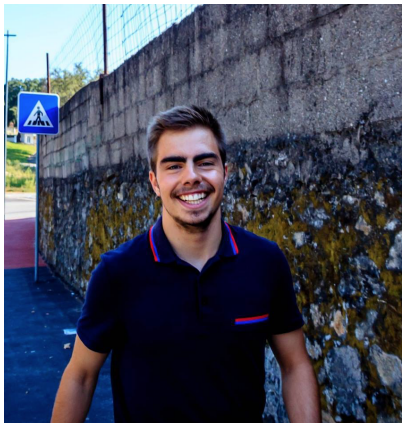
Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado em Engenharia Informática
Engenharia de Segurança

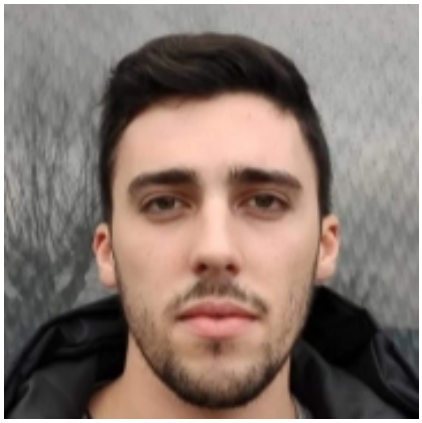
Aula 10

20 de Abril de 2020

Grupo 1



Ricardo Pereira a73577



Tiago Ramires pg41101

Braga, 27 de Abril de 2020

1. Integer Vulnerability

Question P1.1

Analisando a função *vulneravel()* somos capazes de compreender que a mesma recebe quatro parâmetros, sendo dois deles usados para fazer alocação de espaço na memória para a variável *matriz*. De seguida, percorre-se o *array matriz* e todas as posições são igualadas a *valor*. É importante que esta matriz é representada num *array* com um índice apenas, ao contrário da comum representação com dois índices.

1

Posto isto, a função *vulnerable*() aloca espaço para *matriz* de tamanho correspondente a $x * y$ e depois percorre-a. Assumindo que x é o número de linhas e y o número de colunas, a travessia da matriz é feita do seguinte modo: o primeiro ciclo fixa a linha, devendo assim multiplicar i por y para obter a linha em questão e somar a este valor j, variável do ciclo interior que itera sobre as "colunas". Contudo, os valores que as variáveis x e y podem tomar podem ser valores muito maiores que os máximos permitidos às variáveis i e j. Assim, caso x e y sejam demasiado grande, o programa corre o risco de sofrer *integer* overflow para as variáveis i e j, visto que o seu valor máximo pode ser ultrapassado. Relembremo-nos que numa arquitetura *64 bits* o número de *bytes* para uma variável *size_t* é oito e para uma variável *int* quatro *bytes*.

2

Invocando a função *vulneravel* da seguinte forma na *main*, é possível experienciar a vulnerabilidade em questão.

```
int main() {
        char *matriz;
        vulneravel(matriz, 5000000000, 5000000000, 0);
}
```

3

A execução da função dá origem a um *segmentation fault*.

Question P1.2

1

A vulnerabilidade que existe na função *vulneravel* é semelhante à anterior mas agora denomina-se *underflow*. Enquanto que no programa anterior as variáveis atingiam valores demasiado grandes, neste, as variáveis podem ter valores demasiado baixos. Imaginando que a variável *tamanho* tem o valor *0*, o limite superior é verificado e cumprido, contudo o inferior não é verificado e também não é cumprido, incorrendo-se num *underflow*.

2

```
int main() {
        vulneravel("ola", 0);
}
```

3

A função *main*, com a invocação em questão leva o programa a incorrer num *segmentation fault*.

4

A alteração em questão previne que as variáveis em questão adquiram valores inferiores a *0* evitando assim *segmentation faults*.

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

int main() {
        vulneravel("ola", 0);
}
```