



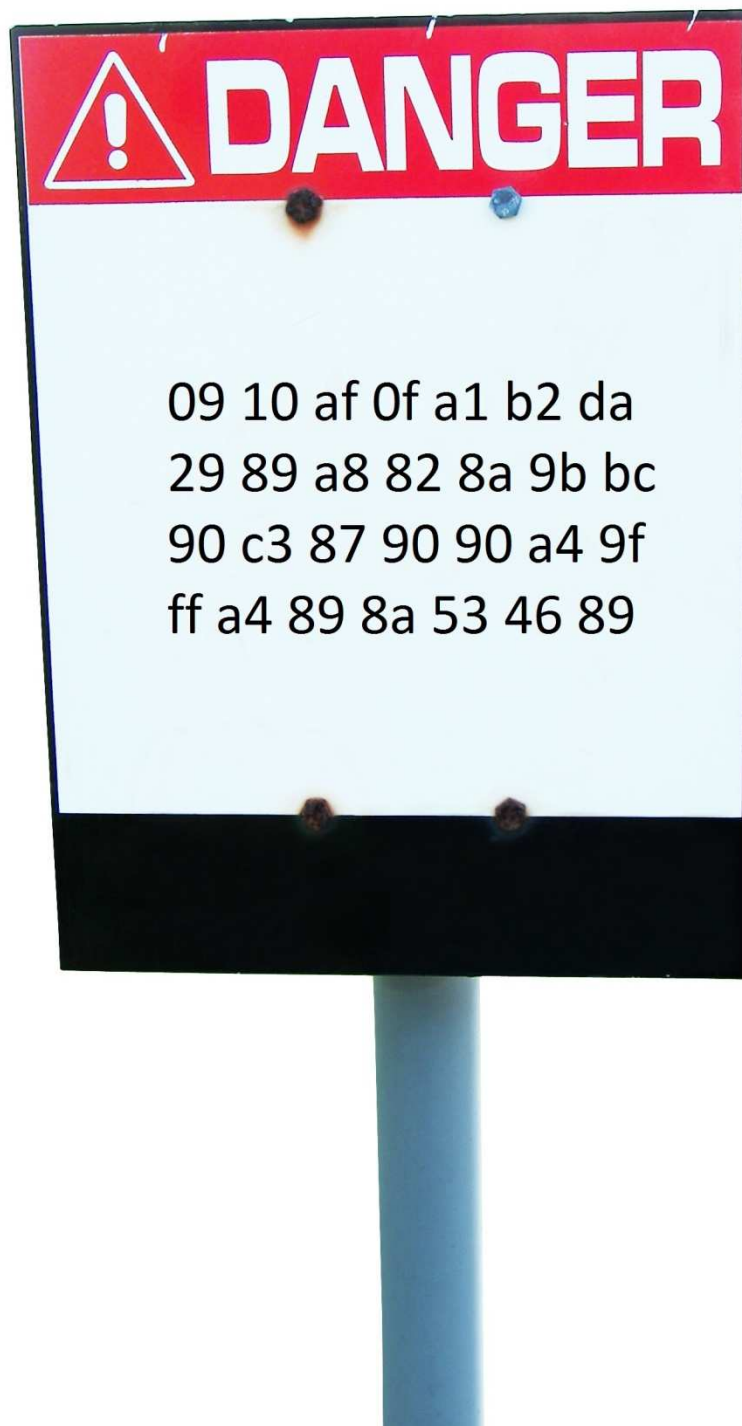
EPOCHE & ESPRI



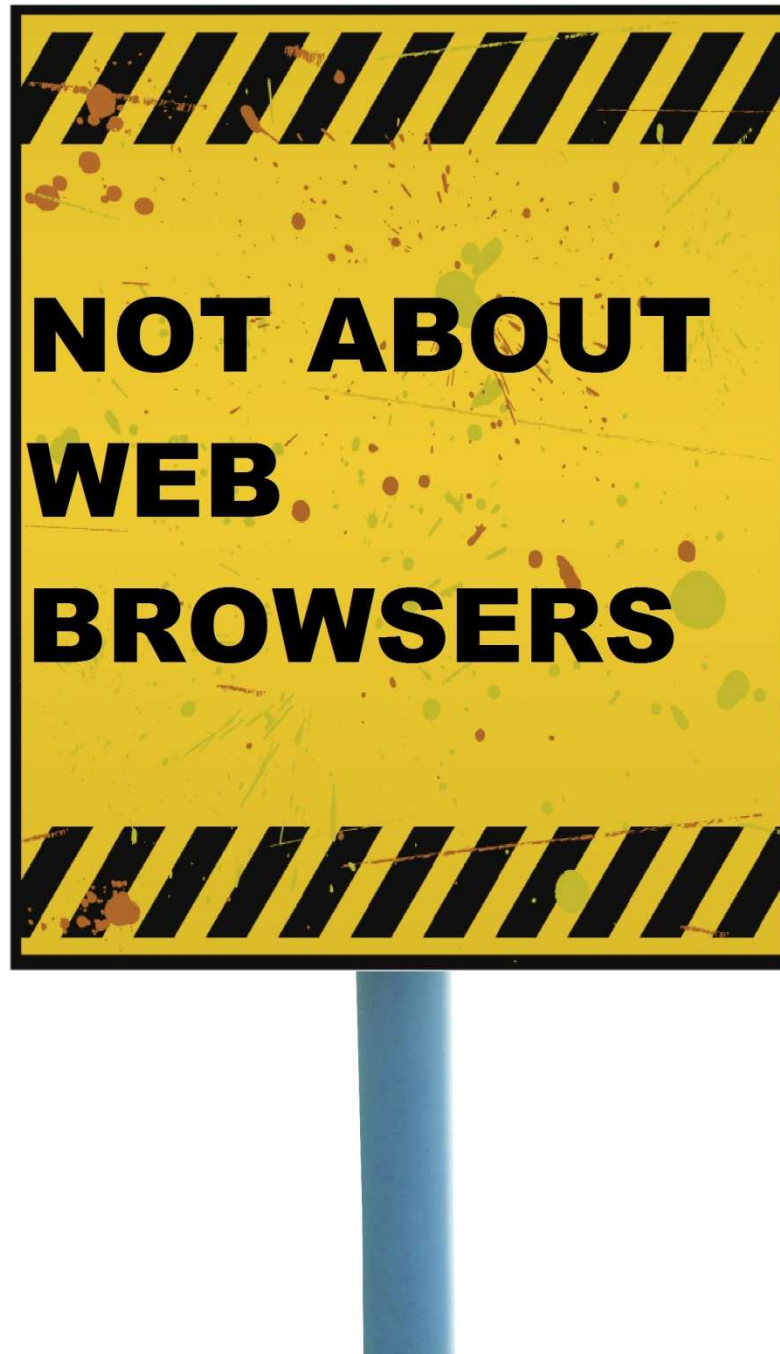
Overflowing Attack Potential

Scoring Defence-in-Depth

Javier Tallón Guerri
11ICCC - Turkey







- 1. Buffer overflows, a bit of background**
- 2. Reviewing and bypassing defence-in-depth techniques**
- 3. Impact in the CC**
- 4. What to do?**



E P O C H E & E S P R I

1. Buffer overflows, a bit of background



2. Reviewing and bypassing defence-in-depth techniques

3. Impact in the CC

4. What to do?

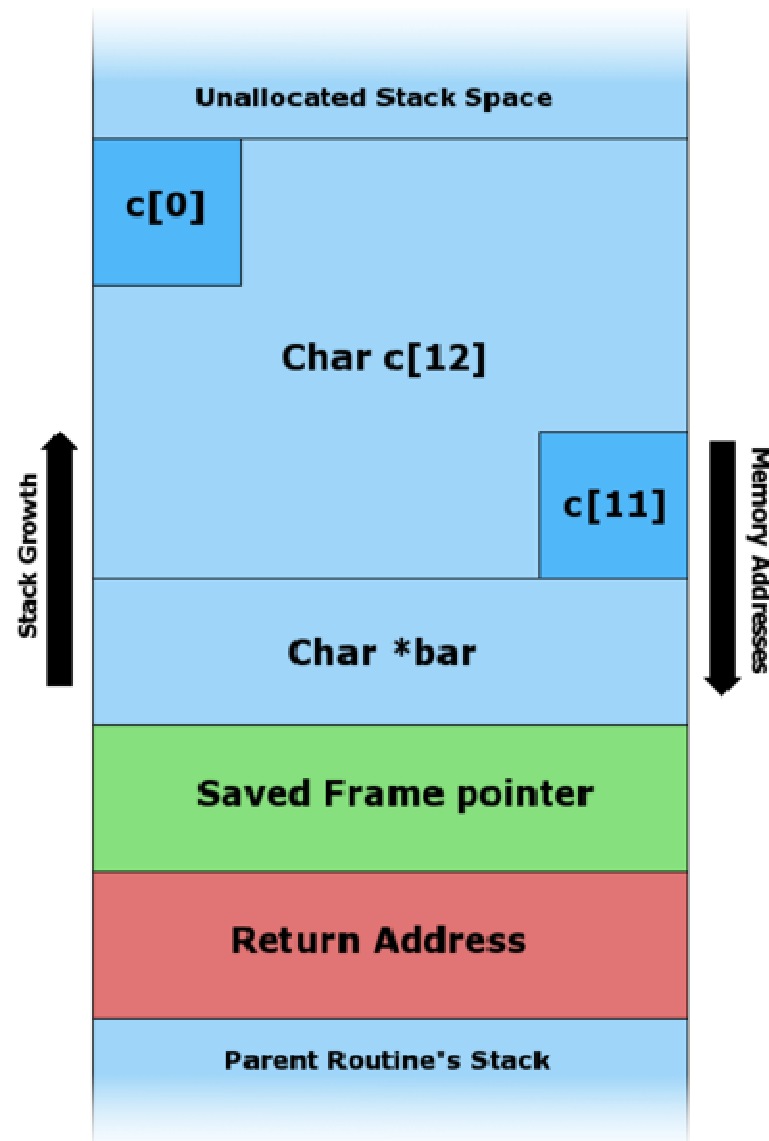
1. Buffer overflows, a bit of background

- You know... The classic stack overflow....

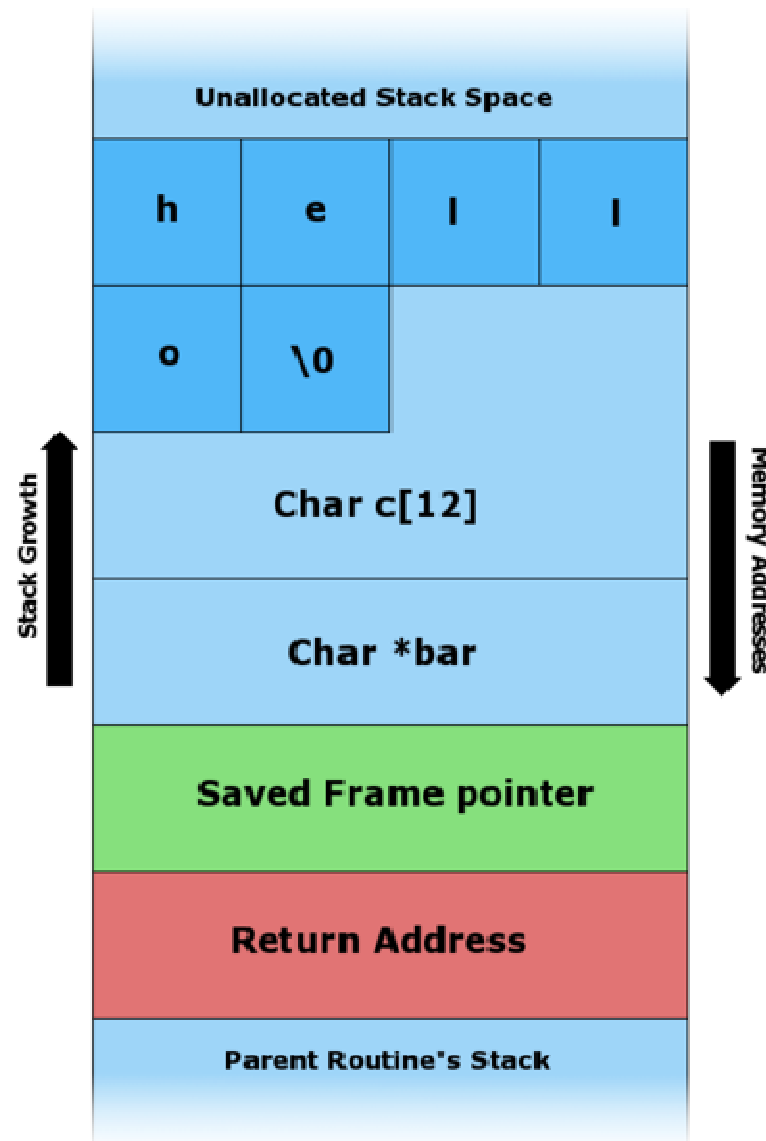
```
#include <string.h>
void foo (char *bar) {
    char c[12];
    strcpy(c, bar); // no bounds checking...
}

int main (int argc, char **argv) {
    foo(argv[1]);
}
```

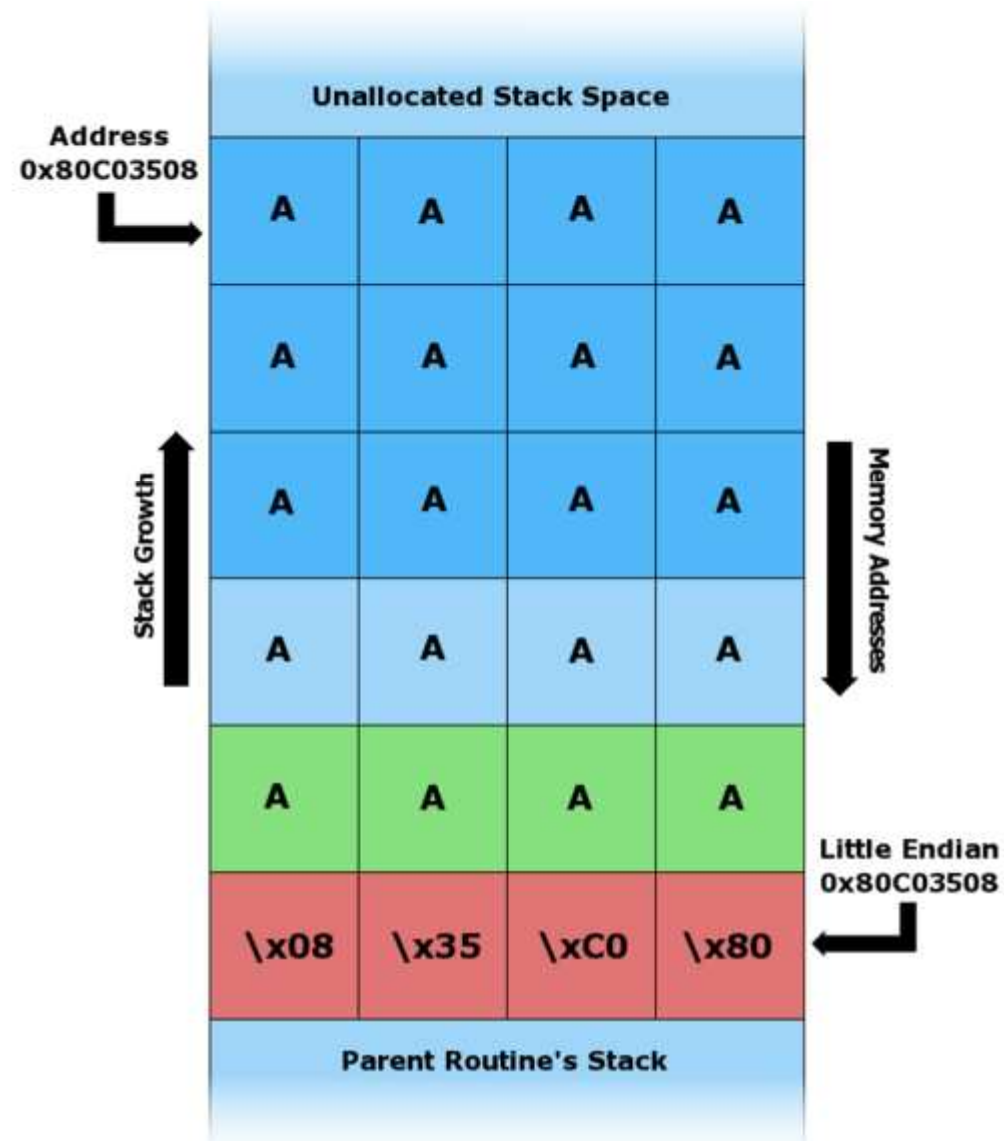
1. Buffer overflows, a bit of background



1. Buffer overflows, a bit of background



1. Buffer overflows, a bit of background



1. Buffer overflows, a bit of background

- There are also heap and integer overflows....

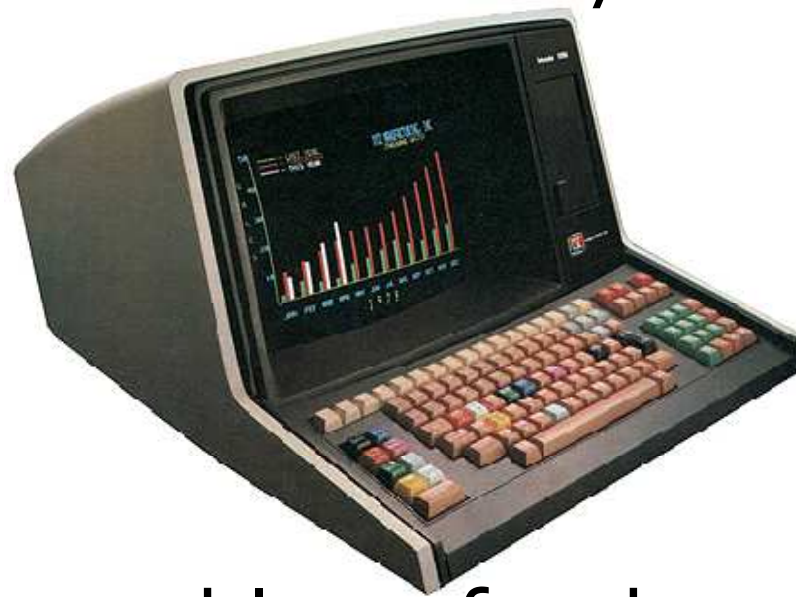
1. Buffer overflows, a bit of background

- Could lead to arbitrary code execution



1. Buffer overflows, a bit of background

- Those were the old days...



- Very few problems for the attacker:
 - Null bytes
 - Shellcode size and other constraints
 - Shellcode development



EPOCHÉ & ESPRI

1. Buffer overflows, a bit of background

2. Reviewing defence-in-depth techniques



3. Impact in the CC

4. What to do?

2. Reviewing defence-in-depth techniques

- Stack canaries approach:
 - The compiler place a value before the return address when a function is called and check that the value has not changed when the function finalize.



2. Reviewing defence-in-depth techniques

- Bypassing stack canaries:
 - Implementation can be not correct
 - It can be a statistical problem



2. Reviewing defence-in-depth techniques

- Bypassing stack canaries:
 - Windows: SEH overwriting
 - Protected by SafeSEH and so on...
 - Unix: Other (more complex) techniques...

2. Reviewing defence-in-depth techniques

- Non-eXecutable Stack approach:
 - Effective implementation in hardware. Widely deployed (every computer since 2001 allow this)
 - Code is code and data is data
 - However, it is easy to bypass

2. Reviewing defence-in-depth techniques

- Bypassing Non-eXecutable Stack:
 - Save the payload in the heap
 - Return into libc (standard C library) attacks

2. Reviewing defence-in-depth techniques

- 64bits hardware saves the way:
 - In 64 bits personal computers, arguments are loaded in registers, not in the stack.
 - Return into libc attack is not possible



2. Reviewing defence-in-depth techniques

■ ~~64 bits hardware saves the way:~~



- Return oriented programming

2. Reviewing defence-in-depth techniques

RETURN ORiented PROGRAMming
IS LIke a rAnsom nOTE, BUT
inStead Of cUttinG CUt LEtters
frOm maGAZines yOU Are
CUttinG Out iNStRUCtiONS FrOM
tEXt SEGmEnts

2. Reviewing defence-in-depth techniques

- ASLR (Address Space Layout Randomization) approach:
 - The code is loaded in different memory regions each time

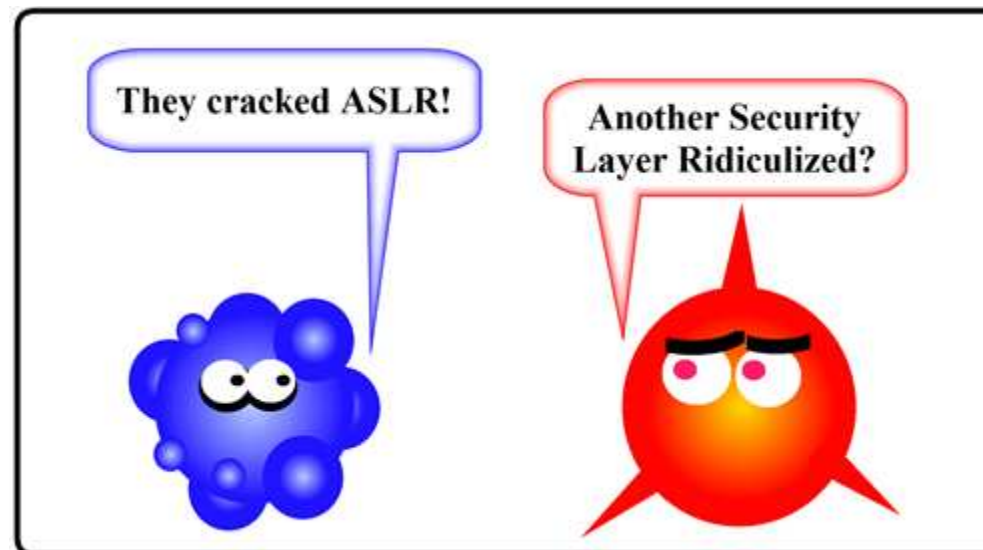
2. Reviewing defence-in-depth techniques

- Bypassing ASLR:
 - It could be an statistical question



2. Reviewing defence-in-depth techniques

- Bypassing ASLR:
 - Maybe not all the libraries are randomly loaded



2. Reviewing defence-in-depth techniques

- Mixed approach:
 - Standalone use of this techniques is not very useful



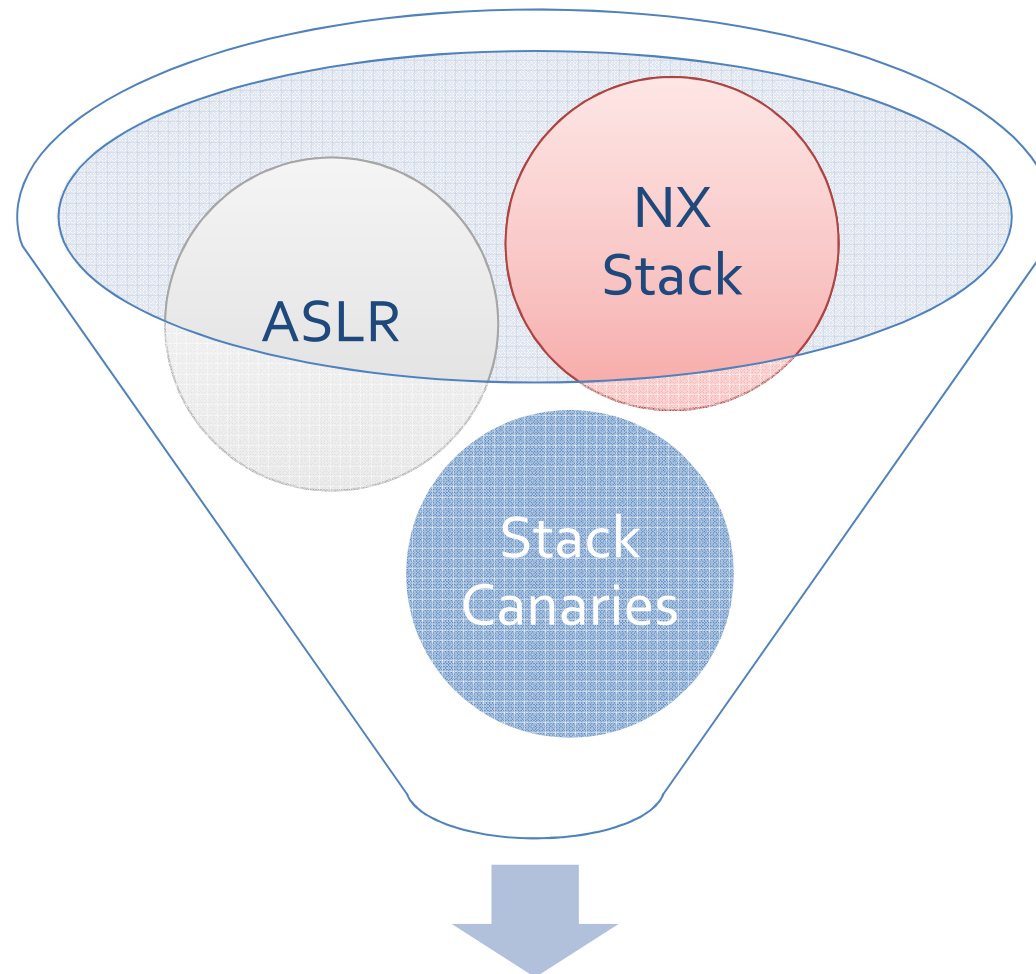


OCHE & ESPRI

2. Reviewing defence-in-depth techniques

- Non-eXecutable Stack + ASLR:
 - Make very difficult the return attacks.

2. Reviewing defence-in-depth techniques

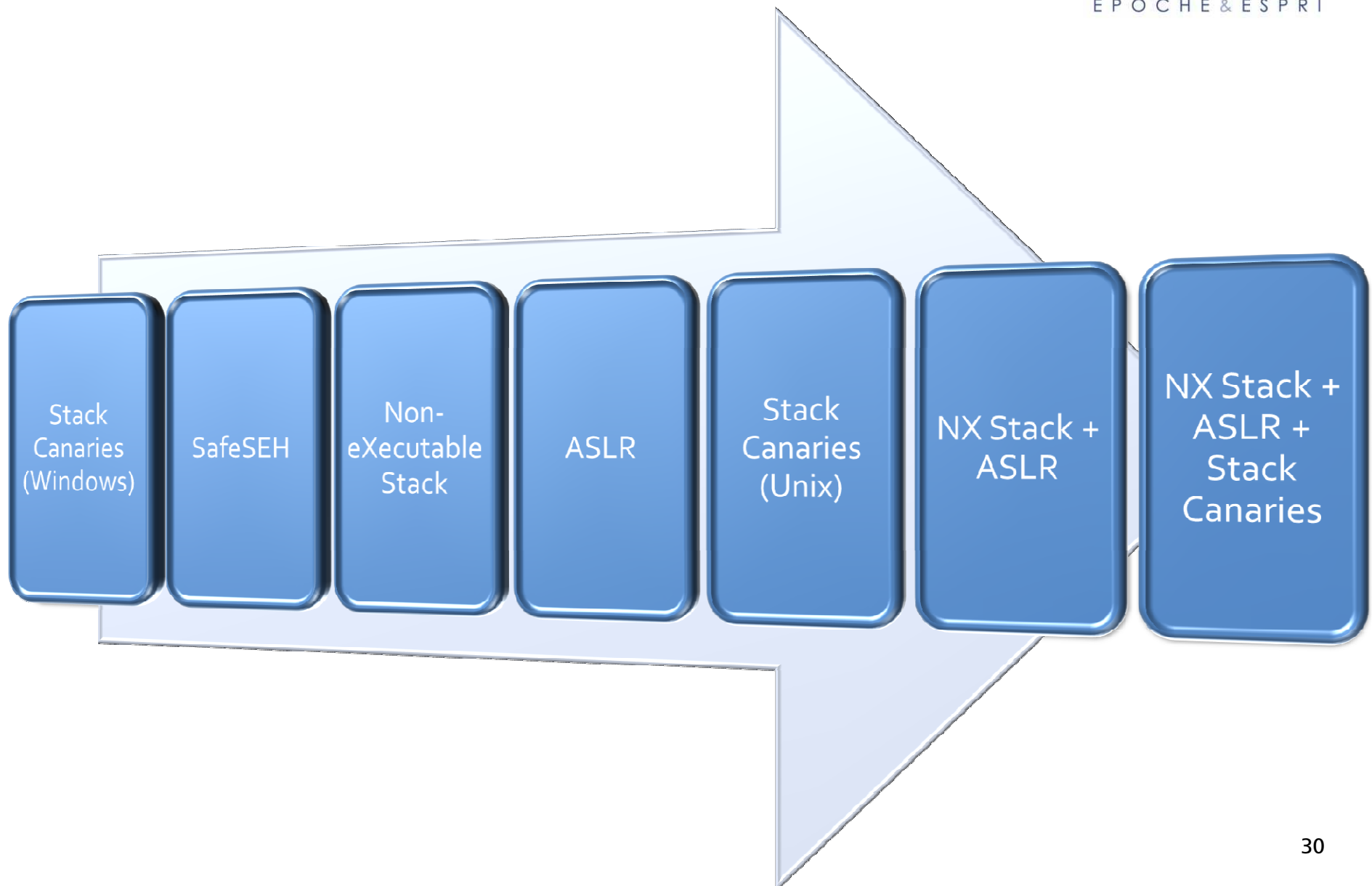


Each technique makes successful exploitation more difficult

2. Reviewing defence-in-depth techniques


- There exists more defence-in-depth techniques
- Attackers also develop new techniques to bypass the countermeasures

2. Reviewing defence-in-depth techniques



2. Reviewing defence-in-depth techniques



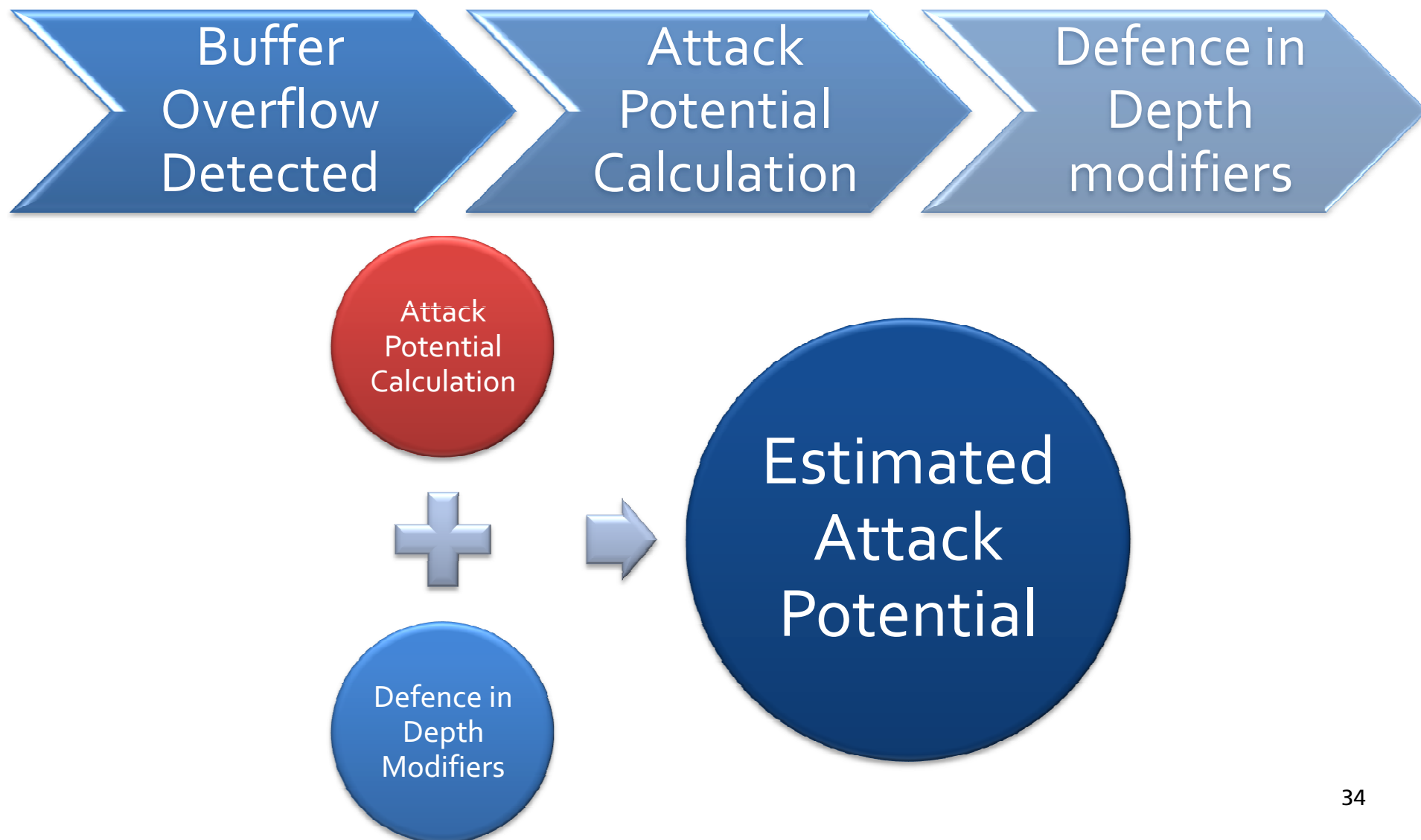
1. Buffer overflows, a bit of background
2. Reviewing defence-in-depth techniques
3. Impact in the CC 
4. What to do?

3. Impact in the CC

- We start from a detected buffer overflow
 - Unique characteristics
 - Unique exploit path


- Attack potential calculation

3. Impact in the CC



3. Impact in the CC

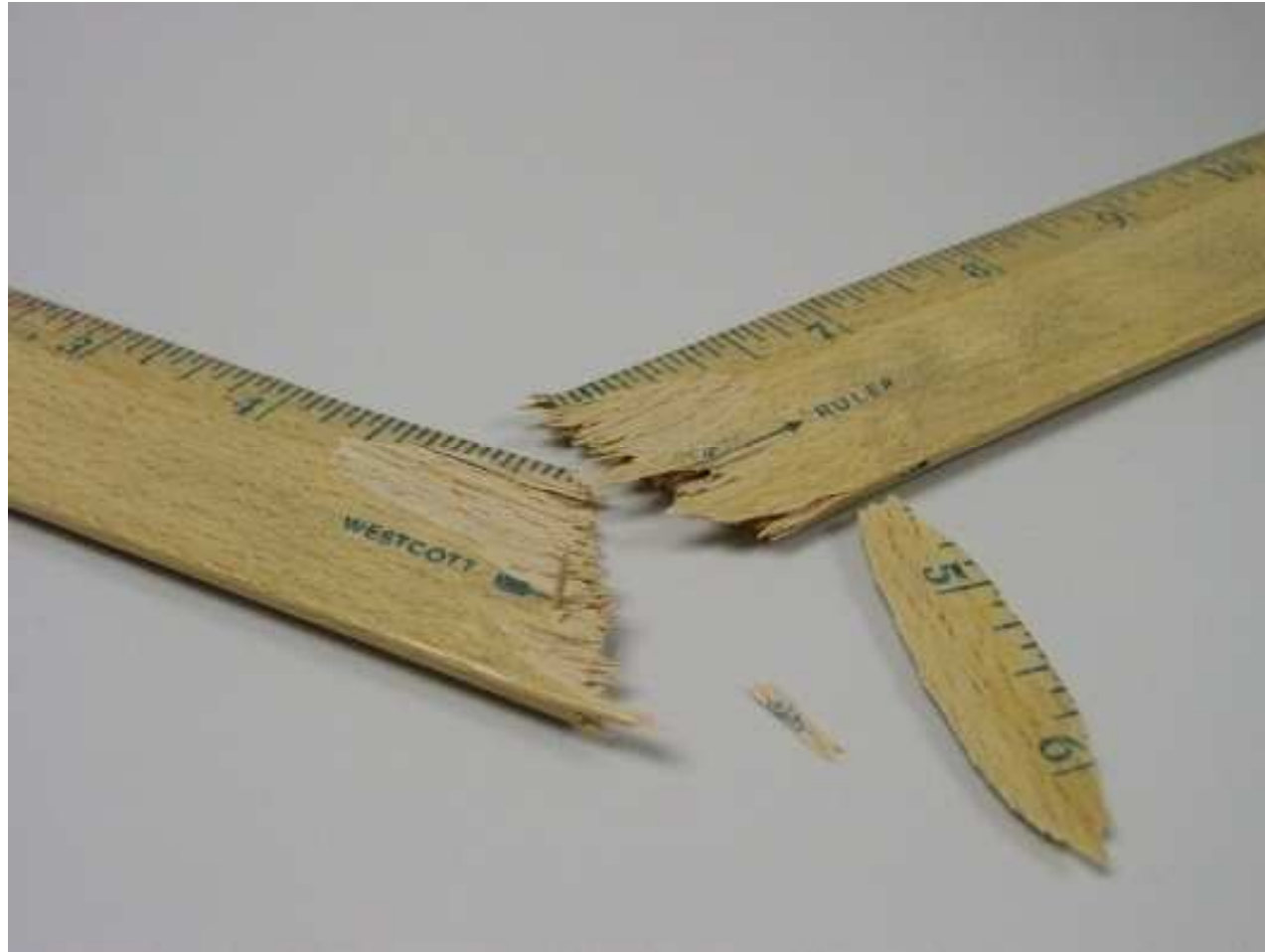
Defence in depth technique	Attack potential factor
Stack Canaries (Windows)	x 1.2
SafeSEH	x 1.3
Non-eXecutable Stack	x 1.35
ASLR	x 1.50
Stack Canaries (Unix)	x 1.52
NX Stack + ASLR	x 1.54
NX Stack + ASLR + Stack Canaries (Windows)	x 1.62
NX Stack + ASLR + Stack Canaries (Windows) + SafeSEH	x 1.66
NX Stack + ASLR + Stack Canaries (Linux)	x 1.68
...	...

1. Buffer overflows, a bit of background
2. Reviewing defence-in-depth techniques
3. Impact in the CC
4. What to do? 

4. What to do?



EPOCHE & ESPRI



THERE IS NO EXACT RULE³

4. What to do?



EPOCHE & ESPRI

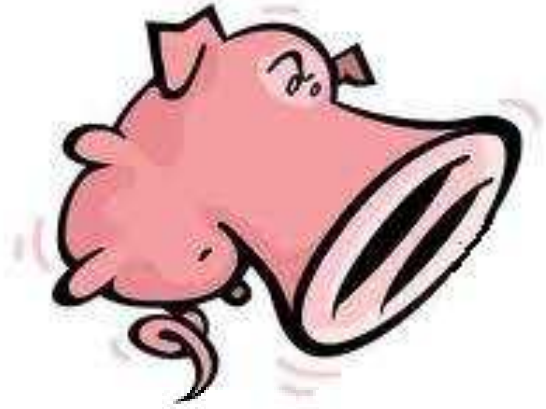


TIME CHANGES THE THINGS

4. What to do?



EPOCHE & ESPRI



OTHER AREAS OF APPLICATION?

4. What to do?

Apply those techniques!

- Whenever it is possible
- Through compiler
- Through Operating System

4. What to do?





E P O C H E & E S P R I

Thanks for your attention!

Javier Tallón

Epoche & Espri, S.L.
Avda. de la Vega, 1
28108, Alcobendas,
Madrid, Spain.

eval@epoche.es