ROP - Return Oriented Programming

Ou POG – Programação orientada a gambiarra

Helvio B. D. De Carvalho Junior

Malware and Security Researcher | OSCE | OSCP | eMAPT | CEHv9

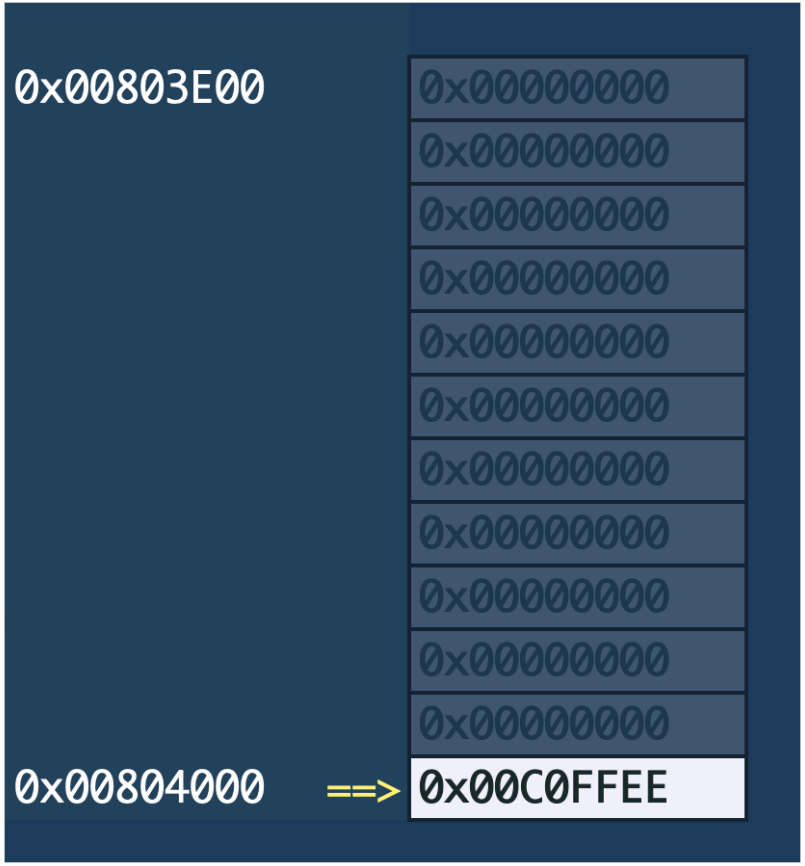
- E-mail: helvio_junior@hotmail.com
- Instagram: @helvio_m4v3r1ck
- Mais de 20 anos de atuação com TI
- Foco de estudo e pesquisa:
  - Segurança ofensiva (Red Team)
  - Criação de exploits
  - Bug hunting, Cyber threat hunting
  - Criação e engenharia reversa de Malware
- Especialista em Cyber Security no Banco Original
- https://www.helviojunior.com.br/
- https://github.com/helviojunior
- https://treinamentos.helviojunior.com.br/

Agenda

- Relembrando um Stack based Buffer Overflow
- Entendendo as instruções POP e RET
- Passagem de parâmetros para função
- Entendendo o ROP
- Explorando a vulnerabilidade

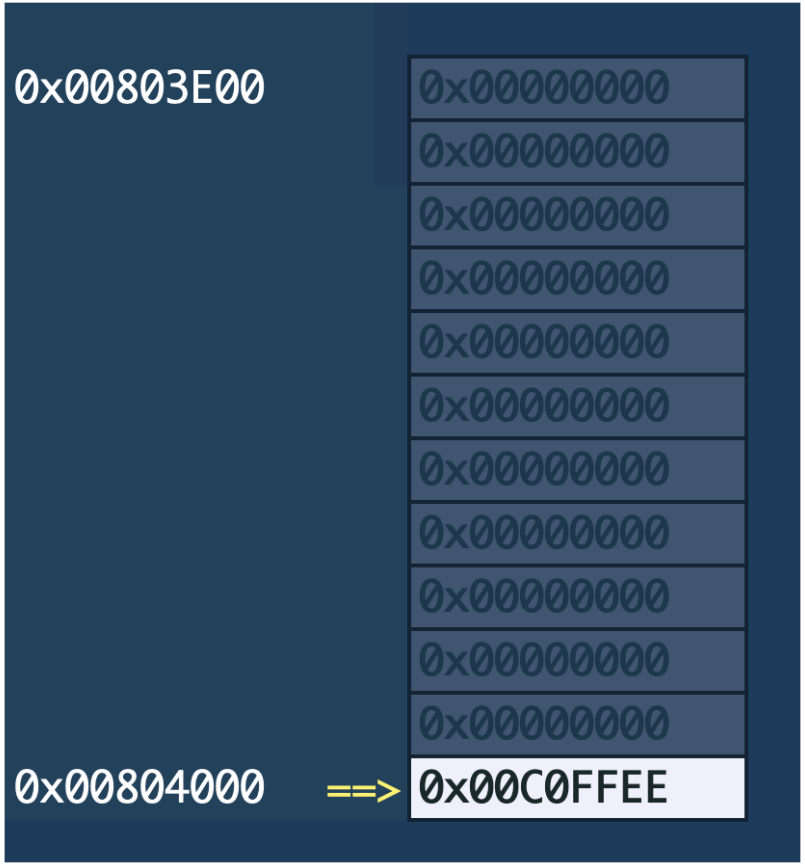
Stack Overflow

```
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
```



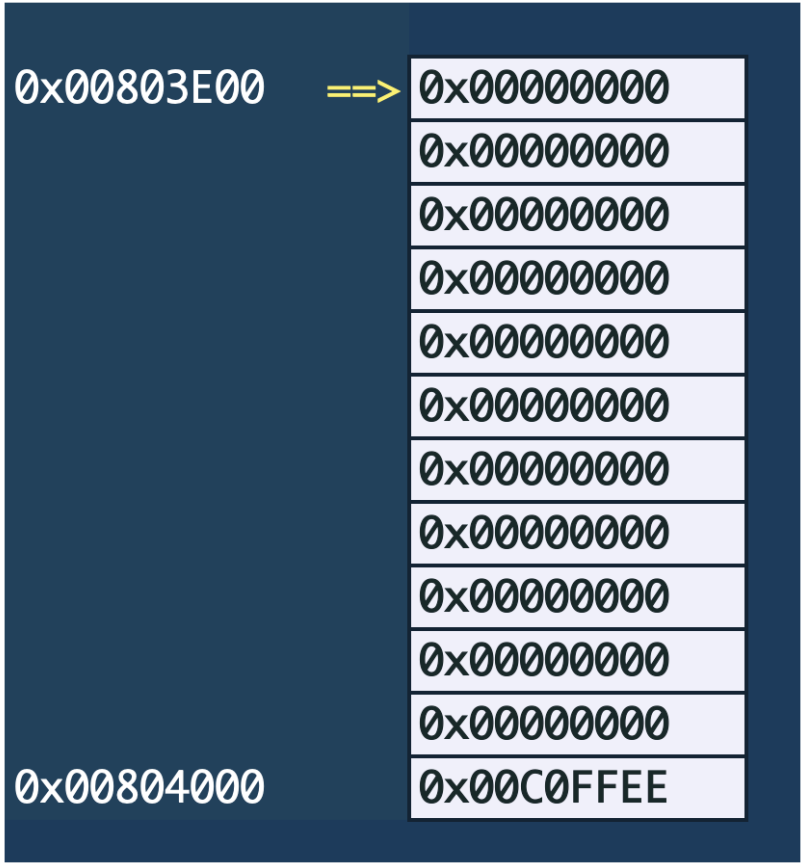
Stack Overflow

```
==> int QueryUser()
    {
        char buffer[512];
        gets(buffer);
        ...
        return 1;
    }
```



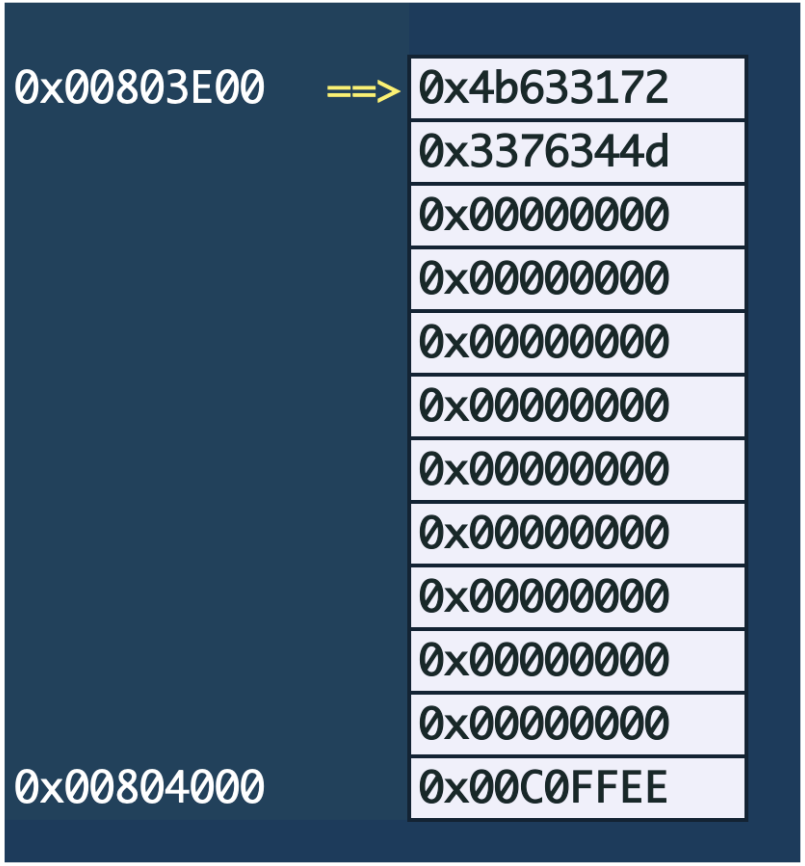
Stack Overflow

```
    int QueryUser()
    {
==>     char buffer[512];
        gets(buffer);
        ...
        return 1;
    }
```



Stack Overflow

```
int QueryUser()
{
    char buffer[512];
==> gets(buffer);
    ...
    return 1;
}
```



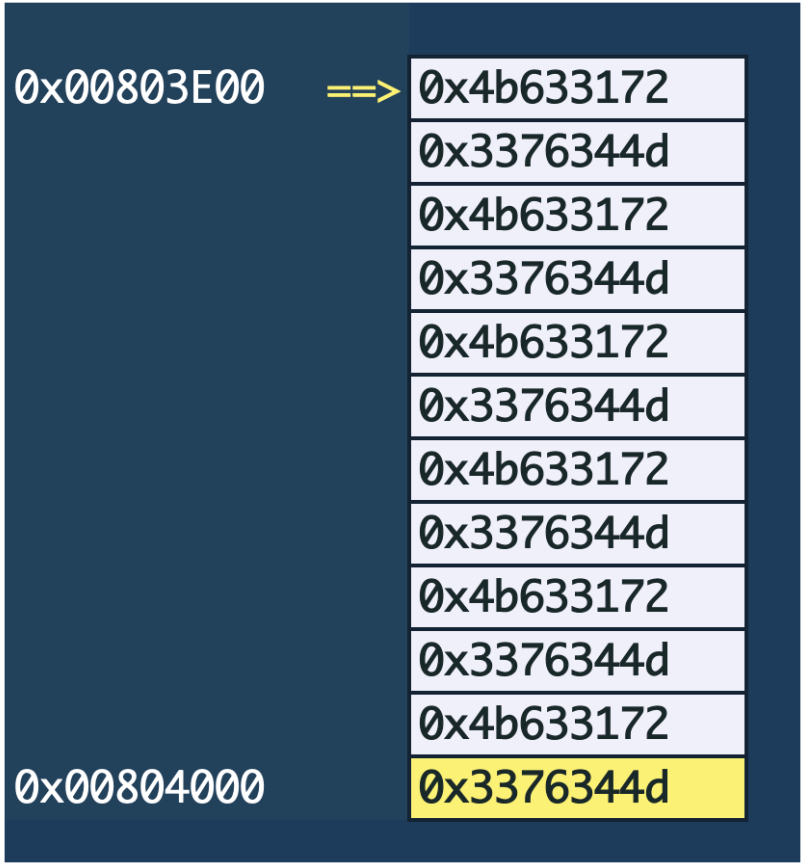
Stack Overflow

```
int QueryUser()
{
    char buffer[512];
==> gets(buffer);
    ...
    return 1;
}
```

0x00803E00	==>	0x4b633172
		0x3376344d
		0x4b633172
		0x3376344d
		0x4b633172
		0x3376344d
		0x4b633172
		0x3376344d
		0x4b633172
		0x3376344d
		0x00000000
0x00804000		0x00C0FFEE

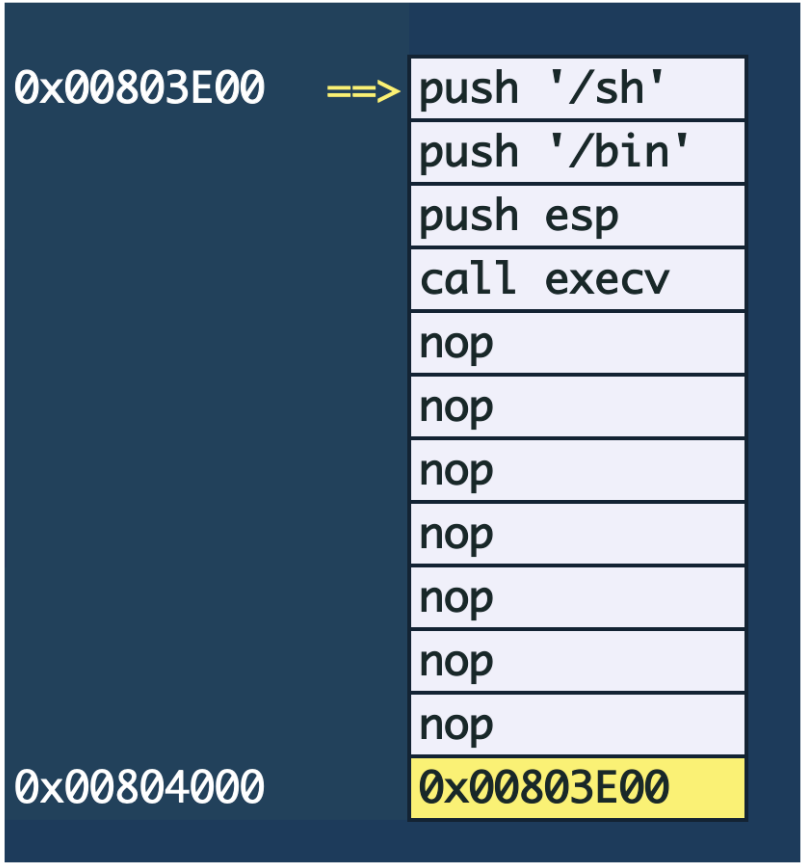
Stack Overflow

```
int QueryUser()
{
    char buffer[512];
==> gets(buffer);
    ...
    return 1;
}
```



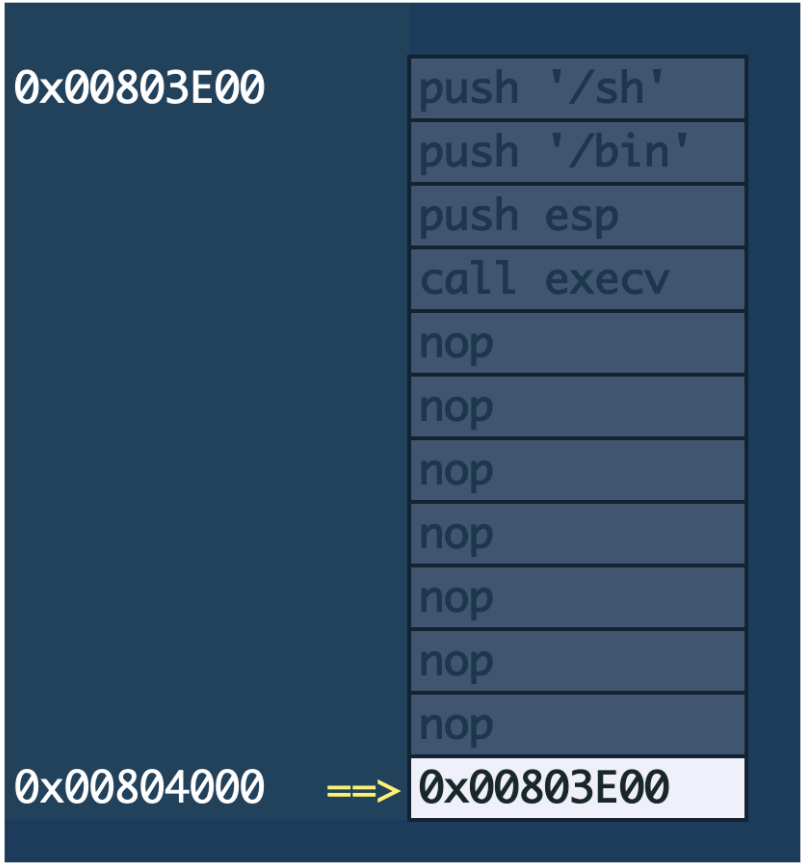
Stack Overflow

```
int QueryUser()
{
    char buffer[512];
==> gets(buffer);
    ...
    return 1;
}
```



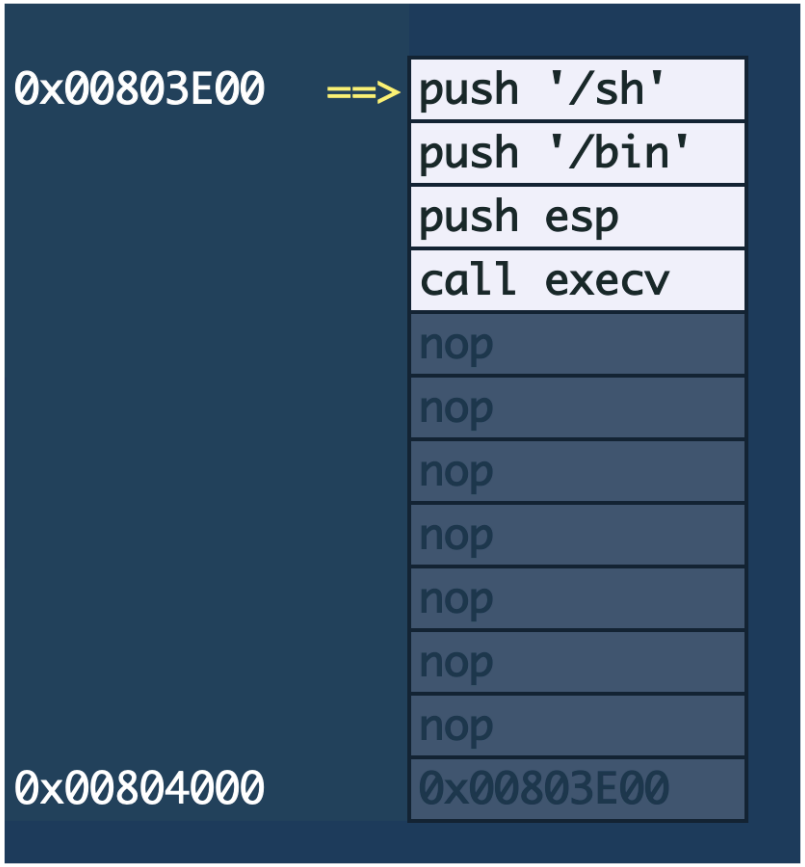
Stack Overflow

```
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
==> return 1;
}
```



Stack Overflow

```
int QueryUser()
{
    char buffer[512];
    gets(buffer);
    ...
    return 1;
}
```



Módulo 02 – Assembly - POP

POP

Remove um conteúdo armazenado na pilha o colocando em um registrador

Exemplo: POP EAX, remove o conteúdo da pilha e o armazena no reistrador EAX.

Supondo que antes da execução do POP o endereçamento do topo da pilha seja 0x0000000A (decimal 10) após o POP o topo da pilha é incrementado no mesmo tamanho do dado que foi removido. Neste nosso caso removemos 4 bytes, então o novo top da pilha passará a ser 0x0000000E (decimal 14), ou seja, 4 bytes a mais.

Módulo 02 - POP

```
            ...
    0x7F200100    push 0x22002200
    0x7F200105    push 0x00110011
==> 0x7F20010A    pop eax
    0x7F20010B    pop edx
    0x7F20010C    ...
            ...
```

EAX	0x00000000
ECX	0x00000000
EDX	0x00000000
EIP	0x7F20010A

0x00803FF0		0x00000000
0x00803FF4		0x00000000
0x00803FF8	==>	0x00110011
0x00803FFC		0x22002200
0x00804000		0x00C0FFEE

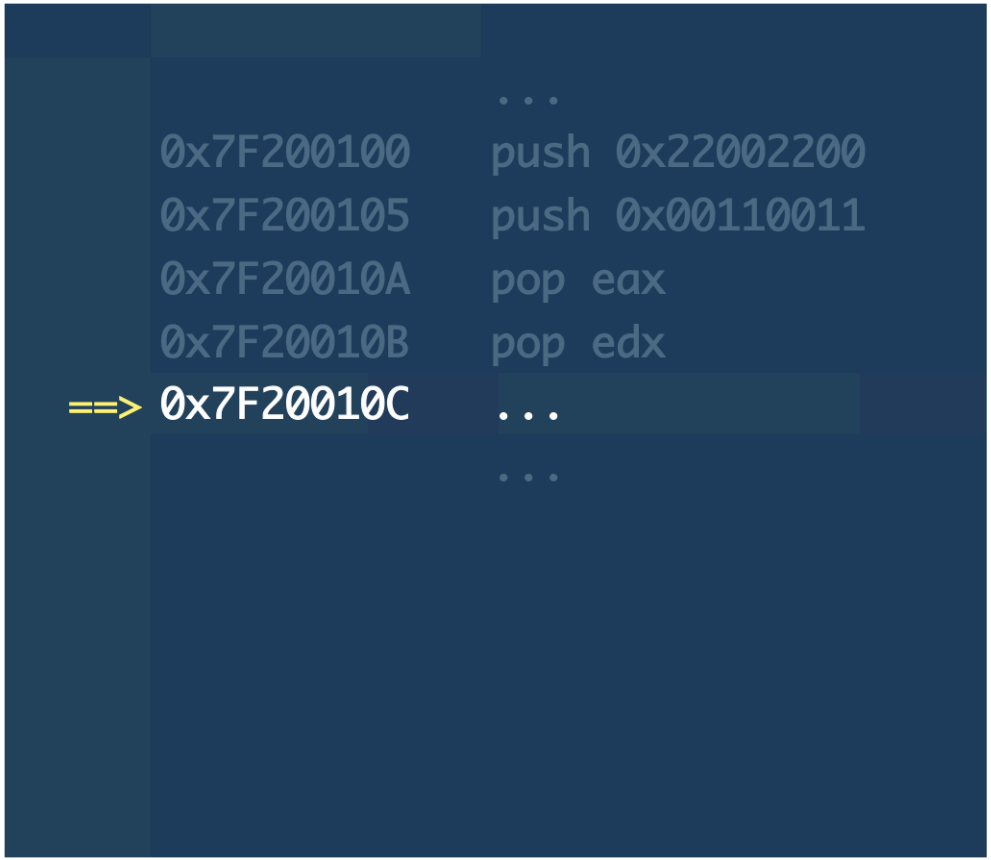
Módulo 02 - POP

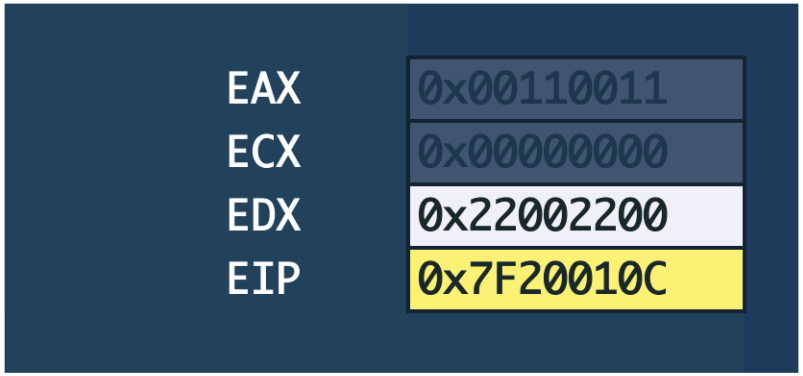
```
                ...
    0x7F200100  push 0x22002200
    0x7F200105  push 0x00110011
    0x7F20010A  pop eax
==> 0x7F20010B  pop edx
    0x7F20010C  ...
                ...
```

EAX	0x00110011
ECX	0x00000000
EDX	0x00000000
EIP	0x7F20010B

0x00803FF0		0x00000000
0x00803FF4		0x00000000
0x00803FF8		0x00110011
0x00803FFC	==>	0x22002200
0x00804000		0x00C0FFEE

Módulo 02 - POP





Módulo 02 – Assembly - RET

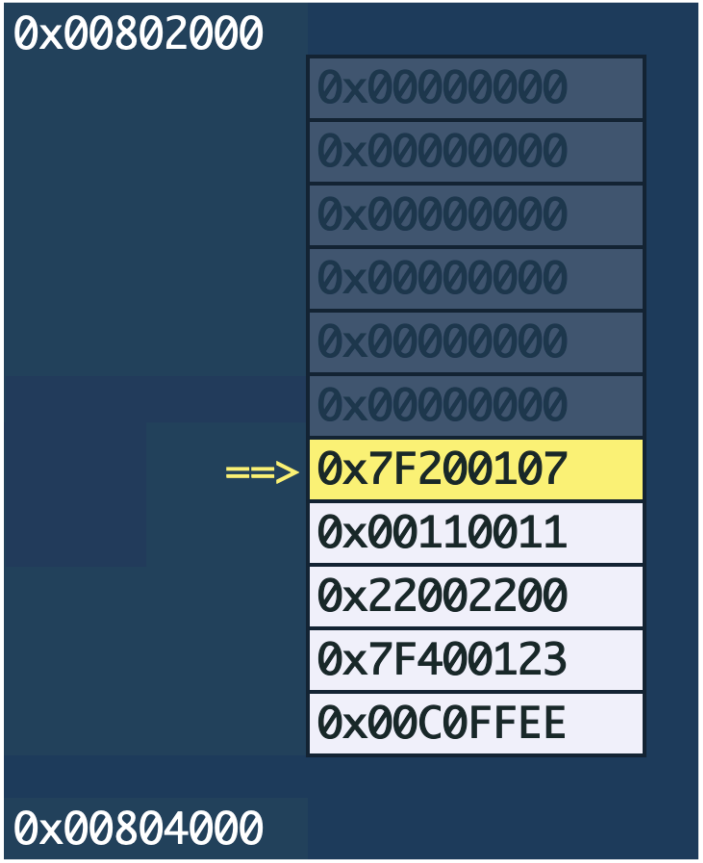
RET

Transfere o fluxo de execução da aplicação para o endereço localizado no topo da pilha.

Supondo que no topo da pilha contenha o endereço 0x01010101 o RET poderia ser comparado como igual a um JMP 0x01010101.

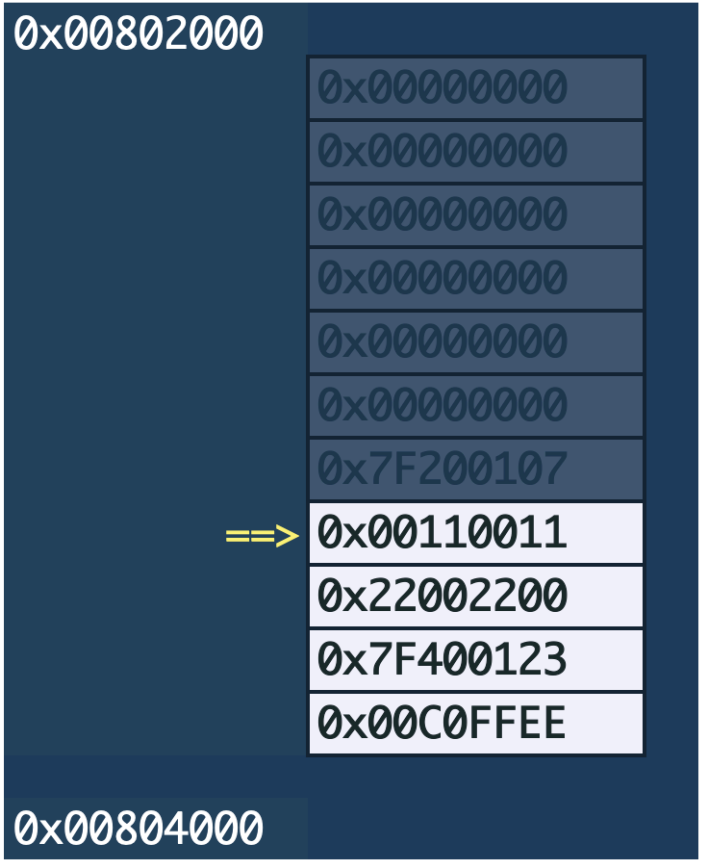
Módulo 02 - Assembly - CALL

```
    FuncFirst:
                ...
    0x7F200100  push 0x22002200
    0x7F200101  push 0x00110011
    0x7F200102  call FuncSecond
    0x7F200107  ...

    FuncSecond:
    0x7F204C00  sub esp, 0x8
    0x7F204C03  ...
    0x7F204D19  add esp, 0x8
==> 0x7F204D1C  ret
```



Módulo 02 - Assembly - CALL

```
    FuncFirst:
                    ...
    0x7F200100      push 0x22002200
    0x7F200101      push 0x00110011
    0x7F200102      call FuncSecond
==> 0x7F200107      ...

    FuncSecond:
    0x7F204C00      sub esp, 0x8
    0x7F204C03      ...
    0x7F204D19      add esp, 0x8
    0x7F204D1C      ret
```



Shellcoding

Passagem de parâmetros

- Arquitetura 32 bits
  - esp+0x00 : Primeiro parâmetro
  - esp+0x04 : Segundo parâmetro
  - esp+0x08 : Terceiro parâmetro
  - esp+0x0C : Quarto parâmetro
  - esp+0x10 : Quinto parâmetro
  - ...
- Resultado da função, quando existir, ocorre em EAX
- Pseudo-código

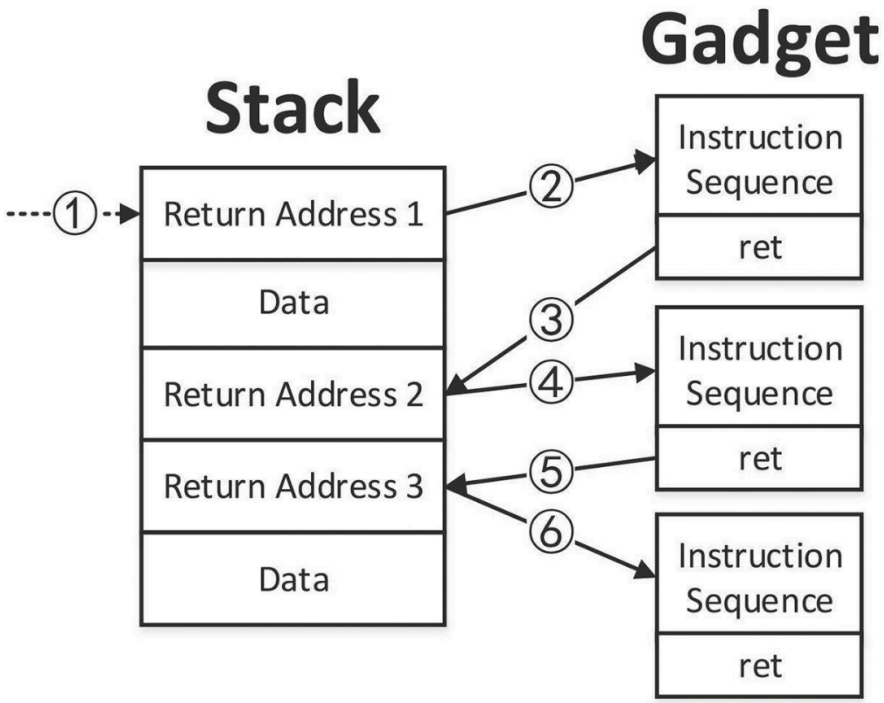
```
eax = Func1(esp+00, esp+04, esp+08, esp+0C, esp+10, …)
```

Shellcoding

```
int func1(int, int, int);

int caller(void)
{
    return func1(1, 2, 3);
}
```

```
...
; Coloca os parâmetros na pilha em ordem reversa
push    3
push    2
push    1

; Alguns compiladores adicionam o espaço na pilha
; posteriormente copiam os dados diretamente
; sub esp, 12
; mov [ebp-4], 3    : ou mov [esp+8], 3
; mov [ebp-8], 2    : ou mov [esp+4], 2
; mov [ebp-12], 1   : ou mov [esp], 1

call    func1       ; chama a função
```

ROP - Return Oriented Programming



Fonte: http://trust.csu.edu.cn/conference/SpaCCS2016/BestPaper.pdf

Hora da maldade

Explorando com ROP!

Shellcoding

```
int func1(int, int, int);

int caller(void)
{
    return func1(1, 2, 3);
}
```

```
...
; Coloca os parâmetros na pilha em ordem reversa
push    3
push    2
push    1

; Alguns compiladores adicionam o espaço na pilha
; posteriormente copiam os dados diretamente
; sub esp, 12
; mov [ebp-4], 3    : ou mov [esp+8], 3
; mov [ebp-8], 2    : ou mov [esp+4], 2
; mov [ebp-12], 1   : ou mov [esp], 1

call    func1       ; chama a função
```

Obrigado!

helvio_junior@hotmail.com
@helvio_m4v3r1ck