CS 361S - Network Security and Privacy
Spring 2017

Project #2

Part 1 due: 11:00, April 3, 2017
Part 2 due: 11:00, April 10, 2017

Submission instructions

Follow the submission instructions in the **Deliverables** section.
If you are submitting late, please indicate how many late days you are using.

Collaboration policy

This assignment can be done individually or in two-person teams. Any cheating (*e.g.*, submitting another person's work as your own, or permitting your work to be copied) will automatically result in a failing grade. The Computer Science department code of conduct can be found at `https://www.cs.utexas.edu/academics/conduct`.

Late submission policy

The first part of this project is due at the **beginning of class** on **April 3**. The second part is due at the beginning of class on **April 10**. All late submissions will be subject to the following policy.

You start the semester with a credit of 3 late days. For the purpose of counting late days, a "day" is 24 hours starting at 11am on the assignment's due date. Partial days are rounded up to the next full day. You are free to divide your late days among the take-home assignments (3 homeworks and 2 projects) any way you want: submit three assignments 1 day late, submit one assignment 3 days late, *etc.* After your 3 days are used up, no late submissions will be accepted and you will automatically receive 0 points for each late assignment.

Project #2 (75 points + 25 bonus points)

The objective of this project is to give you hands-on experience with implementing buffer overflow exploits. You are given source code for five exploitable programs (`target1.c`, ..., `target5.c`). These programs are all to be installed as setuid root in the the virtual machine (VM). Your goal is to write five exploit programs (`sploit1`, ..., `sploit5`). The `sploit[i]` program will execute `target[i]`, giving it a certain input that should result in a root shell on the VM.

You will need:

- Project #2 Virtual Machine (VM) Image:

  `https://www.cs.utexas.edu/~ojensen/courses/cs361s/vms/proj2-vm.ova`

- Project #2 Files:

  `https://www.cs.utexas.edu/~ojensen/courses/cs361s/vms/proj2-files.tgz`

Getting Started

1. Download the Project #2 VM Image, and import into your VM of choice (e.g. Virtualbox, VMPlayer, etc). If you do not need Internet access within the VM, I recommend you set networking to "Host Only" for ease of use.

2. Login to the virtual machine. There are two accounts: `root/root` and `user/user`. You want to do most of your work as `user`.

3. Put the Project #2 Files onto the VM. You can do this by downloading the files tarball, and then (e.g. using `scp`) to transferring it onto the VM. Extract the files into the user's home directory.

4. Set up the executable targets: Compile the targets (run `make`) and copy the targets to `/tmp` together with the corresponding .c files (run `make install`). Set up the permissions so that the targets are owned by root, are setuid root, and the .c files are publicly readable (run `make setuid` *as root*).

5. Every time you restart the VM, you'll have to set up the targets in the VM's `/tmp` because it be wiped clean. Files in your home folder will of course persist between reboots.

The virtual machine we provide is configured with Debian Etch. Should you need any other packages to do your work (*e.g.*, emacs), you can install it with the command `apt-get` (*e.g.*, `apt-get install emacs`). You may need to edit the `/etc/apt/sources.list` file and replace `http://mirrors.kernel.org/debian` with `http://archive.debian.org/debian`.

Buffer Overflow Project

Targets

The Project #2 Files contain the source code for the targets, along with a `Makefile` specifying how they are to be built.

Your exploits should assume that the compiled target programs are installed setuid-root in `/tmp` – `/tmp/target1`, `/tmp/target2`, *etc.*

Exploits

The Project #2 Files also contain skeleton source code for the exploits which you are to write, along with a `Makefile` for building them. Also included is `shellcode.h`, which gives Aleph One's shellcode. Exploit programs are very short, so there is no need to write a lot of code.

Your assignment

You are to write one exploit per target, along with the corresponding explaination of the exploit. Each exploit, when run in the virtual machine with its target installed setuid-root in `/tmp`, should yield a root shell (`/bin/sh`). You can use `whoami` to tell if you are root or not.

Grading

There are five targets. Each successful exploit will earn you the following points:

<u>Part 1</u> (due October 27)

Target 1: 10 points

Target 2: 15 points

<u>Part 2</u> (due November 3)

Target 3: 25 points

Target 4: 25 points

Target 5 is the **bonus** target, worth 25 points. You will receive these points on top of the regular points for this assignment.

Hints

`gdb` is your best friend in this assignment. It will help you inspect the contents of memory as your target is executing and generally understand what's going on. In particular, notice the `disassemble` and `stepi` commands. You may find the `x` command useful to examine memory (and the different ways you can print the contents such as `/a /i` after `x`). The `info register` command is helpful in printing out the contents of registers such as `ebp` and `esp`. Another very useful command is `info frame`. It prints a detailed description of the selected frame.

When you run `gdb`, you will find the `-e` and `-s` command-line flags useful. For example, the command `gdb -e sploit1 -s /tmp/target1` in the VM tells `gdb` to execute `sploit1` and use the symbol file in `target1`. These flags let you trace the execution of `target1` after the sploit has forked off the *execve* process. When running `gdb` using these command-line flags, be sure to first issue `catch exec`, then `run` the program before you set any breakpoints; the command `run` naturally breaks the execution at the first *execve* call before the target is actually exec-ed, so you can set your breakpoints when `gdb` catches the *execve*. Note that if you try to set break points before entering the command `run`, you'll get a segmentation fault.

Keep in mind that it'll be easier to debug the exploits if the targets aren't setuid, and are owned by `user` (otherwise, gdb will throw permission errors). If an exploit succeeds in getting a user shell on a non-setuid target in /tmp, it should succeed in getting a root shell on that target when it is setuid. (But be sure to test that way, too, before submitting your solutions!)

If you wish, you can instrument your code with arbitrary assembly using the `__asm__()` pseudofunction.

IMPORTANT: Your code **must** run within the provided VM environment.

Warnings

Aleph One gives code that calculates addresses on the target's stack based on addresses on the exploit's stack. Addresses on the exploit's stack can change based on how the exploit is executed (working directory, arguments, environment, *etc.*). In our testing, we do not guarantee to execute your exploits as bash does.

You must therefore hard-code target stack locations in your exploits. You should **not** use a function such as `get_sp()` in the exploits you hand in.

Your exploit programs should not take any command-line arguments.

Your exploits should compile. *Any exploit that doesn't compile within the VM will receive no credit.*

AGAIN: Your code **must** run within the provided VM environment.

Deliverables

Make sure you start early.

You will submit a single tarball that contains the source code for all your exploits, along with any files (`Makefile`, `shellcode.h`) necessary for building them. All the exploits should build if the `make` command is issued. There should be no directory structure: all files in the tarball should be in its root directory. Before you submit, you should do the following:

- Modify `SUBMISSION-PART1.txt` (for Part 1) or `SUBMISSION-PART2.txt` (for Part 2). The first line should state how many (possibly `0`) late days were used. Then give the following on a single line, one line for each student:
  - Your **UT EID**, followed by a single **space**, followed by your **real name**.

  For example, the output should look like this:

  ```
  $ cat SUBMISSION-PART1.txt
  0
  bevo314 Bevo Longhorn
  as525 Adam Smith
  $
  ```

- You may also (optionally) modify `README-PART1.txt` (for Part 1) or `README-PART2.txt` (for Part 2) with comments about your experiences or suggestions for improvement.

To create the tarball for submission, issue the following command inside the `/sploits` directory:

- Part 1: `tar cvzf cs361s-proj2-part1.tgz [files to include]`
- Part 2: `tar cvzf cs361s-proj2-part2.tgz [files to include]`

You will then submit the tarball using `Canvas`. Submit `cs361s-proj2-part1.tgz` (for Part 1) or `cs361s-proj2-part2.tgz` (for Part 2).

Evaluation

You will receive full credit for each exploit accompanied by a valid explanation of how it works that yields a root shell in our testing. No partial credit will be awarded for exploits that are "close".

We may also ask you to explain to us how and why each exploit works (i.e. at office hours, on a homework, on an exam, etc.). If you work with a partner, be sure that each of you understands every exploit you turn in! Dividing up the work and working separately is *not* recommended.