Universidade do Minho

Mestrado Integrado em Engenharia Informática

Engenharia de Segurança

Trabalho TP8

Grupo 2
Paulo Gameiro - A72067
Pedro Rodrigues - PG41092
Rafaela Soares - A79034

Braga, Portugal
4 de Maio de 2020

Conteúdo

1 Experiência 1.1

```
int válido   entre -2147483648 e 2147483647
byte válido  entre -128 e 127
short válido entre -32768 e 32767
long válido  entre -9223372036854775808 e 9223372036854775807
```

2 Experiência 1.2

O **Integer Overflow** é atingido quando é introduzido um inteiro inferior ao maior permitido, sendo que, após os cálculos serem efetuados, são obtidos valores superiores ao limite, e como tal obtém-se assim um *overflow*, como é possível observar no caso da multiplicação do primeiro número por dez, em que se obtém um número negativo.

```
Input de dois valores inteiros: 2147483642 2149214

Valores entrados:
Inteiros: 2147483642 2149214
Multiplicação do primeiro número por dez: -60
Soma dos dois números: -2145334440
Produto dos dois números: -12895284
```

3 Experiência 1.3

No caso de ser atribuído o valor de um long a outro tipo de dados, dá-se um *overflow* e os valor ultrapassa o máximo desse tipo de dados, passando assim para um valor associado à memória a seguir, que será normalmente um valor negativo.

```
Insira valor Int: 15392
Insira valor Byte: 15
Insira valor Short: 9223372036854775802
Insira valor Long: 2195291

Inseriu os seguintes valores:
Int: 15392
Byte: 15
Short: -6
Long: 2195291
```

4 Pergunta P1.1

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

A função **vulneravel()** recebe dois parâmetros de tipo **size_t**, sendo que o size_t pode ser definido como o maior **unsigned integer** deste sistema, e tendo em conta que estes dois valores são utilizados num ciclo **for** em que é feita uma comparação com variáveis de tipo **int**, é possível que estas variáveis size_t contenham valores superiores aos permitidos em variáveis de tipo **int**.

Como tal, é possível fazer com que o ciclo entre em valores superiores aos valores máximos do tipo de dados **int**.

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

```
int main() {
  char *matriz;
  vulneravel(matriz, 212919210, 2218412941, 200);
}
```

Ao executar dá algum erro? Qual?

Na execução do programa, ocorre um **Segmentation Fault**, que significa que este programa tentou aceder a um endereço de memória que não existe ou que está reservado para outro programa.

5 Pergunta P1.2

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

Devido a não ser feita nenhuma verificação relativamente ao mínimo da variável **tamanho** (apenas se verifica que é menor que o MAX_SIZE), é possível passar o valor 0 (zero) e, sendo assim, a variável **tamanho_real** vai tomar o valor de -1.

Ou seja, tendo em conta que a variável **tamanho_real** é do tipo size_t pode apenas conter números positivos e como tal irá dar-se um *underflow*, atribuindo a esta variável o valor máximo do tamanho de uma variável neste sistema, ou seja, ao realizar o **malloc** deste valor, vai ultrapassar o tamanho da memória que é possível alocar e portanto retorna um resultado nulo, terminando o programa quando tentar realizar o **memcpy** pois a variável de destino tem o apontador para endereço de memória nulo.

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

```
int main() {
  char palavra[2] = "ES";
  size_t teste = 0;
  vulneravel("T", teste);
}
```

Ao executar dá algum erro? Qual?

Ao executar o programa, obtém-se novamente um **Segmentation Fault**, o que significa que foi efetuada uma tentativa de aceder a um endereço de memória fora do permitido para esta aplicação.

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

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

De forma a impedir o *underflow* da variável **tamanho_real**, foi efetuada a validação do *input* para que o valor resultante de **tamanho - 1** não seja negativo.

6 Experiência 1.4

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

Ao contrário do programa anterior, neste foi feita a validação do *input* para que a variável **tamanho** seja superior a um, mas o tipo de **tamanho_real** foi alterado de **size_t** para **int**, o que significa que se for passado como parâmetro um valor que seja superior ao máximo de um **int**, ao realizar-se o **tamanho_real = tamanho - 1**, a variável **tamanho_real** irá tomar um valor superior ao possível, incorrendo num *overflow* e ficando negativa.

Tal como na pergunta anterior, é feita uma tentativa de alocar memória que incorre num erro.

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

```
int main() {
    char *origem;
    vulneravel(origem, 2147483650);
}
```

Ao executar dá algum erro? Qual?
Ao executar o programa, obtém-se novamente um **Segmentation Fault**, o que significa que foi efetuada uma tentativa de aceder a um endereço de memória fora do permitido para esta aplicação.

7 Experiência 1.5

Quando compilado normalmente, obtemos o resultado descrito abaixo, o que nos permite concluir que estamos perante uma arquitetura de 64 *bits*, dado que nesta arquitetura o tipo de dados **time_t** é um **long int**, o que significa que o tamanho da variável faz com que a data apresentada seja a data máxima possível do asctime().

```
1000000000, Sun Sep  9 01:46:40 2001
2147483647, Tue Jan 19 03:14:07 2038
-2147483648, Tue Jan 19 03:14:08 2038
```

Quando compilamos o programa utilizando a *flag* **-m32** do gcc, significa que o programa está a ser compilado utilizando os objetos de 32 *bits*, apesar da predefinição ser de 64 *bits* e, portanto, o tipo de dados **time_t** corresponde a um **Integer32**, ou seja, o tamanho é inferior ao de um int, o que significa que o valor da variável incorre num *overflow*, tornando a data na primeira data possível do **asctime()**.

```
1000000000, Sun Sep  9 01:46:40 2001
2147483647, Tue Jan 19 03:14:07 2038
-2147483648, Fri Dec 13 20:45:52 1901
```