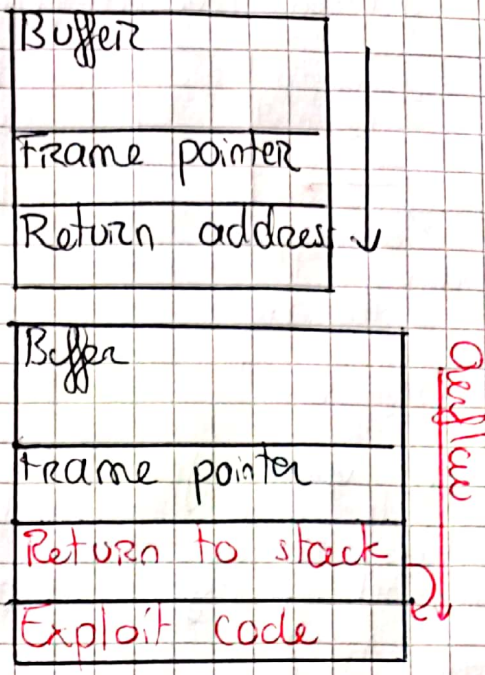
Software development security 25/02 01

Buffer overflow

What's this

```
void f(){
   char buffer[70];
   ...
   gets(buffer); // problematic line
}
```



Mitigations

Non executable stack depends on the os

Randomization
- program address space
- stack address
- heap allocation

Canaries
Depends on the compiler
- Function prolog inserts random data on the stack
- Function epilog checks the data didn't change

```
char make_filename(const char *dir, const char *file)
{
    char *r= emalloc(strlen(dir) + strlen(file) + 1);
    strcpy(r, dir);
    strcat(r, "/");
    strcat(r, file);
    return r;
}
```

it's just a wrapper that errors out in case no memory is left, looks like:

```
void *
emalloc(size_t sz)
{
  void *p= malloc(sz);
  if (!p)
    err(1, "malloc of size %zu", sz);
}
```

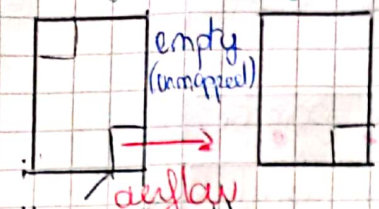
But it's not on the stack

- if you use linked lists, you killed the next pointer or the next size
- if you have power-of-two allocators, you killed the next allocation

Buffer that can overflow ------> bad

isNotRoot

Mitigation: guard pages



allocate at the end of the page

However, when first used, created insane amount of bugs
If a string was allocated at the beginning of a page, it segfaults

Smart ass?

bugs and you

- simple code should look simple
- Make things explicit
- depend on your compiler

Better APIs

- don't use strcpy, strcat
- don't use strncpy, strncat
- prefer strlcpy, strlcat

Example

```
char *dir, *file, pname[PATH_MAX]
...
if (strlcpy(pname, dir, sizeof(pname)) >= sizeof(pname))
    goto toolong;
if (strlcat(pname, file, sizeof(pname)) >= sizeof(pname))
    goto toolong;
```

The Dropper fallacy

- "But I don't write wrong code"
- The reason for slow adaptation of strlcpy

Better APIs 2

- prefer snprintf to sprintf
- use asprintf if you must

Size everywhere

- if you want to help auditors
- if size isn't obvious, make it part of the API

Sturgeon's law

90% of all software is:
- crap
- unimportant to optimize
- bogus
- copied-and-paste
- imperfect

The Dripper fallacy 2

- You can't fix everything
- ... therefore don't fix anything
- "low-hanging fruit"

Compilers help

```
#define MAX_BUF 512

char *
make_filename(const char *file, const char *dir)
{
    char buffer[MAX_BUF];
    snprintf(buffer, sizeof buffer, "%s/%s", file, dir);
    return buffer;
}
```

TOO MANY WARNING

In detail

```
function ask_user_info($user)
{
    $db->do("select * from users where user='".$user."'");
}
```

What you assume

user = Robin
select * from users where user='Robin'

What the attacker may do

user = Robin'; drop all tables

More subtle

user = Robin' or 1==1;--
select * from user where user=='Robin' or 1==1;--'

But why???

- because it's not taught in all database courses
- php use do()

Bad solutions

- sanitize (quote) the argument
- user should be okay if there's no ' in it
- (or should it)
- what about Mr. O'Brian?
- ya got to quote everything
- ... there are several things wrong with this!

What to do

Use Prepared statement
// the better way
function ask_user_info ($user)
{
  $stmt = $db->prepare("select * from users where user=?");
  $stmt->bind_param("s", $user);
  stmt->execute();
}

Or switch to an actual ORM such as symfony in php

?: placeholder → works in every database

Exemple: 2 windows app could not be launched at the same time (docker and razer)
Docker: should be only one instance → c/c code from internet is really buggy
→ ⊗ use the docker class, used the framework class
is razer use the same framework ⇒ bug

<Insert a quote here> - Anonymous

More about quoting

- What you don't know WILL kill you!
- Never do matching against negative patterns
- e.g. an email address is NOT something that does not contain some characters
- it IS something that only matches a given pattern
- (subsidiary question: figure out a Regexp that matches email)

```
void
print_msg (const char *msg)
{
  printf("There is a problem here:\n");
  printf(msg); → problem
}
```

It will compile but with many warnings

What can go wrong?

- Can dump the stack → since printf is a variadic function
- It is possible to write in memory w/ printf → buffer overflow
- This might take stuff from the stack and show you stuff you should not know
- It is actually WAY WORSE

⇒ can happen if string is formated the wrong way
is will count as parameters