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);
}
```

alocar memória para 4 bytes, str pointer
4 9 4, critical pointer
endereço endereço em int
7
3
aponta para o mesmo endereço

d) começa a ler do str (1º end. de mem.)
imprime todos os endereços de memória, como conteúdo se existir senão "?"
até ao fim de critical (último end. de mem.


Address of str is [0x555555756260, 93824994337376]
Address of critical is [0x555555756280, 93824994337408]
[0x555555756260, 93824994337376]: x (0x78)
[0x555555756261, 93824994337377]: y (0x79)
[0x555555756262, 93824994337378]: z (0x7a)
[0x555555756263, 93824994337379]: ? (0x0)
str
[0x555555756264, 93824994337380]: ? (0x0)
[0x555555756265, 93824994337381]: ? (0x0)
[0x555555756266, 93824994337382]: ? (0x0)
[0x555555756267, 93824994337383]: ? (0x0)
[0x555555756268, 93824994337384]: ? (0x0)
[0x555555756269, 93824994337385]: ? (0x0)
[0x55555575626a, 93824994337386]: ? (0x0)
[0x55555575626b, 93824994337387]: ? (0x0)
[0x55555575626c, 93824994337388]: ? (0x0)
[0x55555575626d, 93824994337389]: ? (0x0)
[0x55555575626e, 93824994337390]: ? (0x0)
[0x55555575626f, 93824994337391]: ? (0x0)
[0x555555756270, 93824994337392]: ? (0x0)
[0x555555756271, 93824994337393]: ? (0x0)
[0x555555756272, 93824994337394]: ? (0x0)
[0x555555756273, 93824994337395]: ? (0x0)
[0x555555756274, 93824994337396]: ? (0x0)
[0x555555756275, 93824994337397]: ? (0x0)
[0x555555756276, 93824994337398]: ? (0x0)
[0x555555756277, 93824994337399]: ? (0x0)
[0x555555756278, 93824994337400]: ! (0x21)
→ random char
[0x555555756279, 93824994337401]: ? (0x0)
[0x55555575627a, 93824994337402]: ? (0x0)
[0x55555575627b, 93824994337403]: ? (0x0)
[0x55555575627c, 93824994337404]: ? (0x0)
[0x55555575627d, 93824994337405]: ? (0x0)
[0x55555575627e, 93824994337406]: ? (0x0)
[0x55555575627f, 93824994337407]: ? (0x0)
[0x555555756280, 93824994337408]: s (0x73)
[0x555555756281, 93824994337409]: e (0x65)
[0x555555756282, 93824994337410]: c (0x63)
[0x555555756283, 93824994337411]: r (0x72)
[0x555555756284, 93824994337412]: e (0x65)
[0x555555756285, 93824994337413]: t (0x74)
[0x555555756286, 93824994337414]: ? (0x0)
[0x555555756287, 93824994337415]: ? (0x0)
[0x555555756288, 93824994337416]: ? (0x0)
secret
critical = secret

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
```

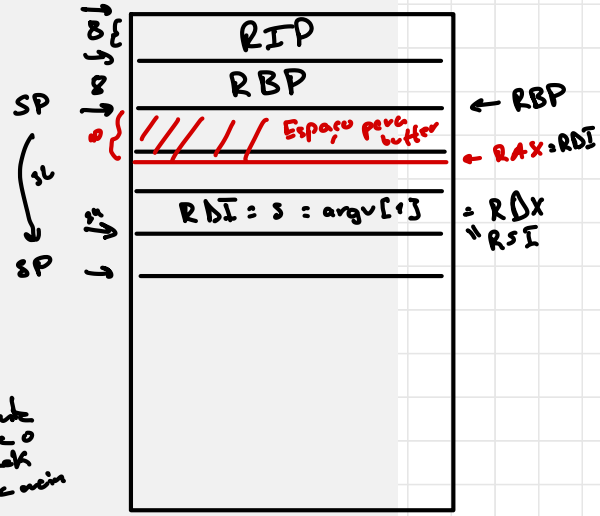
c) Look at the generated file (`stack_overflow.s`) and determine the number of bytes needed to create an overflow of buffer `buf` and write over something relevant (like, RBP and RIP) in the stack. Justify. 10 chars para RBP errado e 16 para RIP errado
+10 → 16 bytes

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
```

se 10 corrompe RBP e quando voltar para o stack frame da main vai voltar para outra instrução que não a certa.

```
        .file   "stack_overflow.c"
        .text
        .globl  test
        .type   test, @function
test:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $32, %rsp
        movq    %rdi, -24(%rbp)
        movq    -24(%rbp), %rdx
        leaq    -10(%rbp), %rax
        movq    %rdx, %rsi
        movq    %rax, %rdi
        call    strcpy@PLT
        nop
        leave
        ret
        .size   test, .-test
        .section        .rodata
.LC0:
        .string "I'm OK!"
        .text
        .globl  main
        .type   main, @function
main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        movq    -16(%rbp), %rax
        addq    $8, %rax
        movq    (%rax), %rax
        movq    %rax, %rdi
        call    test
        leaq    .LC0(%rip), %rdi
        call    puts@PLT
        movl    $0, %eax
        leave
        ret
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0"
        .section        .note.GNU-stack,"",@progbits
```



prepara os args para strcpy

SP para RBP e pop RBP aponta para o stack de main

para o RIP voltar para a main e fazer a próxima instrução

→ pushq rip

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 **`f14g`**. 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 **`f14g`**

Good luck!