CE441: Data and Network Security **Fall 2021**

Project #1*

Due: Parts 1 and 2: Friday, Mehr 23 - 11:59pm,
Part 3: Saturday, Aban 1 - 11:59pm.

The goal of this assignment is to gain hands-on experience finding vulnerabilities in code and mounting buffer overflow attacks. In Parts 1 and 2, 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 an exploit string. In each instance, the result will yield a root shell even though the attack was run by an unprivileged user. In Part 3, you will use a fuzzer to find a vulnerability in a program called bsdtar, part of a widely used library called libarchive. You'll download and build the vulnerable version of libarchive in your VM and then run the fuzzer to find an input that causes the program to crash.

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 16.04 LTS, with ASLR (address randomization) turned off. It has a single user account "user" with password "ce441", but you can temporarily become the root user using `sudo`. 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
```

* Acknowledgment: This project is obtained from CS155, Spring 2020, Stanford University.
Edited By H. Ahmadzadeh & A. Ehteshami.

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`. You can login to the VM from your host machine using the command:

```
$ ssh user@192.168.56.144
```

For start working, you have to clone the handouts repository to the vm.

Parts 1 and 2

Parts 1 and 2 ask you to develop exploits for six different vulnerable target programs.

Targets

The `targets/` directory in the assignment 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 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 const char*` shellcode.

Part3

In Part 3 you'll learn how to find security vulnerabilities using a fuzzer called American Fuzzy Lop. Fuzzing is a technique for finding vulnerabilities in a program by running the program on random data until it crashes. We will be using afl-fuzz, one of the most successful and widely-used fuzzers currently available. afl-fuzz has already been installed on the VM. You can read more about afl-fuzz and how it works at http://lcamtuf.coredump.cx/afl/.

Fuzzing libarchive

You will be fuzzing libarchive (https://www.libarchive.org/), a widely used archive and compression library. It provides a program called bsdtar that offers similar functionality to the more common GNU tar program. For example, you can use bsdtar to extract a `.tar.gz` file in the same way as regular tar:

```
$ bsdtar -xf <some-file>.tar.gz
```

In the `proj1/fuzz/` directory, download and extract the source code for libarchive version 3.1.2:

```
$ curl -O http://www.libarchive.org/downloads/libarchive-3.1.2.tar.gz
$ tar -xf libarchive-3.1.2.tar.gz
```

This should give you a directory called `libarchive-3.1.2/`. In that directory, run:

```
$ CC=afl-gcc ./configure --prefix=\$HOME/ce441-001-handouts/proj1/
fuzz/install
```

This will configure libarchive so that it will be built using the afl-fuzz compiler, `afl-gcc`, and so that it will install itself in the `install/` directory rather than system-wide. You can then build and install libarchive, including bsdtar:

```
$ make
$ make install
```

(Note that we are not using sudo.) After this, bsdtar should be installed under `install/bin/`.

The `fuzz/testcases/` directory contains a seed testcase that afl-fuzz will modify to try to crash bsdtar. Run afl-fuzz on this testcase by running the following command from the `fuzz/` directory:

```
$ afl-fuzz -i testcases -o results install/bin/bsdtar -O -xf @@
```

This command instructs afl-fuzz to run bsdtar and supply the arguments `-O -xf @@`. The `-O` (capital-O, not numeral-0) option tells bsdtar to not write any files to disk. afl-fuzz will replace `@@` with the name of the input to test for a crash, so the `-xf @@` part will cause bsdtar to try to extract the input file generated by afl-fuzz. The result is that afl-fuzz will generate a bunch of test cases based on the seeds in the `testcases/` directory and then run bsdtar on each generated file until bsdtar crashes.

The fuzzer may run for several minutes before finding a crash. Once it does, hit Ctrl-C to stop AFL.

Write-up

Spend a few minutes investigating the crash. Use GDB to get a backtrace at the time of the crash. Try to figure out what the vulnerability is in the source code.

In the `fuzz/README` file, include your backtrace from GDB and briefly describe the vulnerability (two or three sentences; no more than 200 words).

Deliverables

Deliverables of The assignment is divided into three parts:

- Parts 1 and 2 (due on Mehr 23 11:59pm) consists of targets six targets.
- Part 3 consists of fuzzing a real-world program (bsdtar) to find a vulnerability.

For each submission, you'll need to provide a screen recording & push all of the proj1 files to your repository in tarasht. Your repo will contain the contents of the `proj1/sploits/` directory, the crashes found during fuzzing, the README in the `proj1/fuzz/` directory, and `proj1/ID.csv`. Make sure that if you clone your repository:

1. In the `sploits/` directory, running make with no arguments should yield `sploit1` through `sploit6` executables in the same directory.
2. In the `fuzz/` directory, running the vulnerable version of bsdtar on any of the crashing testcases should reproduce the crash.
3. In the `fuzz/` directory, there should be a README file that describes the vulnerability found via fuzzing.
4. The repository must include the file `ID.csv` which conatins a comma-separated line with your SUID number, Tarasht ID, last name, first name (order matters). The top-level directory already contains such a file, so you just need to modify it.

In your screen recording, explain your solution step by step and record your voice with the screen. Your screen recording should contain all the tools you have used to solve the problems (e.g. gdb). Submit only one screen recording for each part and try to minimize the size of it (max size: 30MB). Moreover, its length should be less than 5 minutes for each exploit and less than 10 minutes for part 3.

Do Not push your screen recordings to your repository in the tarasht. Upload them to a file hosting service, e.g. Google Drive, and insert the links in `proj1/Links.txt`. Push `proj1/Links.txt` in your repository.

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!*

Setup: Set-by-step

Download the VM from `http://partov.ce.sharif.edu/assets/40441-991/CE441_vm.ova.xz` and extract. It contains a file CE441.ova (`md5sum: a3f9457567ceadbe48699f89225ce980`), which is an Open Virtualization Format archive of the virtual machine. (You also can use `this link`.)

Import the virtual machine using VirtualBox. We **strongly recommend** using VirtualBox. VirtualBox is free for Windows, macOS, and Linux, and the virtual machine was developed and tested using VirtualBox. To import the virtual machine, choose "File", "Import Appliance", and select the CE441.ova file. This will set up a new virtual machine called CE441.

Note: You might need to set network adapter 2 to use a host-only network if SSH doesn't work out of the box. On macOS and Linux, you can add a host-only network by selecting "File", "Host Network Manager", clicking the "Create" button and creating a virtual network interface named "vboxnet0". On Windows 10, the host-only network should be auto-configured when you open and close the "Settings" for the VM.

Once the VM has booted, login with username "user" and password "ce441". The VM should be configured with a host-only network adapter and static IP address 192.168.56.144, so you should also be able to log in using SSH from your host:

```
$ ssh user@192.168.56.144
```

Once logged in, clone handouts repository and change origin to your repository:

```
$ git clone https://tarasht.ce.sharif.edu/ce441-001-students/ce
441-001-handouts
$ cd ce441-001-handouts
$ git remote set-url origin https://tarasht.ce.sharif.edu/ce441-
001-students/ce441-001-"student-id"
```

build and install the targets:

```
$ cd proj1/targets
$ make && sudo make install
Password: ce441
```

Write, build and test your exploits:

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