Introduction to Computer Security Lecture Slides

Application Security

Asst. Prof. Mihai Chiroiu

- “My software never has bugs. It just develops random features.”

Contents

- Computer Vulnerabilities
  - Cause & classification
  - Memory safety
  - Common mitigations
- State of the Art
  - Eternal War in Memory (paper presentation)

Software – the final frontier

- Access control and crypto are the bricks for building blocks
- Protocols/algorithms used to design usefull blocks
- Software implements all of the above

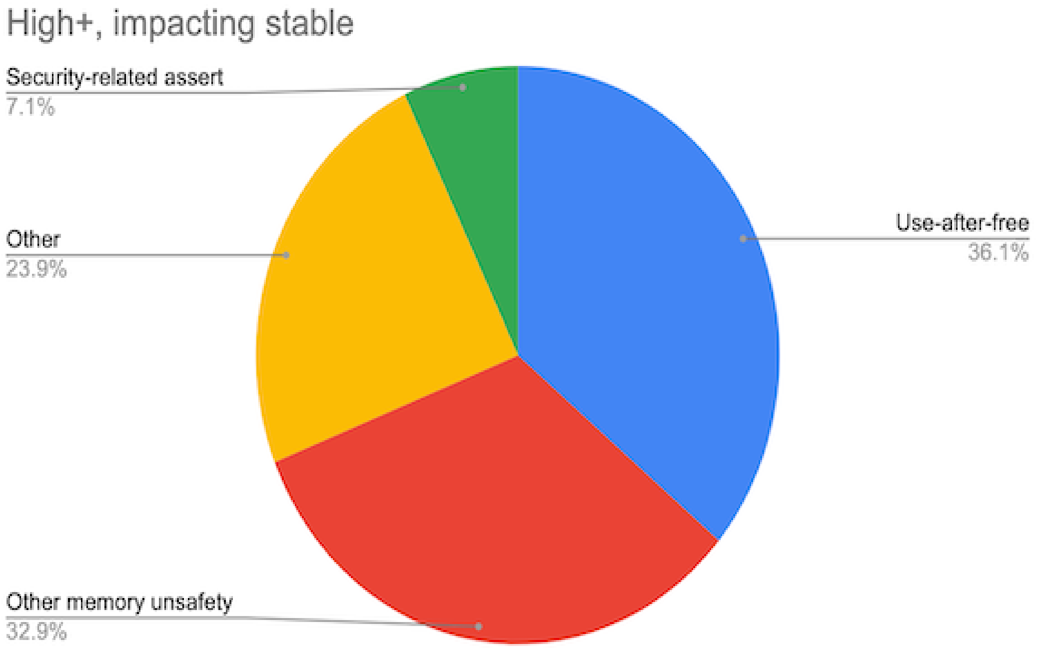
Software vulnerabilities

- **Memory safety**: buffer overflow, dangling pointer, race condition, memory leak, free after use etc.
- Input validation: code injection, format string attacks, path traversal…
- Side channel attacks
- UI confusion
- Privilege escalation
- And many more!

Software vulnerabilities (2)

- **Chrome: 70% of all security bugs are memory safety issues**

https://www.zdnet.com/article/chrome-70-of-all-security-bugs-are-memory-safety-issues/

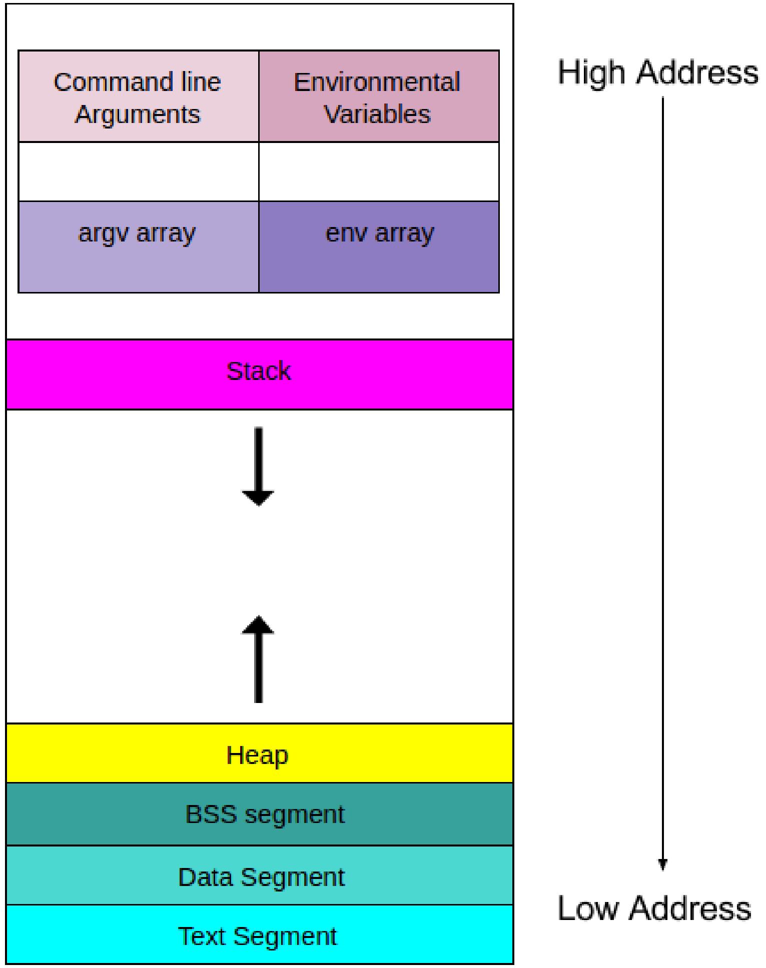


Intro: address space

- Userspace processes have virtual memory

- Compiler (linker) + OS decide where each segment goes.

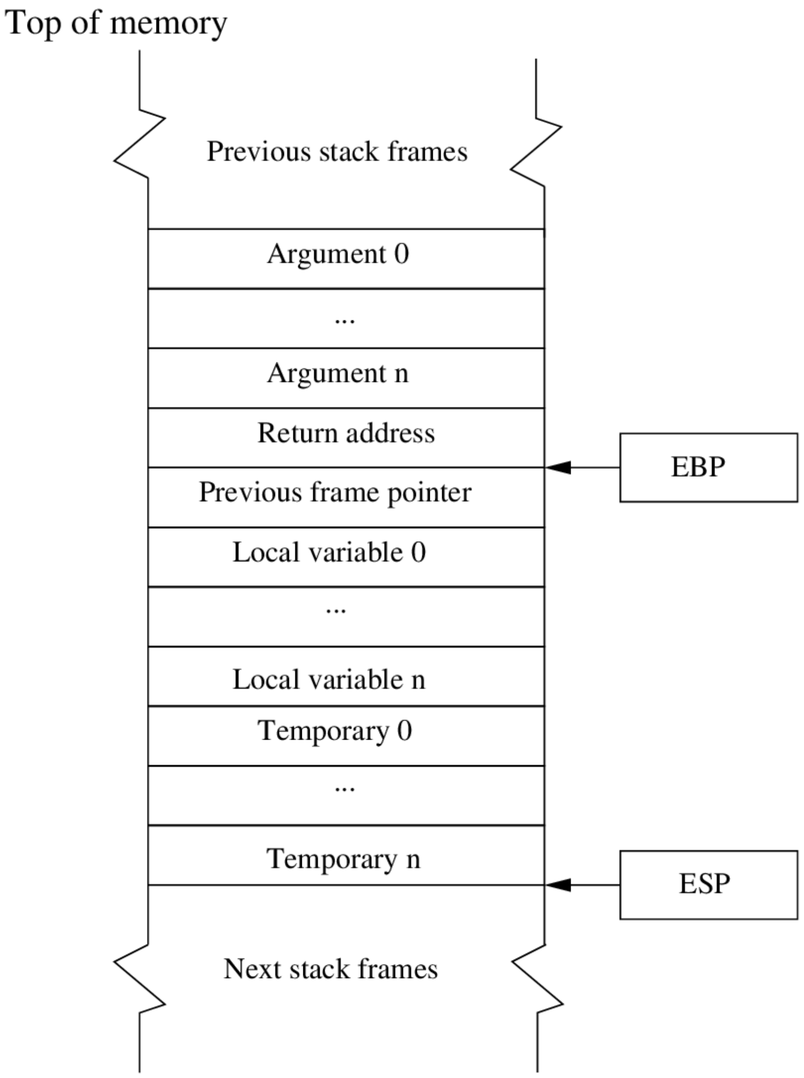
- Address space layout has impact on application's security



Intro: stack frame

- Stack grows downward (x86)

- Contains function arguments, saved CPU state (registers, instruction / frame pointers) and local variables.



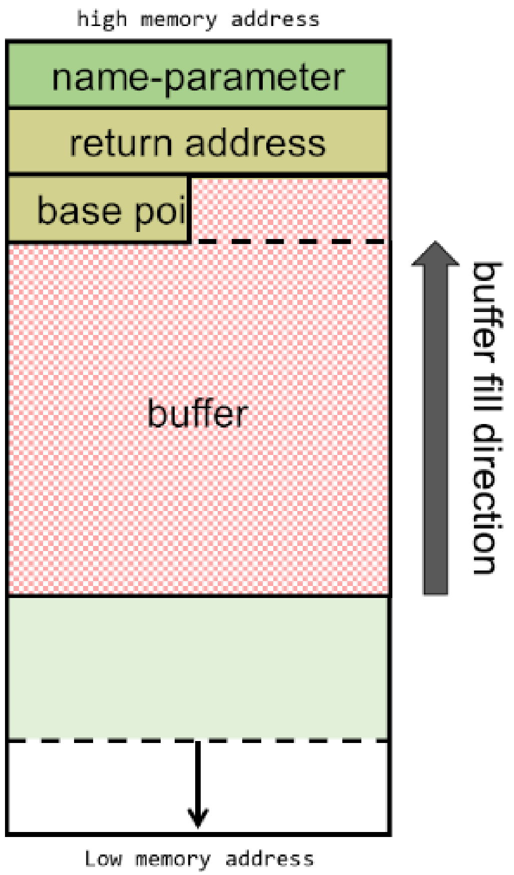
ISC security crunch

Stack buffer overflow

- Happens when a buffer's is written after its allocated size.

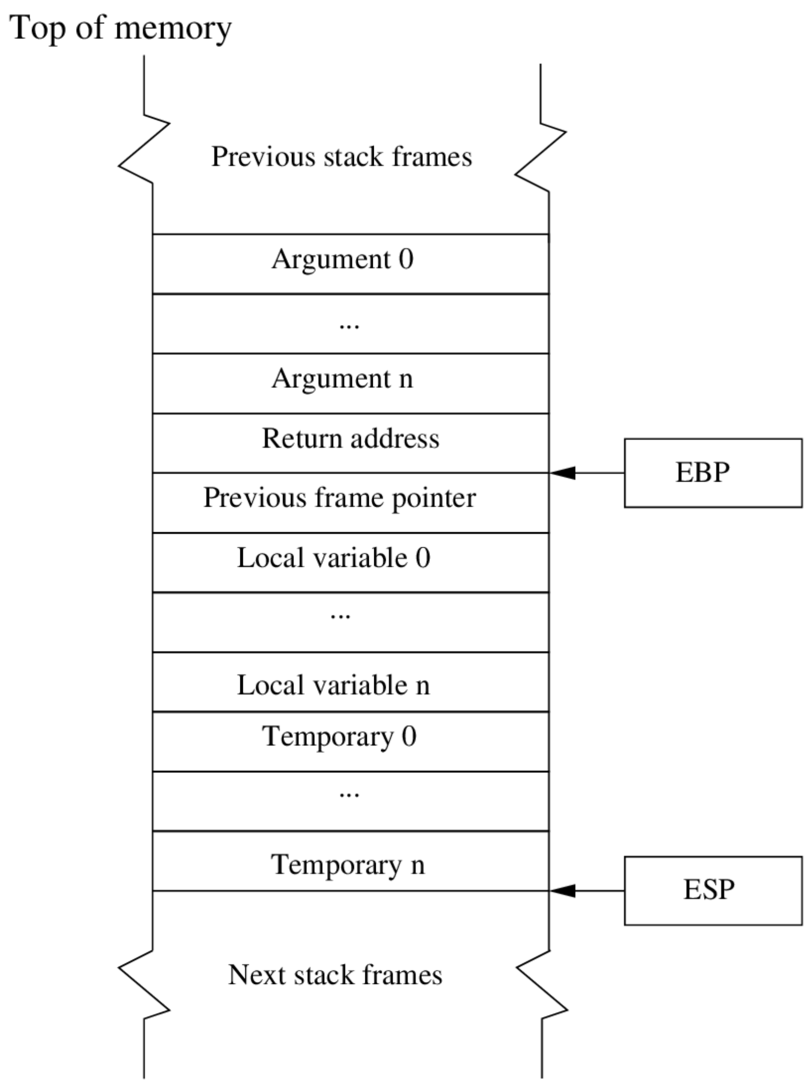
```
char buf[10];
char *input = "This text is larger than expected";

strcpy(buf, input);
```



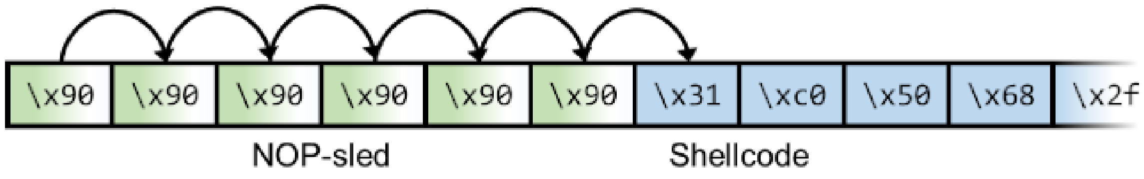
Buffer overflow (2)

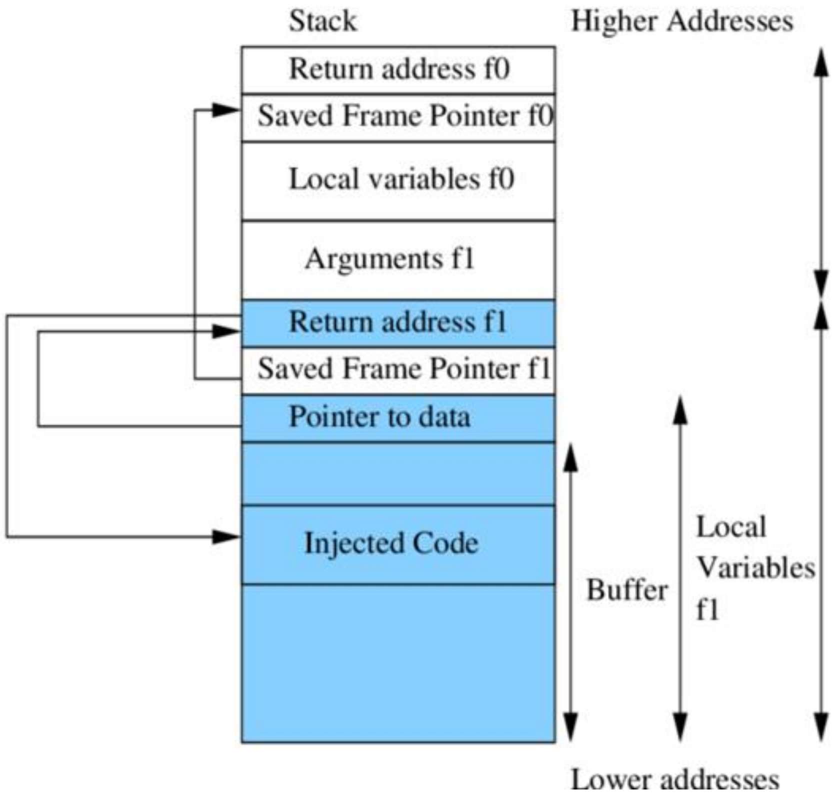
```
(gdb) disas func
Dump of assembler code for function func:
   0x0804841b <+0>:     push   %ebp
   0x0804841c <+1>:     mov    %esp,%ebp
   0x0804841e <+3>:     sub    $0x64,%esp
   0x08048421 <+6>:     pushl  0x8(%ebp)
   0x08048424 <+9>:     lea    -0x64(%ebp),%eax
   0x08048427 <+12>:    push   %eax
   0x08048428 <+13>:    call   0x80482f0 <strcpy@plt>
   0x0804842d <+18>:    add    $0x8,%esp
   0x08048430 <+21>:    lea    -0x64(%ebp),%eax
   0x08048433 <+24>:    push   %eax
   0x08048434 <+25>:    push   $0x80484e0
   0x08048439 <+30>:    call   0x80482e0 <printf@plt>
   0x0804843e <+35>:    add    $0x8,%esp
   0x08048441 <+38>:    nop
   0x08048442 <+39>:    leave
   0x08048443 <+40>:    ret
End of assembler dump.
```



Buffer overflow (3)

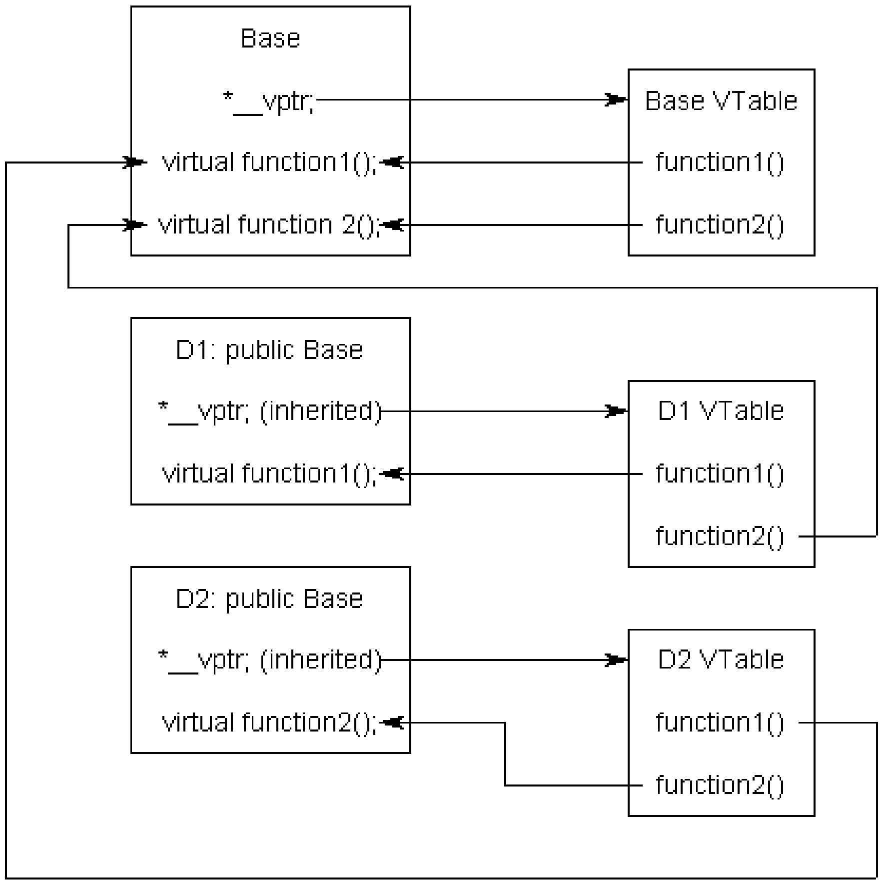
- ret instruction will pop the return address from the stack, then jump to it.
- CPU will execute the injected code (shellcode).
- NOP sled when the address is not fixed:





What about heap?

- C++ (and other OOP languages) use virtual method tables for implementing polymorphism
- Attacker replaces VTable pointers to controlled memory
- When an object method is called, the function pointer is loaded from the attacker's VTable

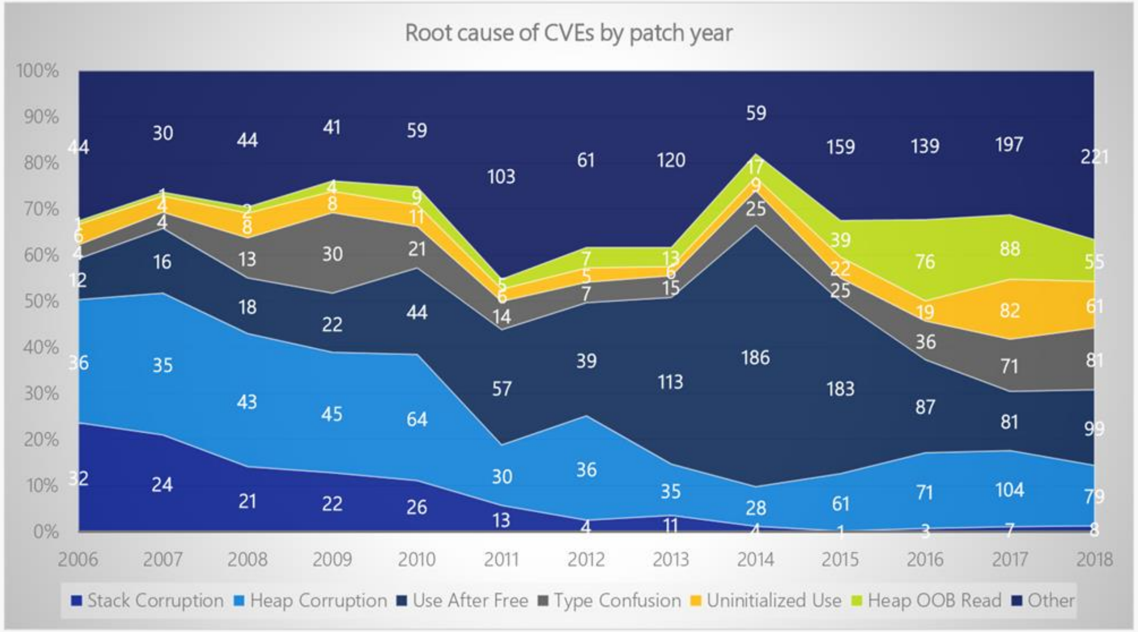


What about .data?

```
#include ...
struct app_state_t {
    int buf [20];
        void *next_item;
} ;
struct app_state_t app_state;
...
main() {
        // ... buffer overflow on app_state.buf ...
        *app_state.next_item = new_item;
}
```

Microsoft: *BlueHatIL - Trends, challenge, and shifts in software vulnerability mitigation*

Drilling down into root causes



Stack corruptions are essentially dead

Use after free spiked in 2013-2015 due to web browser UAF, but was mitigated by Mem GC

Heap out-of-bounds read, type confusion, & uninitialized use have generally increased

Spatial safety remains the most common vulnerability category (heap out-of-bounds read/write)

Top root causes since 2016: | #1: heap out-of-bounds | #2: use after free | #3: type confusion | #4: uninitialized use

11

Note: CVEs may have multiple root causes, so they can be counted in multiple categories

Real World Examples

- EternalBlue - SMB Protocol Vulnerability (CVE-2017-0144)
  https://research.checkpoint.com/2017/eternalblue-everything-know

- Microsoft Exchange RCE Vulnerability (CVE-2021-26857)
  https://www.microsoft.com/security/blog/2021/03/02/hafnium-targeting-exchange-servers/

- Flash Player (CVE-2018-15982)
  https://securityaffairs.co/wordpress/78712/hacking/cve-2018-15982-flash-zero-day.html

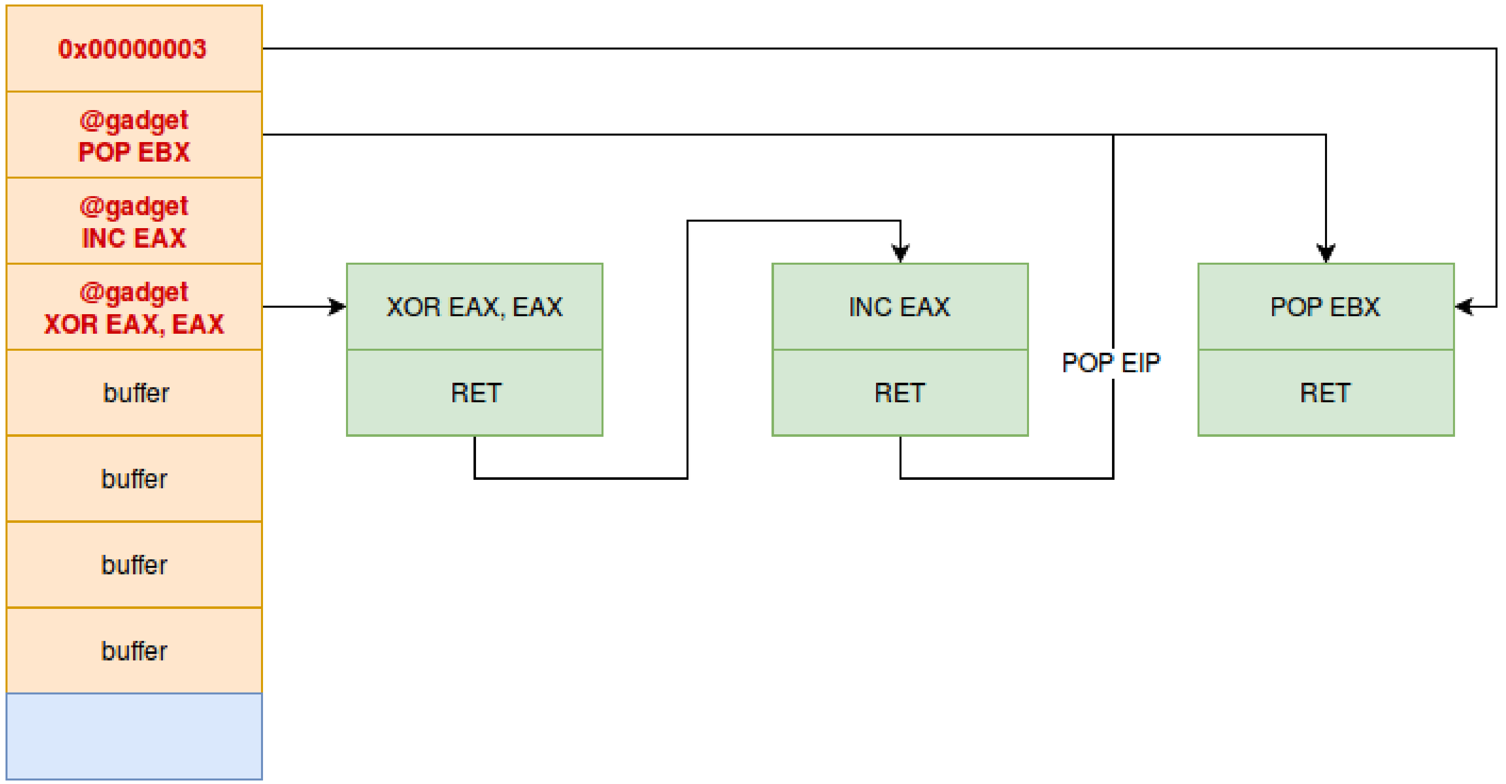
Buffer overflow mitigation

- DEP (data execution prevention) / No-execute (NX) bit
  - *defeated by ROP (return oriented programming)*

- Address space layout randomization
- Stack Canaries
  - *defeated by memory leakage, side channels etc.*

- Control flow integrity

Return oriented programming



Control Flow Integrity

```
bool lt(int x, int y) {
    return x < y;
}

bool gt(int x, int y) {
    return x > y;
}

sort2(int a[], int b[], int len)
{
    sort( a, len, lt );
    sort( b, len, gt );
}
```