CS155: Computer Security **Spring 2016**

Project #1

Due: Part 1: Thursday, April 7 - 11:59pm, Part 2: Thursday, April 14 - 11:59pm.

The goal of this assignment us to gain hands-on experience in finding vulnerabilities in code and mounting buffer overflow attacks. You are given the source code for six exploitable programs which are to be installed with `setuid` root in a virtual machine we provide. You'll have to identify a vulnerability (buffer overflow, double free, format string vulnerability, etc) in each program. You'll write an exploit for each that executes the vulnerable program with crafted argument, causing it to jump to a exploit string. In each instance, the result will yield a root shell even though the attack was run by an unprivileged user.

The Environment

You'll run your exploits in a virtual machine (VM) provided for the assignment. This serves two purposes. First, the vulnerable programs contain real, exploitable vulnerabilities and we strongly advise against installing them with `setuid` root on your machine. Second, everything from the particular compiler version, to the operating system and installed library versions will affect the exact location of code on the stack. The VM provides an identical environment to the one in which the assignment will be tested for grading.

The VM is configured with Ubuntu Linux 14.04 LTS, with ASLR (address randomization) turned off. It has two users, "root" and "user", both with the password "cs155". The exploits will be run as "user" and should yield a command line shell (`/bin/sh`) running as "root". The VM comes with a set of tools pre-installed (curl, wget, openssh, gcc, vim etc), but feel free to install additional software. For example, to install the emacs editor, you can run as root:

```
$ apt-get install emacs
```

When you first run the VM, it will have an OpenSSH server running so you can login from your host machine as well as transfer files using, e.g., `ssh` and `scp`. If you login into the VM, it will show you a welcome message that will include the `ssh` command to use to get in.

Targets and Skeleton code

The `targets/` directory in the assignment tarball (note that this has already been copied to the VM for you) contains the source code for the vulnerable targets as well as a Makefile for building and installing them on the VM. Specifically, to install the target programs, as the non-root "user":

```
$ cd targets
$ make
$ sudo make install
```

This will compile all of the target programs, set the executable stack flag on each of the resulting executables, and install them with `setuid` root in `/tmp`.

Your exploits **must** assume that the target programs are installed in `/tmp/` such as `/tmp/target1`, `/tmp/target2`, etc.

Exploit Skeleton Code

The `sploits/` directory in the assignment tarball contains skeleton code for the exploits you'll write, named `sploit1.c`, `sploit2.c`, etc, to correspond with the targets. Also included is the header file `shellcode.h`, which provides Aleph One's shellcode in the static variable `static char* shellcode`.

Deliverables

The assignment is divided into two parts:

- Part 1 (due on April 7th 11:59pm) consists of targets 1 and 2.
- Part 2 consists of the other four targets.

In each part, you'll need to submit a gzipped tarball (`.tar.gz`) with **no directory structure**, containing the exploits and a Makefile to build them. Running `make` with no arguments from your extrated submission tarball should yield `sploit1` through `sploit6` executables in the same directory. Finally, the tarball must also include a file `ID.csv` which conatins a comma-separated line for each group member with your SUID number, Leland username, last name, first name (order matters). The `sploits/` directory already contains such a file, so you just need to modify it.

Assuming you use the skeleton code (which is highly recommended), you can create such a tarball by running `make submission` for the top level directory of the assignment source. This command will output a file `submission.tar.gz` which you can submit.

NOTE: *Due to the size of the class, the correctness of your submission will be graded primarily by script. As a result, following the the submission format is important. We really, really want to give you full credit! Help us help you!*

Instructions for submitting the tarball will be posted on Piazza.

Extra Credit

The extra credit for this assignment is quite different from the rest of the assignment. Most significantly, you'll mount an exploit on a binary that has stack canaries turned on (specifically, GCC's stack protector). Because of the nature of exploits on canaries, you'll be attacking the program over the network rather than by invoking it through an call to `exec`. This will also require different shell code.

The Vulnerable Program

In the `targets/` directory, `extra-credit.c` is a forking network echo server. It listens for TCP connections on port 5555, for each connection it forks and, in the child process, reads in lines (separated by a carriage return and newline characters, `\r\n`) from the client, echoes them back until it reads an empty line. You're goal is to construct a payload to send over the network that

redirects execution of the child process handling your connection to `unlink` the file `/tmp/passwd`. To achieve this you'll have to do two things:

1. Write new shell code that `unlink`s the file `/tmp/passwd` instead of `exec`ing `/bin/sh`. We've provided a file `shellcode.S` with the latter and a Makefile target (`shellcode.bin`) that compiles the shellcode into binary format such that it can be used in your exploit.

2. In the sploits directory, we've included a python script `extra-credit.py` that can serve as skeleton code for your exploit. It reads in `shellcode.bin` (which is compiled from `shellcode.S`), as well as connects to the vulnerable server. Modify `extra-credit.py` to mount your exploit. Note that the function `try_exploit` takes an exploit string and the connection socket, sends the exploit string across and returns `True` if the connection died and `False` if it is still alive (how can you use this signal?).

Extra Credit Deliverables

For the extra credit, we should include two files in your `submission.tar.gz`:

1. A text file `extra-credit.txt` with no more than 500 words answering the questions:

   - How does your exploit work?
   - What would you modify GCC's stack canaries mechanism, or Linux's fork syscall to protect against your attack?

2. The modified file `extra-credit.py`.

Hints

- Suppose you overflowed a buffer such that only a single byte of the canary is overwritten. If the process crashes and the connection is closed, what does that tell you? Conversely, if it doesn't crash, what does that tell you?

- The syscall number for `unlink` is 0x0a (10 in decimal), which is also the newline character in ASCII. Does this matter? How might you get around this?

- You can debug a running process with `gdb` by passing the `-p` argument with process ID of the target process. Note that each child process (the result of fork will have a different process ID).

- We strongly encourage you to use a NOP slide for this problem, even if you are able to find an exact address for the shellcode. In previous years we've noticed some variability in the exact position of things in memory that resulted in some student's solutions failing in the grading VM. Using a NOP slide will help prevent this problem.

Late Policy

The general course policy on late submissions applies to this assignment. The policy is details in the course overview on the website: `http://crypto.stanford.edu/cs155/info.html`

Setup: Set-by-step

Download the VM from `http://crypto.stanford.edu/cs155/hw_and_proj/proj1/vm-cs155.tar.gz` and extract. The tarball contains a folder, `vm-cs155`, which contains a virtual hard disk (`vm-cs155.vmdk`) and a VMWare virtual machine image specification file (`vm-cs155.vmx`). This allows the machine to be run using most virtualization software.

Run the machine using Qemu, VirtualBox or VMWare Player/Fusion. We **strongly recommend** VMWare Player or VMWare Fusion for their ease of use. You can obtain a free copy of VMWare Fusion from `http://software.stanford.edu` if on OS X, or you can download VMWare Player from `https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/7_0` if on Windows or Linux.

To run with VMWare Player/Fusion, open the file `vm-cs155.vmx` in VMWare Player/Fusion. Note that you might need to change the network adapter to be "NAT" if you experience network connection issues.

Once in the VM, login with username "user" and password "cs155". Your home directory will now contain the folder "proj1" which contains the targets and starter code for the project. Note that you will also be greeted with a message telling you the command to use if you wish to SSH into the machine from your host OS.

Build and install the targets:

```
$ cd proj1/targets
$ make && sudo make install
Password: cs155
```

Write, build and test your exploits:

```
$ cd ../sploits
...edit,test...
$ make
$ ./sploit1
```

If at any point you'd like a fresh copy of the starter code, you can download the assignment tarball from `http://crypto.stanford.edu/cs155/hw_and_proj/proj1/proj1.tar.gz` and extract it:

```
$ wget http://crypto.stanford.edu/cs155/hw_and_proj/proj1/proj1.tar.gz
$ tar xzf proj1.tar.gz
```

Then repeat the build and installation steps above.