# Universidade do Minho Criptografia 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	Buf	fer 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	
$oldsymbol{2}$	Vulnerabilidades de inteiros			
	2.1	Pergunta 2.1	7	
	2.2	Pergunta 2.2	8	

#### 1 Buffer Overflow

#### 1.1 Pergunta 1.1

• C++ - Começando pelo ficheiro escrito em c++ foram testadas diversas possibilidades. Verificou-se o comportamento esperado quando foi utilizado um número razoavelmente pequeno. Quando foi utilizado um número negativo ou 0, o programa também funcionou como esperado, terminando a sua execeução. No entanto, utilizando um número razoavelmente grande, no nosso caso 31, depois de se inserir alguns números foi apresentado stack smashing detected e o programa foi interrompido.

```
Insira número: 213

*** stack smashing detected ***: /home/francisco/Desktop/4°Ano/ES/LOverflow2 terminated
Aborted (core dumped)
```

Figura 1: Stack smashing em c++

Por fim, utilizou-se um valor ainda maior, que causou com que o programa imprimi-se "Insira número" repetidamente até eventualmente originar um segmentation fault.



Figura 2: Segmentation Fault em c++

• Python - Quanto ao ficheiro python, este demonstrava um funcionamento normal quando eram utilizados números pequenos, negativos ou zero. No entanto, com números como 21, já lançava um erro.

```
Insira numero: 5
Fraceback (most recent call last):
   File "/home/francisco/besktop/4ºAno/ES/LOverflow2.py", line 5, in <module>
    tests[i]=test
IndexError: list assignment index out of range
franciscoefrancisco-X550JK ~ $ 5
```

Figura 3: Index out of range em python

Ao realizar alguns testes foi possível perceber que a função *input* não trata os dados inseridos com segurança, permitindo o acesso ao código.

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/4°Ano/ES/LOverflow2.py
Quantos numeros? 9
Insira numero: 1
1
Insira numero: 2
2
Insira numero: j
Ola
Insira numero: 1
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/4°Ano/ES $ java LOverflow2
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:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at LOverflow2.main(LOverflow2.java:12)
francisco@francisco-X550JK ~/Desktop/4°Ano/ES $ java LOverflow2
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:2127)
    at LOverflow2.main(LOverflow2.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 acessoa 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 visivel na seguinte imagem

```
franciscoeffrancisco-XSSOIK ~ $ g++ /home/francisco/Desktop/4ºAno/ES/LOverflow3.cpp -o /home/francisco/Desktop/4ºAno/ES/LOverflow3 & /home/francisco/Deskto
```

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 sao utilizados valores superiormente fora do array, este levanta um erro, ajudando a impedir acessos maliciosos

```
francisco@francisco-X550JK ~ $ /usr/bin/python /home/francisco/Desktop/4ºAno/ES/LOverflow3.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/LOverflow3.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

```
Que valor deseja recuperar?

-2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -2
    at LOverflow3.main(LOverflow3.java:19)

francisco@francisco-X550JK ~/Desktop/4ºAno/ES $ java LOverflow3
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 LOverflow3.main(LOverflow3.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 valore 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 vísivel na seguinte imagem.

```
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 caratéres 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 vísivel na imagem apresentada.

Figura 10: Overflow

Neste caso, ao pedir para ler 8000 caráteres de uma frase inexistente, podemos ler informações relativas à maquina, algo que põe em causa a segurança do programa.

## 1.5 Pergunta 1.5

Tal como no exercício 1.3, pretende-se alterar a variável de controlo, control de forma a possuir o valor 0x61626364. Para tal, tentamos começar por apenas alterar para qualquer valor. Então, encheu-se o buffer com o valor 9, até se identificar alguma alteração na variável de controlo. A partir daí foi preciso apenas inferir quais os valores necessários para chegar ao valor 0x61626364 que corresponde a abcd, mas como se trata de uma máquina little-endian o valor ficaria 0x64636261, então o verdadeiro código que descodifica é dcba.

Figura 11: Overflow

#### 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, 'l');
    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 involvem 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