FUNCIONAMENTO DO STACK CHEAT SHEET

© JOSÉ CARLOS FONSECA (JOSEFONSECA@IPG.PT), 2020

Registo	Função
ESP	(**Stack Pointer**) Apontador para o último endereço do Stack, que é o do topo, apesar de ser o de endereço inferior
EBP	(**Stack Base Frame Pointer**) Apontador para o endereço do Stack Frame actual, que é usado no acesso ao Stack
EIP	(**Instruction Pointer**) Apontador para o endereço da próxima instrução que irá ser executada

```
7     void main() {
   0x080483fe <+0>:     push   %ebp
   0x080483ff <+1>:     mov    %esp,%ebp
   0x08048401 <+3>:     sub    $0xc,%esp
8        function(1,2,3);
   0x08048404 <+6>:     movl   $0x3,0x8(%esp)
   0x0804840c <+14>:    movl   $0x2,0x4(%esp)
   0x08048414 <+22>:    movl   $0x1,(%esp)
   0x0804841b <+29>:    call   0x80483e4 <function>
9     }
   0x08048420 <+34>:    leave
   0x08048421 <+35>:    ret
```

Prólogo do main

Acrescenta `0xC=12` posições ao Stack, subtraindo esse valor a ESP (topo do Stack). Isto dá para os 3 argumentos (4 bytes cada) da chamada da function

3º argumento da function com o valor c=3 no Stack, 8 posições acima de ESP

2º argumento da function com o valor b=2 no Stack, 4 posições acima de ESP

1º argumento da function com o valor a=1 no Stack, na posição do ESP

PUSH do EIP com o endereço da próxima instrução e chamada da function

Epílogo do main

```
1     void function(int a, int b, int c) {
   0x080483e4 <+0>:     push   %ebp
   0x080483e5 <+1>:     mov    %esp,%ebp
   0x080483e7 <+3>:     sub    $0x18,%esp
2        char buffer1[5];
3        char buffer2[10];
4        strcpy(buffer2,buffer1);
   0x080483ea <+6>:     lea    -0x5(%ebp),%eax
   0x080483ed <+9>:     mov    %eax,0x4(%esp)
   0x080483f1 <+13>:    lea    -0xf(%ebp),%eax
   0x080483f4 <+16>:    mov    %eax,(%esp)
   0x080483f7 <+19>:    call   0x804831c <strcpy@plt>
5     }
   0x080483fc <+24>:    leave
   0x080483fd <+25>:    ret
```

Guarda no Stack o EBP do Stack Frame do main. O ESP aponta para lá

Copia para EBP o valor do ESP. Este é o EBP do Stack Frame da function

Prólogo da function

Acrescenta `0x18=24` posições ao Stack, subtraindo esse valor a ESP. Dá para os 2 buffers (5+10) e para os 2 argumentos do strcpy (4+4), arredondado para múltiplo de 4 (5+10+4+4=23, 23+1=24 =6*4, logo múltiplo de 4)

Copia para o registo acumulador EAX o 2º argumento do strcpy, que é o endereço de buffer1 que se encontra na 5ª posição do Stack

Copia o conteúdo do registo acumulador EAX para a posição 4 a contar do topo do Stack (ESP). Este é o 2º argumento do strcpy

Copia para o registo acumulador EAX o 1º argumento do strcpy, que é o endereço de buffer2 que se encontra na 11ª posição do Stack

Copia o conteúdo do registo acumulador EAX para o topo do Stack (ESP). Este é o 1º argumento do strcpy

Chama a função strcpy que copia o conteúdo de buffer 1 para buffer2

Epílogo da function

Liberta o Stack da function. É o mesmo que `mov %ebp,%esp` seguido de `pop %ebp`. O EBP fica com o valor de EBP do Stack Frame do main e o ESP fica a apontar para o endereço de retorno (EIP)

POP do endereço de retorno (EIP) e mudança de execução para esse endereço

Stack antes da chamada da função strcpy

Endereço	Conteúdo (MSB … LSB)	O que está armazenado no Stack
0xbffff79c	0xbffff7a5	1º arg. do strcpy com endereço do buffer2
0xbffff7a0	0xbffff7af	2º arg. do strcpy com endereço do buffer1
0xbffff7a4	0x0029bff4	bytes para ter endereço múltiplo de 4: **f4**
0xbffff7a8	0x08048440	**buffer2:** bf290040840408c8f7ff
0xbffff7ac	0xbffff7c8	**buffer1:** bfa5641700
0xbffff7b0	0x001764a5	
0xbffff7b4	0xbffff7c8	EBP do Stack Frame anterior (do main)
0xbffff7b8	0x08048420	endereço de retorno (EIP)
0xbffff7bc	0x00000001	1º argumento da function com o valor **a=1**
0xbffff7c0	0x00000002	2º argumento da function com o valor **b=2**
0xbffff7c4	0x00000003	3º argumento da function com o valor **c=3**

Apontador do ESP da function

Stack Frame da function

Apontador do EBP da function

Apontador do ESP do main

Stack Frame do main