Development of Secure Software: Project Peport

Matteo Di Pirro, r0691787

November 27, 2017

1 Overview

Level	Password	Time
casper4	Hp6S6yTWXLvCk6J98WD5M6REKUfbqS0X	4 hours
casper40	SpuH8rft4qWqzdLAzpb9fSNb215gecva	1 hour
casper6	pxnTK4pqGQoRhDCwKblrK7M8YtRndu8U	1 hour
casper60	qWBWotuzvUy3JQXmCk9LwvIVbUvoCCj6	10 minutes

Table 1: Project summary

2 Stack-Based Buffer Overflow

2.1 Casper4

2.1.1 Description and vulnerability

Casper4 requires the user to provide a command-line parameter. If this parameter is not provided, it shows an error message and exits with code 1. Otherwise, it calls the `greetUser` function. `greetUser` declares a 666-characters long buffer and copies the user-provided parameter in it. Afterwards, it prints *Hello* followed by the buffer's content.

Casper4 has an immediate vulnerability since it uses the unsafe C's `strcpy` function. It does not check the length of string to be copied to make sure it is not bigger than the buffer's size. This makes a buffer overflow attack possible. One can simply provide a parameter longer than 666 characters and overwrite the buffer (until the return address), potentially injecting code to be executed after the function's return. This is possible since there are no stack canaries, which detect a "naive" overflow, and the stack is executable. With a non-executable stack this simple version would not be feasible, and a more advanced exploit would be required (see Section 4).

2.1.2 Exploit

The exploit for Casper4 is simple. As aforementioned, it works as follows:

1. Inserts in the buffer some no-operation (NOP) bytes (`x90`);

2. Injects the shell-code corresponding to `/bin/xh` (which length is 21 bytes);
3. Overwrites the portion of the function's stack frame between the end of the buffer and the return address;
4. Overwrites the return address making it point to a generic address within the buffer (but prior to the shell-code beginning).

3 Heap-Based Buffer Overflow

4 Stack-Based Buffer Overflow with Non-Executable Stack