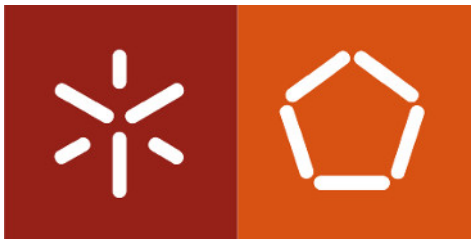

Mestrado em Engenharia Informática
Universidade do Minho

Engenharia de Segurança

Aula 10 TP - 30/04/2020

João Miranda - PG41845
Sandro Cruz - PG41906

1 de Maio de 2020

Conteúdo

Capítulo 1

Vulnerabilidade de inteiros

1.1 Experiência 1.1

```
user@CSI:~/Desktop/engseg$ javac IntegerCheck.java
user@CSI:~/Desktop/engseg$ java IntegerCheck
int válido   entre -2147483648 e 2147483647
byte válido  entre -128 e 127
short válido entre -32768 e 32767
long válido  entre -9223372036854775808 e 9223372036854775807
user@CSI:~/Desktop/engseg$
```

Figura 1.1: Valores máximos e mínimos dos vários inteiros da nossa máquina

As variáveis "int"são de 32 bits, as "shorts"tem 16 bits, os "bytes"tem "8"bits e os "longs"tem 64 bits.

1.2 Experiência 1.2

```
user@CSI:~/Desktop/engseg$ java IntegerError
Maior inteiro: 2147483647
Menor inteiro: -2147483648
Input de dois valores inteiros: 1
2

Valores entrados:
Inteiros: 1 2
Multiplicação do primeiro número por dez: 10
Soma dos dois números: 3
Produto dos dois números: 2
user@CSI:~/Desktop/engseg$
```

Figura 1.2: Output recebido inserindo um inteiro no "range"dos "bytes"

```
user@CSI:~/Desktop/engseg$ java IntegerError
Maior inteiro: 2147483647
Menor inteiro: -2147483648
Input de dois valores inteiros: 150
325

Valores entrados:
Inteiros: 150 325
Multiplicação do primeiro número por dez: 1500
Soma dos dois números: 475
Produto dos dois números: 48750
user@CSI:~/Desktop/engseg$
```

Figura 1.3: Output recebido inserindo um inteiro no "range"dos "shorts"

```
user@CSI:~/Desktop/engseg$ java IntegerCheck
int válido   entre -2147483648 e 2147483647
byte válido  entre -128 e 127
short válido entre -32768 e 32767
long válido  entre -9223372036854775808 e 9223372036854775807
user@CSI:~/Desktop/engseg$ java IntegerError
Maior inteiro: 2147483647
Menor inteiro: -2147483648
Input de dois valores inteiros: 2147483660
Exception in thread "main" java.util.InputMismatchException: For input string: "
2147483660"
        at java.base/java.util.Scanner.nextInt(Scanner.java:2167)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2115)
        at IntegerError.main(IntegerError.java:17)
user@CSI:~/Desktop/engseg$
```

Figura 1.4: Output recebido inserindo um inteiro no "range"dos "longs"

Quando é inserido um valor superior ao valore máximo do "int"é lançada uma excepção. Para que fossem aceitados valores superiores seria necessário declarar as variáveis que armazenam os "ints"como "longs".

1.3 Experiência 1.3

```
user@CSI:~/Desktop/engseg$ java IntegerCheck2
Int válido   entre -2147483648 e 2147483647
Byte válido  entre -128 e 127
Short válido entre -32768 e 32767
Long válido  entre -9223372036854775808 e 9223372036854775807
Insira valor Int: 2147483647
Insira valor Byte: 127
Insira valor Short: 32767
Insira valor Long: 9223372036854775807

Inseriu os seguintes valores:
Int: 2147483647
Byte: 127
Short: 32767
Long: 9223372036854775807
Integer Overflow: 2147483647 + 1 = -2147483648
user@CSI:~/Desktop/engseg$
```

Figura 1.5: Output recebido inserindo valores esperados

```
user@CSI:~/Desktop/engseg$ java IntegerCheck2
Int válido   entre -2147483648 e 2147483647
Byte válido  entre -128 e 127
Short válido entre -32768 e 32767
Long válido  entre -9223372036854775808 e 9223372036854775807
Insira valor Int: 2147483649
Insira valor Byte: 129
Insira valor Short: 32768
Insira valor Long: 9223372036854775900
Exception in thread "main" java.util.InputMismatchException: For input string: "
9223372036854775900"
        at java.base/java.util.Scanner.nextLong(Scanner.java:2282)
        at java.base/java.util.Scanner.nextLong(Scanner.java:2231)
        at IntegerCheck2.main(IntegerCheck2.java:22)
user@CSI:~/Desktop/engseg$
```

Figura 1.6: Output recebido inserindo valores inesperados

```
user@CSI:~/Desktop/engseg$ java IntegerCheck2
Int válido   entre -2147483648 e 2147483647
Byte válido  entre -128 e 127
Short válido entre -32768 e 32767
Long válido  entre -9223372036854775808 e 9223372036854775807
Insira valor Int: 2147483649
Insira valor Byte: 129
Insira valor Short: 32768
Insira valor Long: 9223372036854775807

Inseriu os seguintes valores:
Int: -2147483647
Byte: -127
Short: -32768
Long: 9223372036854775807
Integer Overflow: 2147483647 + 1 = -2147483648
user@CSI:~/Desktop/engseg$
```

Figura 1.7: Output recebido inserindo valores inesperados, expecto para o "long"

```
user@CSI:~/Desktop/engseg$ java IntegerCheck2
Int válido   entre -2147483648 e 2147483647
Byte válido  entre -128 e 127
Short válido entre -32768 e 32767
Long válido  entre -9223372036854775808 e 9223372036854775807
Insira valor Int: 214748364
Insira valor Byte: 127
Insira valor Short: 32767
Insira valor Long: 214748364\
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:860)
        at java.base/java.util.Scanner.next(Scanner.java:1497)
        at java.base/java.util.Scanner.nextLong(Scanner.java:2276)
        at java.base/java.util.Scanner.nextLong(Scanner.java:2231)
        at IntegerCheck2.main(IntegerCheck2.java:22)
user@CSI:~/Desktop/engseg$
```

Figura 1.8: Output recebido inserindo um valor de um outro tipo na no input para "long"

Caso seja inserido um "int"no input do "long"é lançada uma excepção. Caso sejam inseridos os dados correctos é nos dado sempre o mesmo output.

1.4 Pergunta P1.1

Analise o programa overflow.c.

1.4.1 Qual a vulnerabilidade que existe na função vulneravel() e quais os efeitos da mesma?

A vulnerabilidade existente é a possibilidade de existir um overflow das variáveis "i"e "j"dado que são inteiros que vão ser iterados sobre uma variável da qual o inteiro que contem pode ser superior ao maior inteiro disponível, o que iria criar

possivelmente um erro em que o ciclo "for"seria infinito. Também corre o risco de existir um erro de segmentação dado que caso o valor de uma das variáveis "i"ou "j"passe a um valor negativo o programa vai tentar armazenar num valor de "index"negativo um valor, o que vai causar o erro de segmentação.

1.4.2 Complete o main() de modo a demonstrar essa vulnerabilidade.

```c
#include <stdio.h>
#include <stdlib.h>

void vulneravel (char *matriz, size_t x, size_t y, char valor) {
        int i, j;
        matriz = (char *) malloc(x*y);
        for (i = 0; i < x; i++) {
                for (j = 0; j < y; j++) {
                        matriz[i*y+j] = valor;
                }
        }
}

int main() {
      char matriz;
      vulneravel(matriz, 2147483649, 2147483649, "x");
}
```

Figura 1.9: Modificações feitas a função "main"deforma a demonstrar a vulnerabilidade

1.4.3 Ao executar dá algum erro? Qual?

```
user@CSI:~/Desktop/engseg$ ./overflow
Segmentation fault
user@CSI:~/Desktop/engseg$
```

Figura 1.10: Execução do programa "overflow"

Como podemos verificar ocorreu um erro de segmentação, isto ocorreu porque o programa tentou escrever num "index"negativo da "matriz"um valor.

1.5 Pergunta P1.2

Analise o programa underflow.c.

1.5.1 Qual a vulnerabilidade que existe na função vulneravel() e quais os efeitos da mesma?

Se a variável "tamanho"for 0 o programa vai tentar alocar memoria com um valor negativo, o que vai causar com que não sejam alocada a memoria, e quando é feito o "memcpy"vai causar um erro porque a variável destino não terá memoria alocada.

1.5.2 Complete o main() de modo a demonstrar essa vulnerabilidade.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int MAX_SIZE = 2048;


void vulneravel (char *origem, size_t tamanho) {
        size_t tamanho_real;
        char *destino;
        if (tamanho < MAX_SIZE) {
                tamanho_real = tamanho - 1; // Não copiar \0 de origem para destino
                destino = (char *) malloc(tamanho_real);
                memcpy(destino, origem, tamanho_real);
        }
}

int main() {
      char *origem = "Variavel Origem";
      vulneravel(origem,0);
}
```

Figura 1.11: Alterações feitas na "main"para demonstrar a vulnerabilidade

1.5.3 Ao executar dá algum erro? Qual?

```
user@CSI:~/Desktop/engseg$ gcc -o underflow underflow.c
user@CSI:~/Desktop/engseg$ ./underflow
Segmentation fault
user@CSI:~/Desktop/engseg$
```

Figura 1.12: Execução do programa e o output recebido

Um erro de segmentação dado que o programa tentar gravar os dados num variável nula.

1.5.4 Utilize as várias técnicas de programação defensiva introduzidas na aula teórica para mitigar as vulnerabilidades

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int MAX_SIZE = 2048;
const int MIN_SIZE = 0;

void vulneravel (char *origem, size_t tamanho) {
        size_t tamanho_real;
        char *destino;
        if (tamanho < MAX_SIZE && tamanho > MIN_SIZE) {
                tamanho_real = tamanho - 1; // Não copiar \0 de origem para destino
                destino = (char *) malloc(tamanho_real);
                memcpy(destino, origem, tamanho_real);
        };
        else{
                printf("Valor nao valido\n");
        }
}

int main() {
      char *origem = "Variavel Origem \n";
      vulneravel(origem, 0);
}
```

Figura 1.13: Alterações feitas na função "vulneravel"para mitigar as vulnerabilidades

Explique as alterações que fez.

A alteração feita foi a introdução da variável "MIN_SIZE", que é utilizada juntamente com a "MAX_SIZE"para filtrar os valores válidos que a variável "tamanho"poderá ser compreendida.

1.6 Experiência 1.4

Analise o programa erro_sinal.c.

1.6.1 Qual a vulnerabilidade que existe na função vulneravel() e quais os efeitos da mesma?

Se no "memcpy"o valor recebido como parâmetro do número de bytes a serem copiados for muito grande pode causar um erro de segmentação na variável de destino dos "memcpy".

1.6.2 Complete o main() de modo a demonstrar essa vulnerabilidade.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const int MAX_SIZE = 2048;


void vulneravel (char *origem, size_t tamanho) {
        int tamanho_real;
        char *destino;
        if (tamanho > 1) {
                tamanho_real = tamanho - 1; // Não copiar \0 de origem para destino
                if (tamanho_real < MAX_SIZE) {
                        destino = (char *) malloc(tamanho_real);
                        memcpy(destino, origem, tamanho_real);
                }
        }
}

int main() {

      char *origem = "Variavel Origem";
      vulneravel(origem, 2010);

}

```

Figura 1.14: Alterações feitas na "main"para demonstrar a vulnerabilidade

1.6.3 Ao executar dá algum erro? Qual?

```
user@CSI:~/Desktop/engseg$ gcc -o erro_sinal erro_sinal.c
user@CSI:~/Desktop/engseg$ ./erro_sinal
Segmentation fault
user@CSI:~/Desktop/engseg$
```

Figura 1.15: Execução do programa e o output recebido

Acontece um erro de segmentação, isto ocorre no "memcpy"porque o valor da variável "tamanho_real"é demasiado grande para a variável "destino"receber da variável "origem".

1.7 Experiência 1.5

Analise o programa y2k38.c.

1.7.1 Compile-o e execute-o. Do resultado obtido o que pode dizer sobre a arquitetura onde o mesmo foi copilado? Porquê?

```
user@CSI:~/Desktop/engseg$ gcc -o y2k38 y2k38.c
user@CSI:~/Desktop/engseg$ ./y2k38
1000000000, Sun Sep  9 01:46:40 2001
2147483647, Tue Jan 19 03:14:07 2038
-2147483648, Tue Jan 19 03:14:08 2038
user@CSI:~/Desktop/engseg$
```

Figura 1.16: Compilação, sem a opção -m32, e execução do programa

Podemos concluir que foi compilado numa arquitetura x86_64 dado que a "timestamp"do ano 2038 será de valor negativa pois o número máximo dos inteiro será atingido, mas a data volta para a data inicial em 1901 o que demonstra que é uma arquitectura do tipo x86_64.

1.7.2 Compile-o agora com a opção -m32 do gcc. Qual o resultado? Porquê?

```
user@CSI:~/Desktop/engseg$ gcc -o y2k38m32 -m32 y2k38.c
user@CSI:~/Desktop/engseg$ ./y2k38m32
1000000000, Sun Sep  9 01:46:40 2001
2147483647, Tue Jan 19 03:14:07 2038
-2147483648, Fri Dec 13 20:45:52 1901
user@CSI:~/Desktop/engseg$
```

Figura 1.17: Compilação, com a opção -m32, e execução do programa

Neste caso a arquitetura é de 32-bits, porque quando chega ao limite dos inteiros a data volta para a data inicial em 1901, o que revela que foi compilado em 32-bits.