CMPT 479/980:
Systems and Network Security
Spring 2020

Assignment 1

Shellcode and Buffer Overflow

Due date: February 6, 2020

Instructor: Khaled Diab

The goal of the assignment is to:

(a) Reproduce a buffer overflow attack
(b) Defeat a countermeasure used in a major shell software
(c) Get familiar with using different system calls

Never use any of the provided code on a network connected to the Internet.

1. Prerequisites

(a) Disable address space randomization (Task 1 and Task 2)

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

(b) Link zsh to sh (Task 2)

```
$ sudo ln -sf /bin/zsh /bin/sh
```

(c) Running the vulnerable C program (Task 1 and Task 2)

```
$ gcc -o vuln -z execstack -fno-stack-protector vuln.c
$ sudo chown root vuln
$ sudo chmod 4755 vuln
```

(d) Running assembly code (Task 3)

```
$ nasm -f elf create_file.asm
$ ld -o crete_file create_file.o
```

2. Tasks

Task 1: Exploiting a buffer overflow vulnerability (50%)

The provided `vuln.c` program has a buffer overflow vulnerability. The program reads a text file called `shellcode` and copies its content to a buffer that we plan to overflow (`buffer` in function `bof`).

```
int bof(char *str)
{
    char buffer[BUF_SIZE];
    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);
    return 1;
}
```

Your **task** is to generate the contents of the `shellcode` file to exploit the buffer overflow vulnerability. In particular, your shellcode needs to spawn a new shell (i.e., `/bin/sh`) during the normal flow of the program.

(1) You should set the `BUF_SIZE` to be the least significant two digits in your SFU ID. If these two digits are zeros, choose the next significant two digits. Different numbers will result in different stack layouts. E.g., if your SFU ID is 400508678, the buffer size should be 78. If your SFU ID is 400508600, the buffer size should be 86.

(2) We provided a simple Python script (`gen_shellcode.py`) that generates the shellcode.

(3) Create a copy of `gen_shellcode.py` and name it `gen_shellcode_task_1.py`. Then, implement the three missing parts in the new file:

  (a) How the `shellcode` file is initialized

  (b) the offset value `offset`, and

  (c) the return address at `content[offset+0]...content[offset+3]`.

Questions

(1.a) List the tools you used to reproduce the buffer overflow vulnerability.

(1.b) What is the effect of `BUF_SIZE` on your solution?

(1.c) Explain (with snapshots) the steps you made to calculate the offset value `offset` and the return address.

(1.d) If the address space randomization is enabled, suggest a strategy to exploit the buffer overflow vulnerability in the program.

Task 2: Enable `setuid` for Task 1 (20%)

Your **task** is to implement the `setuid(0)` system call in the shellcode you generated in Task 1.
To do so, create a copy of `gen_shellcode_task_1.py` and name it `gen_shellcode_task_2.py`.
Then, modify the variable shellcode to implement the `setuid` system call.

To test your shellcode, do the following:

```
$ sudo ln -sf /bin/dash /bin/sh
$ ./vuln
#    <<< you should see the "#" prompt
```

Questions

(1.a) Did you need to modify the offset value `offset` and the return address? Why?

Task 3. Write a shellcode to perform file operations (30%)

Your **task** is to write a shellcode that creates a new file, writes a single line to the file, and closes the file. To do, you will use three file-related system calls: `open`, `write`, and `close`.
You need to modify the provided file `create_file.asm` to complete this task. The file has six missing parts:

- Task 3.1: Init eax, ecx, edx
- Task 3.2: Set the file name to be "sfusecXX" where XX are the two digits you used in Task 1
- Task 3.3: Set `ebx` with the right value while opening a file
- Task 3.4: Set `ebx` to contain the file descriptor of the new file
- Task 3.5: Set `edx` to contain the message len
- Task 3.6: call `close` syscall

Questions

(1.a) Suppose you have a control over a victim machine, and you have the ability to create/modify an existing file. What file would you create/modify? Why?

3. Submission

You are required to submit:
(1) the completed three files:

`gen_shellcode_task_1.py`,
`gen_shellcode_task_2.py`, and
`create_file.asm`.

(2) a PDF document answering the questions from Section 2.

The files should be compressed in a single (.zip) archive. The code should compile and run without any errors.

4. Policy

- Late submissions will not be graded.
- Make sure that your code is well-organized with sufficient comments.
- Any form of cheating will not be tolerated. Particularly, copying code from other students or from other sources such as the Web.
- You can discuss the assignment with other students. However, the actual coding must be your own.

5. Environment Setup

You need to use a virtual machine (VM) to complete this assignment. We created a VM with a preinstalled Ubuntu 16.04 (32-bit) and the required software to execute the code. You can create your own VM. However, it is your responsibility to ensure that your submitted code works on *our* VM.

We will run your submitted code using the given VM.

Virtual Machine

If you are not familiar with VMs, refer to online articles about virtualization (e.g., https://en.wikipedia.org/wiki/Virtual_machine). In summary, virtualization allows you to run a guest OS on top of your host OS in isolation. Virtualization needs two main components (when it comes to this assignment): (1) Hypervisor, which is the software that allows you to run the guest OS, and (2) VM Image, which contains the OS and installed packages.

Setup. We recommend using VirtualBox (https://www.virtualbox.org/) as the hypervisor. It is a free software and easy to install and use. For the VM Image, we prepared an Ubuntu-based image with the required dependencies to compile and run the code. The setup has three simple steps:

1. Download and install VirtualBox

2. (Optional) Install VirtualBox Guest Additions: https://www.techjunkie.com/ova-virtualbox/
3. Download the VM Image: https://vault.sfu.ca/index.php/s/pq2sVjmUlmfBWwl
4. Import the VM Image: https://techantidote.com/how-to-import-ova-file-into-virtualbox/

The Image is based on Ubuntu 16.04 LTS (32-bit), which requires 2 GHz dual core processor or better, 4 GB system memory, and 25 GB of free hard drive space.

Login. After importing the provided VM Image, you can login to the guest OS using these credentials:
Username: `sfu`
Password: `ufs`
Note that this user has administrative rights.

Starter Code. To work with the code, you should clone the following repository:

```
git clone https://github.com/netsys-security/assignment1.git
```