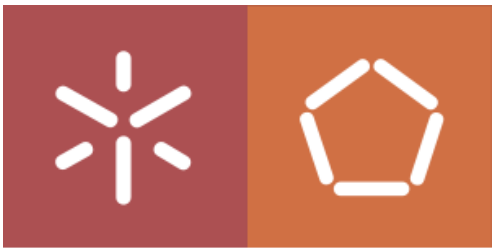

Universidade do Minho
Escola de Engenharia

Segurança de Sistemas Informáticos

Ficha de Exercícios 3 – Information Security

Grupo 04

2022/2023

Esta ficha de exercícios tem por objetivo principal analisar e corrigir erros de programas escritos na linguagem de programação C, usando como guia de erros o Top 25 CWE (Common Weakness Enumeration).

A91697 - Luís Filipe Fernandes Vilas

A91671 - João Manuel Novais da Silva

Exercício 1:

```c
#include <stdio.h>

void code_1 () {
    int arr[5] = {1, 2, 3, 4, 5};
    int index = 5;
    int value = 10;
    arr[index] = value;
}

int main() {
    code_1();
    return 0;
}
```

Neste programa rapidamente é pressentível que existe uma violação da integridade do sistema, uma vez que a linha “arr[index] = value;” está a escrever o valor de “value” numa posição da memória que não foi alocada para o intuito do programa. Por isso estamos num caso de duma falha do tipo CWE-787 Out-of-bounds Write, pois estamos a escrever informação depois do final do limite do buffer alocado, com isto é possível corromper os dados, provocar crashes ou até manipular o funcionamento do sistema para ser explorado por atacantes.

Logo para arranjarmos este problema podemos reescrever o código de maneira ou alterar o valor de “index” para 4 ou para o tamanho do array anteriormente alocado. Caso o objetivo deste código fosse acrescentar um novo elemento no final do array teríamos que realocar memória para um novo array com espaço para mais um inteiro.

```c
void code1_fix(){
    int arr[5] = {1,2,3,4,5};
    int index = 4; // or int index = sizeof(arr)/4-1;
    int value = 10;
    arr[index] = value;
}
```

Exercício 2:

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main() {
    int fd;
    char filename[] = "file.txt";

    fd = open(filename, O_CREAT);

    if (fd == -1) {
        perror("Error opening file");
        return 1;
    }
```

```c
    printf("File created successfully!\n");

    close(fd);

    return 0;
}
```

Para este caso é possível ver que existe a possibilidade da violação da Confidencialidade e Acessibilidade do sistema uma vez que a função “open()”, como descrito no manual, necessita de receber uma flag que indique as permissões, porém isto não é obrigatório, no caso de não serem fornecidas as permissões a atribuição destas é feita pelo compilador. Neste caso, usando uma máquina com Windows11 todos os utilizadores têm permissões de escrever, ler e executar o ficheiro “file.txt”. Isto poderá comprometer a segurança do utilizador pois qualquer utilizador da máquina tem acesso ao ficheiro, cujo pode conter informações importantes. Logo estamos num caso duma fraqueza do tipo CWE-276 Incorrect Default Premissions.

Para concertar este programa podemos definir as permissões:

```c
int code2_fix(){
    int fd;
    char filename[] = "file.txt";

    fd = open(filename, O_CREAT, O_RDONLY);
    if(fd == -1){
        perror("Error opening file");
        return 1;
    }

    printf("File created successfully!\n");

    close(fd);
    return 0;
}
```

Exercício 3:

```c
#include <stdio.h>

void code_3(int n) {
    int* arr;
    int i;

    arr = (int*) malloc(n * sizeof(int));

    if (arr == NULL) {
        perror("Error allocating memory");
        return;
    }

    for (i = 0; i < n; i++) {
        arr[i] = i * 2;
    }

    printf("Memory allocation and operations completed successfully!\n");

}

int main() {
    int n = 10;

    code_3(n);

    return 0;
}
```

O terceiro programa apresenta uma vulnerabilidade que pode comprometer a Integridade e Acessibilidade do sistema, devido ao facto de alocação de memória para o array “arr” ser da escolha do utilizador. Por isso, o utilizador pode introduzir um valor inteiro que seja maior do que o limite máximo da linguagem, originado assim no Integer Overflow ou num Wraparound fazendo com que o valor utilizado na função malloc não seja o que o utilizador pretendia podendo ser este um valor muito pequeno ou até negativo. Com isto é possível ver um caso em que seja alocado uma lista de tamanho pequena, mas o programa assuma que esta é grande originando em problemas Out-of-Bounds (CWE-119).

Assim, temos aqui fraquezas do tipo CWE-190 Integer Overflow or Wraparound e também CWE-20 Improper Input Validation que pode resultar em corrupção de dados, comportamentos não esperados, loops infinitos e crashes do sistema.

Para resolver este problema podemos fazer apenas uma simples verificação se o valor introduzido é demasiado grande ou negativo:

```c
void code3_fix(int n){
    int *arr;
    int i;

    if(n<=0 || n > INT_MAX/sizeof(int)){
        printf("The number inputed causes a Integer Overflow or Wraparoud!");
        return;
    }

    arr = (int*) malloc(n*sizeof(int));

    if(arr == NULL){
        perror("Error allocating memory");
        return;
    }

    for(i=0; i<n; i++){
        arr[i] = i*2;
    }

    printf("Memory allocation and operations completed successfully!\n");
}
```

Exercício 4:

```c
#include <stdio.h>

void code_4() {
    char username[20];
    char password[20];

    printf("Enter username: ");
    scanf("%s", username);
```

```c
    strcpy(password, "mypassword");

    printf("Access granted for user: %s with password: %s\n", username,
password);
}

int main() {
    code_4();

    return 0;
}
```

Finalmente, no programa 4 é possível verificar que, mais uma vez, temos a necessidade de input do utilizador, podendo originar na exploração de vulnerabilidades por um atacante. Neste caso alocamos 20 bytes de memória para a variável “username” mas depois valos dar-lhe o valor do input do utilizador, por isso, se o utilizador inserir mais do que 20 caracteres podemos correr riscos de violar a Integridade e Acessibilidade do sistema.

Temos aqui duas fraquezas do tipo CWE-787 Out-of-bounds Write e CWE-798 Hard-coded Credentials, no sentido em que a palavra-passe do utilizador é fixada no programa.

Para corrigirmos estes problemas podemos passar por apenas alocar memória às variáveis que vão guardar os dados no momento em que recebemos o input, assegurando que assim, alocamos a memória necessária para o input, e também pedir ao utilizador para introduzir a sua palavra-passe.

```c
void code4_fix(){
    char *username;
    char *password;
    size_t len=0;
    size_t read;

    printf("Enter username: ");
    read = getline(&username, &len, stdin);

    if(read == -1){
        perror("Error reading the username input!");
        return;
    }

    printf("Enter password: ");
    read = getline(&password, &len, stdin);

    if(read == -1){
        perror("Error reading the password input!");
        return;
    }

    printf("Access granted for user: %s with password: %s\n", username, password);
}
```