



# Sistemas de Software Seguros

## Segurança de Software

### 2024/2025

#### Class Project: Experiments with Buffer Overflows

### Setup for the class project:

Recall that you must do all the work within the virtual machine set up in a previous class. Go through that class description of work for instructions on configuring the virtual machine in VirtualBox.

## 1. Heap Overflow

a) Start by creating a program with the following C code and store it in a file with the name `heap_overflow.c`:

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

int main (int argc, char **argv)
{
    char *str = (char *) malloc(sizeof(char)*4);
    char *critical = (char *) malloc(sizeof(char)*9);
    char *tmp;

    printf("Address of str is [%p, %lu]\n", str, (unsigned long) str);
    printf("Address of critical is [%p, %lu]\n", critical,
        (unsigned long) critical);

    strcpy(critical, "secret");
    strcpy(str, argv[1]);
    tmp = str;
    while (tmp < critical+9){
        printf("[%p, %lu]: %c (0x%x)\n", tmp, (unsigned long) tmp,
            isprint(*tmp) ? *tmp : '?', (unsigned) (*tmp));
        tmp +=1;
    }
    printf("critical = %s\n", critical);
}
```

b) Compile the program in the following way:

```
gcc -o heap_overflow heap_overflow.c
```

c) Run the program in the following way:

```
./heap_overflow xyz
```

d) Understand the output from the above execution and relate it with the code you wrote in point a).

e) Run the program again, but now in such a way as to create an overflow that makes the `printf` of variable `critical` present the value "CIENCIAS" (without the quotes).

## 2. Stack Overflow

a) Start by writing the following C code and save it in a file with the name `stack_overflow.c`:

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

void test(char *s)
{
    char buf[10];
    strcpy(buf, s);
}

int main (int argc, char** argv)
{
    test(argv[1]);
    printf("I'm OK!\n");
}
```

b) Generate the assembly code in the following way:

```
gcc -fno-stack-protector -fno-asynchronous-unwind-tables -S stack_overflow.c
```

c) Look at the generated file (`stack_overflow.s`) and determine the number of bytes needed to create an overflow of buffer `buf` and write over something relevant (like, RBP and RIP) in the stack. Justify.

d) Confirm your result by compiling the code and executing it with the appropriate argument. Explain what has occurred.

```
gcc -fno-stack-protector -o stack_overflow stack_overflow.c
```

e) Introduce the following C code in a file `stack_overflow_2.c`:

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

void test(char *s)
{
    char buf[16];
    strcpy(buf, s);
}

void cannot()
{
    printf("This function should not be executed! Was there a BO (-; ?\n");
    exit(0);
}

int main (int argc, char** argv)
{
    printf(" &cannot = %p\n", &cannot);
    test(argv[1]);
    printf("I'm OK!\n");
}
```

f) Compile the program in the following way:

```
gcc -fno-stack-protector -o stack_2 stack_overflow_2.c
```

g) Run the program in the following way:

```
./stack_2 12345
```

h) Confirm that the program did not print the following message: This function should not be executed! ... Justify.

i) Introduce the following C code in a file `call_stack_overflow_2.c`:

```
#include <malloc.h>
#include <unistd.h>

int main()
{
    int i, A, B, C, D, E, F, G, H, I;
    char *buf = (char *) malloc(1000);
    char **arr = (char **) malloc(10);

    A = 0;
    B = 0;
    C = 0;
    D = 0;
    E = 0;
    F = 0;
    G = 0;
```

```

H = 0;
I = 0;
for (i=0; i<A; i++)
    buf[i] = 'A';

buf[A]    = B;
buf[A+1]  = C;
buf[A+2]  = D;
buf[A+3]  = E;
buf[A+4]  = F;
buf[A+5]  = G;
buf[A+6]  = H;
buf[A+7]  = I;

arr[0] = "./stack_2";
arr[1] = buf;
arr[2] = 0x00;
execv("./stack_2", arr);
}

```

j) Substitute the values of A, B, C, D, E, F, G, H, I in the above code so that the program prints the message `This function should not be executed!...` Justify the values that you selected (Help: analyze like in question c)

## Delivery of the Report

The output of the class project is a report answering all the questions and including the justifications for the responses. Each group should deliver the report either by submitting it in the course Moodle page or, if there is some difficulty with this method, by emailing it to the TP professor. The file type should be a pdf. **NOTE: only one element of the group needs to deliver the report!**

**Deadline:** 21 October 2024 (there will be no extensions)

```

=====
The following exercise is optional!
=====

```

## Challenge exercise

A server on the internet is vulnerable because it has a flaw in how it manages and accesses memory. Fortunately, the code implementing the server has been leaked, together with some client code that can interact with the server. I captured both codes and made them available on the course website, with the names `serv-challenge.c` and `client-challenge.c`.

You aim to collect in the client the value of a flag stored in the server, i.e., exploit the vulnerability to perform an information leak. The relevant server flag is **£14g**. How can you achieve this by only calling the server through the offered interface?

Some hints:

- 1) You cannot modify the server code, but you can change the client code as you like
- 2) Look through the server code and understand very well how each operation is implemented
- 3) What kind of vulnerability might the server have? It manages memory with malloc and free ...
- 4) Remember that addresses have 8 bytes in an x86-64 architecture, and that information is stored in memory in a little-endian format
- 5) Probably, the vulnerability cannot be exploited by executing a single server operation. So, you will have to find an appropriate schedule of operations, with the correct inputs, to be able to collect your prize ... flag **£14g**

Good luck!