

UNIVERSIDADE DO MINHO
CRİPTOGRAFIA E SEGURANÇA DA INFORMAÇÃO
DEPARTAMENTO DE INFORMÁTICA

ENGENHARIA DE SEGURANÇA

TP8 - Aula 11

Grupo 7

Carlos Pinto Pedrosa A77320

José Francisco Gonçalves Petejo e Igreja Matos A77688

7 de Maio de 2019

Conteúdo

1	Buffer Overflow	2
1.1	Pergunta 1.1	2
1.2	Pergunta 1.2	3
1.3	Pergunta 1.3	4
1.4	Pergunta 1.4	5
1.5	Pergunta 1.5	6
2	Vulnerabilidades de inteiros	7
2.1	Pergunta 2.1	7
2.2	Pergunta 2.2	8

De forma a demonstrar esta falha, inseriu-se uma variável *j* com a string *Ola* e imprimiu-se em cada iteração o valor inserido pelo utilizador. O resultado demonstra que o utilizador tem acesso a informação que em teoria não deveria ter.

```
francisco@francisco-X550JK ~ $ /usr/bin/python /home/francisco/Desktop/4ºAno/ES/L0verflow2.py
Quantos numeros? 9
Insira numero: 1
1
Insira numero: 2
2
Insira numero: j
Ola
Insira numero: i
3
```

Figura 4: Falha de segurança da função input em python

- Java - Por fim, o programa em java provou-se ser o mais robusto, lançando exceções sempre que eram utilizados ou números muito grandes, ou não números. O programa funciona como esperado para todos os outros casos.

```
francisco@francisco-X550JK ~/Desktop/4ºAno/ES $ java L0verflow2
Quantos números? a
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at L0verflow2.main(L0verflow2.java:12)
francisco@francisco-X550JK ~/Desktop/4ºAno/ES $ java L0verflow2
Quantos números? 32114218942184621896482196421896218461894
Exception in thread "main" java.util.InputMismatchException: For input string: "32114218942184621896482196421896218461894"
    at java.util.Scanner.nextInt(Scanner.java:2123)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at L0verflow2.main(L0verflow2.java:12)
```

Figura 5: Exceção originada de input inesperado em Java

1.2 Pergunta 1.2

- C++ - O programa em c++ possui algumas vulnerabilidades, isto é, é possível ter acesso a valores do array que não deveriam ser possíveis, pois este não protege o array. Isto é, quando se utiliza números entre 0 e o total de valores a guardar no array, funciona como esperado, no entanto, com números negativos, seria de esperar que ou desse erro ou que não devolvesse nenhum valor, no entanto é possível obtê-los, como é visível na seguinte imagem

```
francisco@francisco-X550JK ~ $ g++ /home/francisco/Desktop/4ºAno/ES/L0verflow3.cpp -o /home/francisco/Desktop/4ºAno/ES/L0verflow3 66 /home/francisco/Desktop/4ºAno/ES/L0verflow3
Quantos valores quer guardar no array? 10
Que valor deseja recuperar? -3
0 valor e -1
```

Figura 6: Acesso a dados externos do array em c++

- Python - O programa em python já é mais robusto e apenas retorna valores contidos no array, impedindo o utilizador de aceder a dados externos. Quando são utilizados valores superiores fora do array, este levanta um erro, ajudando a impedir acessos maliciosos

```
francisco@francisco-X550JK ~ $ /usr/bin/python /home/francisco/Desktop/4ºAno/ES/L0verflow3.py
Quantos valores quer guardar no array? 2
Que valor deseja recuperar? 10
0 valor e
Traceback (most recent call last):
  File "/home/francisco/Desktop/4ºAno/ES/L0verflow3.py", line 7, in <module>
    print '0 valor e ', str(vals[which])
IndexError: list index out of range
```

Figura 7: Erro de acesso superior ao array em python

- Java - Por fim, mais uma vez, java é a opção mais segura, levantando sempre uma exceção quando o utilizador executa qualquer input fora do esperado

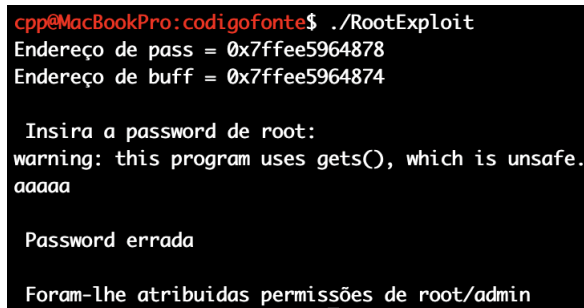
```
Quantos valores quer guardar no array?
10
Que valor deseja recuperar?
-2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -2
    at L0verflow3.main(L0verflow3.java:19)
francisco@francisco-X550JK ~/Desktop/4ºAno/ES $ java L0verflow3
Quantos valores quer guardar no array?
10
Que valor deseja recuperar?
12431241244
Exception in thread "main" java.util.InputMismatchException: For input string: "12431241244"
    at java.util.Scanner.nextInt(Scanner.java:2123)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at L0verflow3.main(L0verflow3.java:17)
francisco@francisco-X550JK ~/Desktop/4ºAno/ES $
```

Figura 8: Exceções levantadas em java

1.3 Pergunta 1.3

O nosso objetivo neste exercício passa por manipular a stack de forma a atingir a variável que verifica a password *pass*. Para tal é preciso primeiro

entender o funcionamento da stack, então realizou-se uns testes para entender o endereço do *buf* e da *pass*. Sendo assim, foi possível inferir que a diferença destes era 4 bits, por isso, no momento de inserção da palavra *pass* do root, apenas é preciso inserir 4 valores para preencher o buffer, e depois inserir qualquer valor, de forma a ativar a variável *pass* e entrar na secção que atribui permissões de admin, como é visível na seguinte imagem.

A terminal window with a black background and white text. The prompt is 'cpp@MacBookPro:codigofonte\$'. The user enters './RootExploit'. The program outputs 'Endereço de pass = 0x7ffee5964878' and 'Endereço de buff = 0x7ffee5964874'. It then prompts 'Insira a password de root:' and shows a warning: 'warning: this program uses gets(), which is unsafe.' followed by 'aaaaa'. The program then outputs 'Password errada' and finally 'Foram-lhe atribuidas permissões de root/admin'.

```
cpp@MacBookPro:codigofonte$ ./RootExploit
Endereço de pass = 0x7ffee5964878
Endereço de buff = 0x7ffee5964874

Insira a password de root:
warning: this program uses gets(), which is unsafe.
aaaaa

Password errada

Foram-lhe atribuidas permissões de root/admin
```

Figura 9: Overflow

1.4 Pergunta 1.4

Tendo em conta que este programa é utilizado para encontrar casos de overflow, será esse os testes que iremos fazer. Mas primeiramente, é importante perceber o objetivo do programa, que consiste na impressão de *n* caracteres definidos pelo utilizador de uma frase também definida por este. Através da leitura de valores superiores aqueles passados, podemos aceder a dados externos da memória, como é visível na imagem apresentada.

2 Vulnerabilidades de inteiros

2.1 Pergunta 2.1

A função vulneravel utiliza os valores x e y com o tipo size_t. Este tipo deve ser usado com especial atenção para não ser dada a possibilidade ao utilizador de aceder a essas variáveis, quer seja em definir o seu valor diretamente, ou através de overflow. Este tipo está especialmente vulnerável a ataques de underflow.

Completando então a main com o seguinte código:

```
int main() {  
    int underflow = -4;  
    char *matriz;  
    printf("Matriz: %d\n", underflow);  
    vulneravel(matriz, underflow, underflow, '1');  
    return 0;  
}
```

Figura 12: Main

Com esta main foi possível obter uma segmentation fault:

```
francisco@francisco-X550JK ~/Desktop/4ºAno/ES $ ./"overflow"  
Matriz: -4  
Segmentation fault (core dumped)  
francisco@francisco-X550JK ~/Desktop/4ºAno/ES $
```

Figura 13: Segmentation Fault

2.2 Pergunta 2.2

Mais uma vez, a utilização de `size_t` e operações que a envolvem originam possíveis cenários de underflow. O código da main para provar este cenário é o seguinte, e obteve-se, mais uma vez segmentation fault

```
francisco@francisco-X550JK ~/Desktop/4ºAno/ES $ ./"underflow"  
Segmentation fault (core dumped)  
francisco@francisco-X550JK ~/Desktop/4ºAno/ES $
```

Figura 14: Main

```
int main() {  
    char origem[8] = "segredo";  
    vulneravel(origem, 0);  
}
```

Figura 15: Segmentation Fault