Lab 2 - Buffer Overflow Attacks

Objective

The objective of this lab is to familiarize you with Buffer Overflow attacks. You will exploit a buffer overflow vulnerability to compromise and gain `root` access on your Linux VM (we will assume that you don't already have `sudo` privileges). Buffer Overflows have been around for several decades, and they are still very common according to the CVE database. As these vulnerabilities can alter a program's flow / give root access to a system, their impact can be potentially devastating for an organization.

This lab requires some basic knowledge of C programming and debugging.

Background

A buffer overflow vulnerability in a program allows an attacker to write data beyond the pre-allocated memory space of a buffer. Each program's stack contains a mix of program data (e.g., variables / buffers) and program controls (e.g., return address). In case of a buffer overflow, the attacker fills the variable / buffer beyond the allocated memory space to overflow the program control sections of the stack. Typically, it entails overwriting the return address to point it to some malicious code. As such, this vulnerability can be used by an attacker to alter the control flow of the program in an undesirable way i.e., execute arbitrary piece of code or commands.

There are various online resources that discuss this vulnerability. This video shows a simple demonstration of exploiting a buffer overflow vulnerability. Note that the overflow in this lab is fairly simple to exploit and can be implemented with knowledge gained from the video, however, the extra credit requires a deeper understanding of buffer overflows and locations for placing the shellcode. The paper Smashing The Stack For Fun And Profit is an excellent resource for understanding how to solve the extra credit part. We recommend reading this paper to get a really good understanding of overflows. The paper starts with describing in detail how variables, buffers, and return addresses are stored on the stack. It then describes how to find and exploit a buffer overflow vulnerability. The focus of the paper is 32 bit processors, but the same techniques apply to 64 bit processors as well. Also, read about gdb here as you would be using it for debugging the program flow during the attack.

Setup

Login to your Linux VM, open a terminal and run the command `sudo setup_overflow_lab`. This shell script sets up an *insecure* environment for the lab, in order to make the exploit easier. The script performs the following steps -

1. `Address Space Layout Randomization (ASLR)` is first disabled on the Linux VM. `ASLR` is a protection that randomizes the program's stack to make guessing addresses of program data and program controls really hard. This protection makes the attack very difficult, however, the attack is still possible.
2. The `root_sh` binary is compiled using `gcc` flags that disable canary values injection on the stack. Canaries are a technique used by compilers to detect buffer overflow attacks. They work by injecting random values in the program space, and detecting whether these values change during execution.
3. The `root_sh` binary file owner is changed to `root` and the file permission is changed to set the `suid` bit. This means that the program can elevate to `root` privileges even when a regular user executes the program. This is standard practice on the Linux ecosystem, e.g., the `passwd` command is also a `suid` program owned by `root`.

Note that ASLR is re-enabled when the VM is restarted, so ensure that you run `sudo setup_overflow_lab` after every restart before performing this lab.

Part 1 - Simple Overflow Attack

1. In this part, let's focus on the program `root_sh` located inside `/usr/local/bin/`. This program accepts a username and password as command line arguments, and executes a shell with the owner's UID upon successful authentication. The username and password are unknown. Try different values of these

arguments to analyze the behavior of the program. Your objective would be to overflow the buffer to authenticate successfully without knowledge of the user credentials. This will give you `root` access to the VM.

2. Read the `root_sh.c` source file located inside `/usr/local/src/labs/overflow/` very carefully and make sure you understand how it works. *Pay close attention to the `check_admin_password` function's arguments and the return type, and its usage by the `main` function.* Think carefully about which buffer(s) can be overflown to force a successful authentication. Now, devise an attack that overflows the buffer(s) to grant you shell access. *To simplify the overflow, use `gdb` to analyze the stack and determine the layout of the buffers.*

3. Upon successful exploitation of `root_sh`, run `whoami` in the resulting shell and take a screenshot of the output for your report. If your overflow was successful, you should have obtained `root` access on the VM. Also record the exploit code / commands to include in your report.

Lab Report

For this lab, each student must submit a report with the following information:

1. Submit screenshots, exploit code / commands, and a brief explanation of the overflow vulnerability to demonstrate that you understand the attack. Make sure that the screenshots show the output of `whoami` command as `root`.

2. Research and document any two operating system, compiler or developer protection schemes for stack-based buffer overflow attacks. For both these schemes, also mention their benefits and limitations.

Grading

- 80 points – Successful exploitation of the buffer overflow vulnerability in Part 1
- 20 points – Answer to remaining question

Optional Extra Credit - Tricky Overflow Attack

Grading - 2.5% added to the final grade

1. In this part, let's focus on `extra.c` that you can download from Blackboard. This program is very similar to `root_sh.c` except that it does not execute a shell upon successful authentication. This means that the previous overflow may not work with this program. Your objective in this part is to devise an attack that injects a `shellcode` someplace on the stack, and then overflows the buffer to execute this `shellcode`. You can use the following `shellcode` which is tested to work on Ubuntu 18.04.

```
\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52
\x57\x54\x5e\xb0\x3b\x0f\x05
```

2. As an extra step in this part, update the `setup_overflow_lab` shell script to compile `extra.c` as a `root` owned `suid` binary file called `extra` located inside `/usr/local/bin/`. Make sure that you use the same `gcc` flags that were used for `root_sh.c`, otherwise the exploit will become very difficult.

3. Upon successful exploitation of `extra`, run `whoami` in the obtained shell and take a screenshot of the output for your report. If your overflow was successful, you should have obtained `root` access on the VM. Also record the exploit code / commands to include in your report.