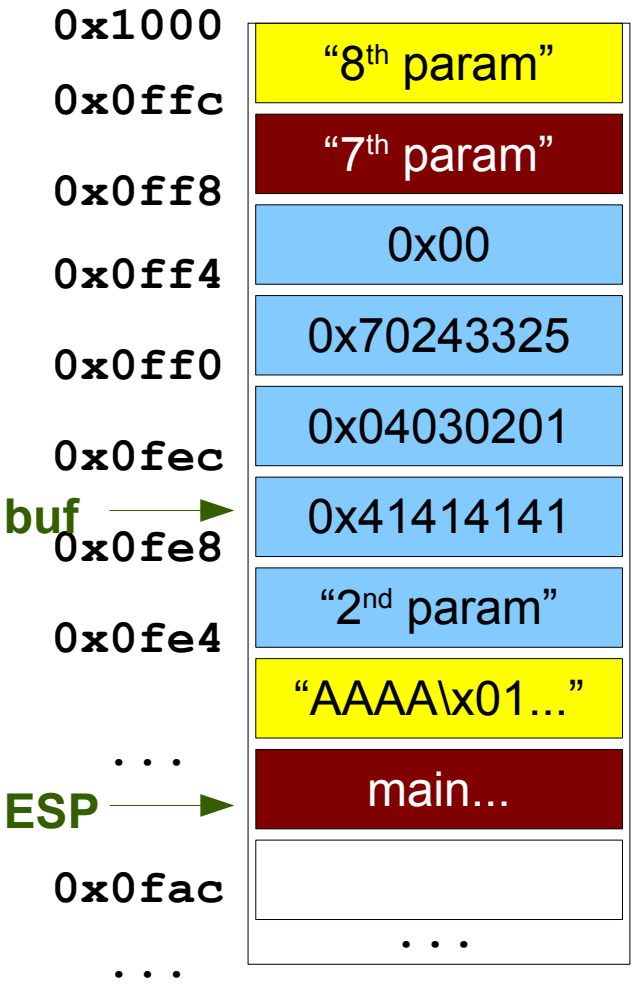
Remember that our victim code was

```
buf[128];
fgets(buf, sizeof(buf), stdin);
printf(buf);
```

I.e. we actually control what is on the stack...

Let's pass "AAAA\x01\x02\x03\x04%3$p"

Prints `0x04030201`



Printf is awfully evil

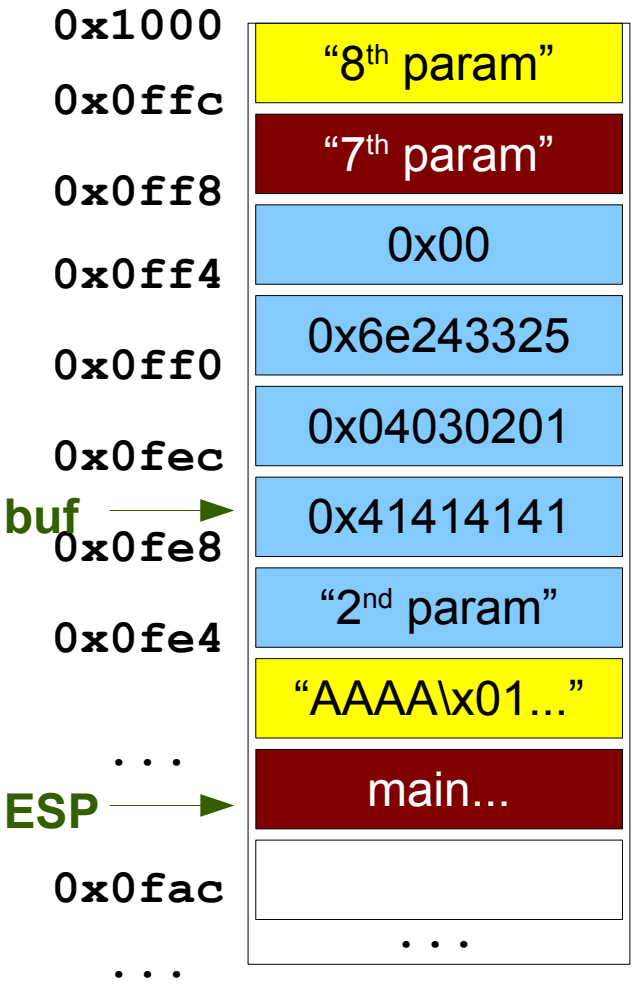
Remember that our victim code was

```
buf[128];
fgets(buf, sizeof(buf), stdin);
printf(buf);
```

I.e. we actually control what is on the stack...

Let's pass "AAAA\x01\x02\x03\x04%3$n"

Writes 8 in memory at `0x04030201`!



Printf is awfully evil

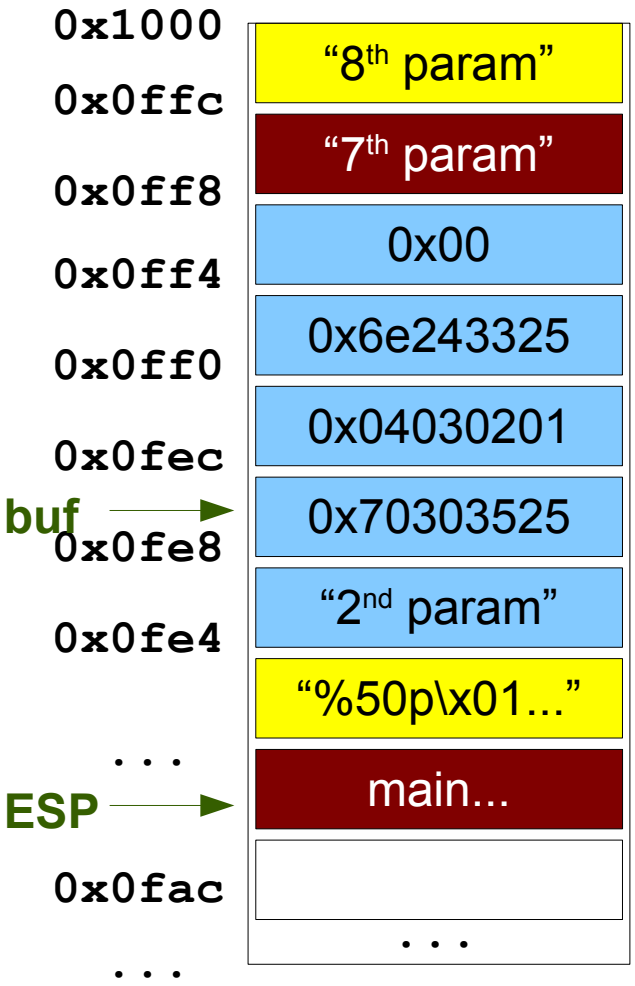
Remember that our victim code was

```
buf[128];
fgets(buf, sizeof(buf), stdin);
printf(buf);
```

I.e. we actually control what is on the stack...

Let's pass "%50p\x01\x02\x03\x04%3$n"

Writes 54 in memory at `0x04030201`!



Printf is definitely evil

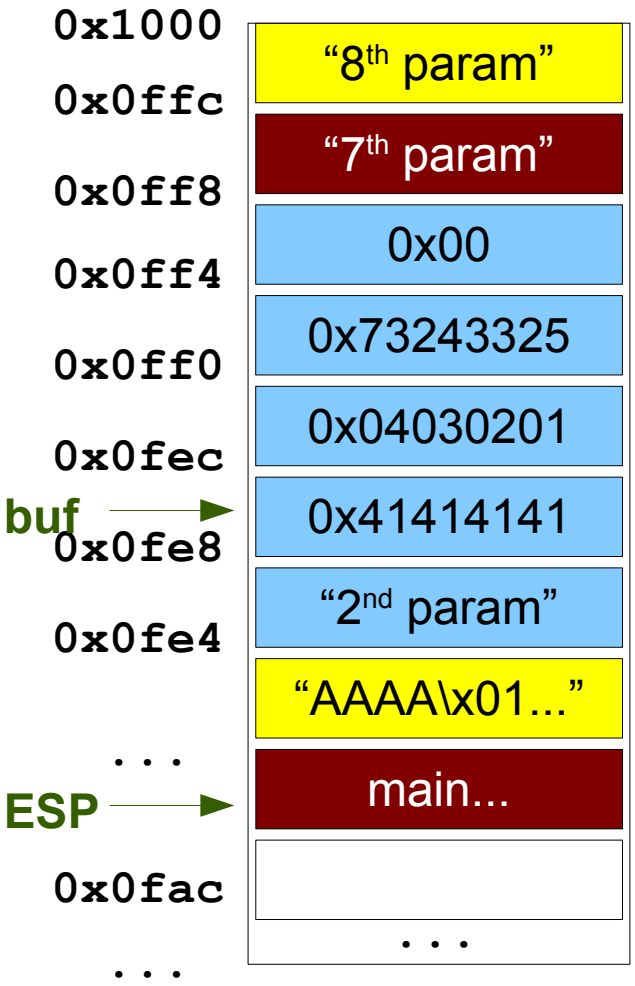
Remember that our victim code was

```
buf[128];
fgets(buf, sizeof(buf), stdin);
printf(buf);
```

I.e. we actually control what is on the stack...

Let's pass "AAAA\x01\x02\x03\x04%3$s"

Prints string from address **0x04030201**!!



Printf is definitely evil

`printf(s);` is definitely evil, never do that

Replace with `printf("%s", s);`

Escaping is tricky...

Escaping is tricky

```
printf("Hello, world!\n");
```

What about printing the " character?

Escaping is tricky

```
printf("Hello, world!\n");
```

What about printing the " character?

```
printf("\"Hello, world!\"\n");
```

\ is called an **escape character**.

Escaping is tricky

```
printf("Hello, world!\n");
```

What are issues with escaping characters?

- Escaping itself `"\\"`
- Unknown escaping `"\z"`
- Escaping at end `"\"`

The January 2021 sudo exploit was an escaping-at-end issue

Escaping is tricky

SMTP protocol (Simple Mail Transfer Protocol):

```
354 Enter message, ending with "." on a line by itself
```

But then how can the mail contain a “.” on a line by itself?

- Emit `"..."` on a line by itself

But then how can the mail contain a “..” on a line by itself?

- Emit `"..."` on a line by itself

But then how can the mail contain a “...” on a line by itself?
etc.

Escaping is tricky

SMTP protocol (Simple Mail Transfer Protocol)

Its escaping looks quite lame... But it is systematic.

Escaping is tricky

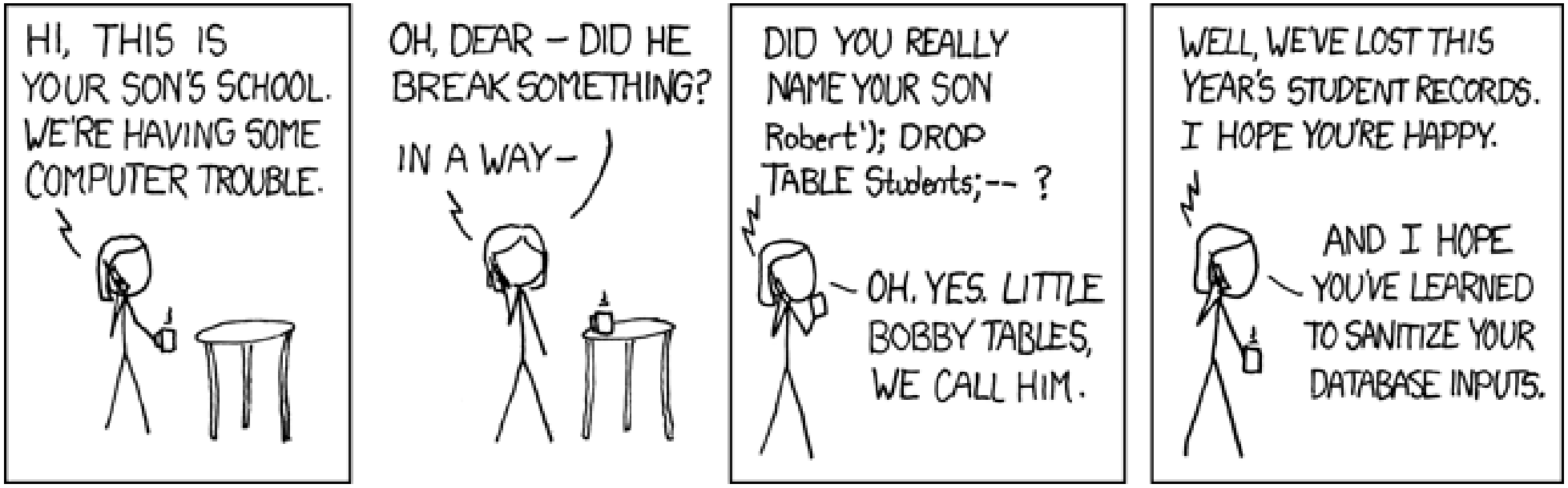
Web form to register for something

- First name
- Family name
- etc.

```
char *query;
asprintf(&query,
    "INSERT INTO people VALUES('%s', '%s')",
    first_name, last_name);
mysql_query(con, query);
```

What if I enter as name: `Samuel'); DROP ALL TABLES;--`

Escaping is tricky



From https://xkcd.com/327/

Escaping is tricky

Escaping is tricky

Web form to register for something

- First name
- Family name
- etc.

```
printf("\\begin{description}\n");
printf("\\item[First name]: %s\n", first_name);
printf("\\item[Last name]: %s\n", last_name);
printf("\\end{description}\n");
```

Which characters have some meaning in TeX??

`#$%&\^_{}~`

Also `@` if **`\makeatletter`** is used

Escaping is tricky

Web form to register for something

- First name
- Family name
- etc.

And then used on a webforum

- Avoid html tags etc.

Text is tricky

Text is tricky

```
u LATIN SMALL LETTER U
◌̀ COMBINING GRAVE ACCENT
◌́ COMBINING ACUTE ACCENT
◌̂ COMBINING CIRCUMFLEX ACCENT
◌̃ COMBINING TILDE
◌̄ COMBINING MACRON
◌̅ COMBINING OVERLINE
◌̆ COMBINING BREVE
◌̇ COMBINING DOT ABOVE
◌̈ COMBINING DIAERESIS
◌̉ COMBINING HOOK ABOVE
◌̊ COMBINING RING ABOVE
```

Text is tricky

Unicode has

- Plain letters: A
- Ligatures: ﬁ -> fi
- Characters with double-spacing: ideograms
- Characters with no space:
  - Zero-Width Space (U+200B)
  - Writing direction: Left-to-Right, Right-to-Left
- Combining characters: e + U+0301 → é

https://www.youtube.com/watch?v=jC4NNUYIIdM

Text is tricky

Fortunately Unicode does *not* have

- An embedded interpreter
- Yes, it got proposed...

Implementation-defined /
Undefined /
Unspecified
behaviors

Odd behaviors

From C99:

Implementation-defined behavior

- unspecified behavior where each implementation documents how the choice is made

Undefined behavior

- behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements

Unspecified behavior

- use of an unspecified value, or other behavior where this International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance.

Odd behaviors

Put another way:

Implementation-defined behavior

- can be anything, but at least sane, and stable.

Undefined behavior

- can be anything. Including insane such as eating your disk.

Unspecified behavior

- can be anything among a few specified sane things, and stable.

Odd behaviors

```
int x = -1;
int y = x >> 2;
```

implementation-defined

```
int x = 32;
int y = 1 << x;
```

undefined

```
int x = 0x7fffffff;
x++;
```

undefined

```
f(printf("a"),
  printf("b"));
```

unspecified

```
printf("%d %d\n",
       x++, x++);
```

undefined

“On x86 the add instruction will be used for signed add. This has two’s complement behavior on overflow. I’ll thus get two’s complement on `int`”

“Somebody told me that in basketball you can’t hold the ball and run. But I tried it and it worked. He doesn’t actually know basketball...”

In 2010, more than half of the SPECINT2006 benchmarks had some integer undefined behavior.

Why such odd behaviors?

Performance

Gives freedom to compiler to optimize code

```
int i;
for (i = 1; i <= n; i++) { ... }
```

will iterate exactly n times.

And **`n`** equal to **`INT_MAX`** is *not* supposed to happen

Why such odd behaviors?

Compilers tend to use this more and more

- Beware of that!!

e.g. memcpy vs memmove

- Optimization of memcpy in glibc broke bogus usage
- glibc had to keep a compatibility symbol...

-fsanitize=undefined

- We'll discuss more of this in the coming weeks

Do other languages do better?

They try to, e.g. Rust

- But still bugs: “I-unsound” bugs
- Poses optimization concerns
- Or runtime-checks that add overhead