CS 577 Cybersecurity Lab

Lab 3 – due 9/25/14 6:16pm

Georgios Portokalidis – Stevens Institute of Technology

This assignment will familiarize you with how programs are represented in machine language, how stack frames are structured in memory, and how buffer overflow attacks work.

NOTE: Conduct the attacks using: a 32-bit Ubuntu 9.11 with the Linux kernel v2.6.28

A VMware image can be found in the following address:
http://128.230.208.57/SEEDUbuntu12.04.zip

Initial setup:

Ubuntu and other Linux distributions have implemented several security mechanisms to make buffer-overflow attacks difficult. We will disable them first.

Address Space Randomization. Ubuntu, and most modern operating systems, use address space randomization to randomize the starting address of heap and stack. This makes guessing the exact addresses difficult. In this lab, we disable these features using the following commands:

```
#sudo sysctl -w kernel.randomize_va_space=0
```

(You may be asked to enter your password)

The StackGuard Protection Scheme. The GCC compiler implements a security mechanism called "Stack Guard" to prevent buffer overflows. We will disable this protection for this lab by using -fno-stack-protector when compiling with GCC. For example, to compile example.c:

```
$ gcc -fno-stack-protector –o  example.c1
```

Non-Executable Stack. To execute shellcode in the stack, we need to make the stack executable, as this is no longer the default:

```
$ gcc -z execstack -fno-stack-protector –o example example.c
```

Exercise 1. Smash the Heap (20%)

Smash the Heap and be the winner!

Instructions

Keep in mind that the offset values may be different on different systems. This has to do with the internals of the various memory allocators.

Provide complete documentation with all the steps you did and the source code you wrote to smash the Heap.

Exercise 2. Smash Global Memory (20%)

Smash the magic variable and be the winner!

Instructions

The names of the global variables affect their ordering in the BSS segment and thus care must be taken to choose variable names that put the target variable after the unchecked buffer. Alternatively, the `static` storage class can be used for accurate placement.

Provide complete documentation with all the steps you did and the source code you wrote to complete this exercice.

Exercise 3. Unbelievable (40%)

a) Jump to: printf("I don't believe it!\n");

Write a program that produces a file, as short and simple as possible, that redirects control flow to the above statement, which should complete successfully and print the given string.

a) Call unbelievable with the argument 0xdeadbeef

Write a program that produces a file, as short and simple as possible, that causes the program to call the unbelievable function with the argument 0xdeadbeef. Explain its principles of operation in a few sentences as a comment within your program.

Tip! You have to construct your own call of unbelievable with the argument 0xdeadbeef, save it in the buffer, and "return" inside the buffer.

Provide a complete documentation with all the steps you did and the source code you wrote.

Exercise 4. Change your marks (20%)

Write a program that produces a file named data10plus, as short and simple as possible, that causes the *hello* program to print your name and recommend a grade of "10". Explain its principles of operation in a few sentences as a comment within your program.

Tip! You have to construct your own printf, save it in the buffer, and "return" inside the buffer.

Provide a complete documentation with all the steps you did and the source code you wrote.

Grading

As always, we will grade your work on quality from both the user's and programmer's points of view. Each program should contain function-level and local comments as appropriate, as well as an explanation of the program's principles of operation. **PLEASE SUBMIT**: ***Documentation***, ***all source code with comments***, ***and a written report***.