CS 5431: Practicum in System Security
Homework 1

Due February 10, 2011, 10:00 pm

Know your enemy as you know yourself, and success will be assured.
—Sun Tzu

You can't make something secure if you don't know how to break it.
—Marc Weber Tobias[1]

Overview

The learning objective of this homework is for you to deepen your understanding of buffer overflows through hands-on experience. You will put what you learned in class, and in readings, into action. Your task is to develop attacks that produce a root shell by exploiting buffer overflow vulnerabilities.

You will carry out this homework inside a virtual machine running Linux. We give you the source code for five vulnerable programs (`target1`, ..., `target5`). Your goal is to write five exploit programs (`sploit1`, ..., `sploit5`). Program `sploit`$[i]$ will invoke program `target`$[i]$, giving it an input that you craft to obtain a root shell on the virtual machine.

Programs `target4` and `target5` are karma problems (see the syllabus for the definition of *karma*). You do not have to produce exploits for them. Their purpose is to provide extra challenge for those students who want it.

You must work with a partner on this homework. You may not work with any students other than your partner.

The Virtual Machine

The virtual machine (VM) is provided at `http://www.cs.cornell.edu/courses/cs5431/2011sp/hw1/vm.tar.gz`. To use this VM, you will need VMware Player, which is installed on the CSUG and MEng lab machines.[2]

[1] Proclaimed the "ultimate lock picker" in *Wired Magazine*, 17(6), June 2009.

[2] It is also freely available from the VMware website (`http://www.vmware.com/products/player/`) and runs on Linux and Windows. For Mac OS X you can download a 30-day free trial version of VMware Fusion, also available on the VMware website (`http://www.vmware.com/products/fusion/`. However, if you elect to install software on your own machine, you're on your own.

The VM is configured with Debian 4.0. It has networking support, so you can fetch files directly from the web onto the VM. You can also `ssh` into the VM from your host machine, and you can transfer files between the VM and the outside world using `scp` or `sftp`. Use the Debian package management system if you should desire any additional packages (e.g. emacs): you can install them with the command `apt-get` (e.g. `apt-get install emacs`).

The Code

The code is provided at `http://www.cs.cornell.edu/courses/cs5431/2011sp/hw1/code.tar.gz`. The `code/targets` directory in the code tarball contains the source code for the targets, along with a `Makefile` helping you to build and install them. The `code/sploits` directory contains skeletons for your exploits, along with a `Makefile` for building them. Also included there is `shellcode.h`, which contains Aleph One's shellcode.

Installation

1. Download the VM tarball (`vm.tar.gz`).
2. Decompress the VM tarball (`tar -xzvf vm.tar.gz`). Using VMware, open file `vm/box.vmx`. If VMware asks you whether you moved or copied the VM, say that you copied it. If VMware asks you to upgrade the VM, or to install VMware Tools or extensions, you should decline.
3. Start the VM and log in. There are two accounts defined in the VM: `root` with password `root`, and `user` with password `user`. While logged in as `user`, you can temporarily become `root` with the `su` command.
4. Ensure that networking is working by running `ifconfig` as `root` and checking that the `inet addr` field of `eth1` has a valid IP address. (It will probably begin with `192.168`.) Make sure you can reach the VM at this address by attempting to `ssh` into it from your host machine.
5. Download the code tarball (`code.tar.gz`) onto the VM. You can do this by downloading the tarball to your host machine, then using `scp` or `sftp` to transfer the file onto the VM. Alternatively, issue this command from the VM shell to download directly to the VM:

   ```
   box:~$ wget http://www.cs.cornell.edu/courses/cs5431/2011sp/hw1/code.tar.gz
   ```

6. As `user`, decompress the code tarball in `user`'s home directory (`/home/user`).
7. As `root`, run "`make install`" in the `code/targets` directory. This compiles the targets, copies the executables to `/tmp`, and sets the correct permissions for them. Every time you reboot the VM you will need to repeat this step, because the `/tmp` directory is erased at each boot.

Setuid. The target programs in `/tmp` are installed with the setuid attribute, meaning that the effective userid of a process executing a target becomes the owner of the file. If you follow the installation instructions above, the owner of the `/tmp/target`$[i]$ files will be `root`. Exploiting a target to open a shell will therefore open a root shell.

Assignment

You are to complete exploit `sploit`$[i]$`.c` for each target `target`$[i]$`.c`. When an exploit is executed in the VM with the corresponding target installed in `/tmp`, a shell (`/bin/sh`) should be opened. If the target is setuid `root`, that shell will be a root shell. A root shell is immediately distinguishable from a normal user's shell by the final character of the prompt, which changes from `$` to `#`. Also, the command `whoami` tells you which user you currently are. Inside a root shell, it will return `root`. Here is an example:

```
user@box:~/code/sploits$ ./sploit1
sh-3.1# whoami
root
```

You can change any of the code provided in the `sploit`$[i]$`.c` skeletons. Those skeletons assign a benign string to `args[1]` before executing the corresponding target. Your main goal is to instead set `args[1]` to an appropriately malicious string. You should not change any of the code in the `target`$[i]$`.c` programs, because we will execute your exploits against unmodified targets.

We will grade your exploits in the VM we provided. Exploits developed on other systems are unlikely to work in that VM. And other systems will likely have defenses installed that (to make your life easier) we have removed from our VM.

Begin by reading Aleph One's "Smashing the Stack for Fun and Profit" carefully.[3] This article is posted on the course website as required reading. It presents a practical application of the theory that we discussed in class. Make sure you have a good understanding of what happens to the stack and relevant registers before and after a function call. Other useful articles are posted as optional reading on the website and are worth studying.

Here are some hints on how to be successful:

- Start early. Theoretical knowledge about exploits does not quickly translate into the ability to write working exploits.
- The first target is the easiest. Each target should be increasingly more difficult to exploit.
- Come to office hours and ask questions if you're having difficulty. We're here to help.

[3] Aleph One gives code for function `get_sp()` that calculates the address of (nearly) the top of the stack. When executing an exploit, that function will not necessarily return the same value on your VM as it will on ours, because the value it returns reflects the environment variables, working directory, etc. You therefore should **not** use `get_sp()` in the exploits you submit. Instead, you should hard-code the target stack locations in your exploits.

gdb

The GNU Debugger, `gdb`, is your best friend for this homework. Although you probably learned to use this tool in your architecture class, you might need to invest some time in re-learning it. One way to do this is to use `gdb` to follow along with the examples in the online readings. There are also many `gdb` tutorials available online. Some useful `gdb` commands include the following: `run`, `continue`, `break`, `disassemble`, `next`, `stepi`, "`info frame`", "`info locals`", and "`info registers`". The `x` command is supremely useful and allows you to examine memory—for example, "`x/32x 0xbffff9b8`" displays 32 words of memory beginning at address `0xbffff9b8`.

The `gdb` you should use is located at `/home/user/bin/gdb`. Ensure that you are using this version by issuing "`which gdb`".

A useful way to run `gdb` for this homework is as follows:

`gdb -e sploit`$[i]$ `-s /tmp/target`$[i]$

This command tells `gdb` to execute `sploit`$[i]$ and load the symbol file of `target`$[i]$, thus enabling you to trace execution of `target`$[i]$ after `sploit`$[i]$ has issued the `execve` system call. When using this command, make sure to issue "`catch exec`" then `run` before you set any breakpoints in `gdb`. (Otherwise, you will get a segmentation fault.) Command `run` naturally breaks execution at the first `execve` call immediately before `target`$[i]$ is executed. Then you can safely set breakpoints within `target`$[i]$.

setuid and gdb. While debugging an exploit in `gdb`, you will need to remove the setuid attribute for the corresponding target. If you fail to do this, `gdb` will be unable to execute the target and you will get the following error message:

```
Executing new program:  /proc/<number>/exe
/proc/<number>/exe:  Permission denied.
```

After your exploit is working, you can restore the setuid attribute to get a root shell.

Run "`chmod -s /tmp/target`$[i]$" as `root` to remove the setuid attribute from file `/tmp/target`$[i]$. To restore the attribute, run "`chmod +s /tmp/target`$[i]$" as `root`.

Grading

You will be graded solely on the number of targets you successfully exploit to produce a root shell. For each target in {`target1`, `target2`, `target3`} you compromise, you get 1 point. The remaining targets (`target4` and `target5`) are karma problems for which you will not get points.

1337 Hackers

For fun, and to encourage you to start working on the homework early, the first team that exploits each target will be recognized as "`1337` hackers" and receive 1 bonus point.

Each team may receive recognition only for one target, hence may receive at most 1 bonus point. (Karma problems will work differently: anyone may receive recognition for them, regardless of previous recognition—and karma problems will not receive bonus points.) The status of each target will be displayed at

```
http://www.cs.cornell.edu/courses/cs5431/2011sp/hw1/status.php
```

We will try to keep this status page up-to-date as successful exploits arrive, but we will not update the page during the night.

If you have exploited a target marked as "available" on the status page, send the C source of your exploit as soon as possible to `clarkson@cs.cornell.edu` (Subject: CS 5431 HW 1 1337 hacker submission). Your source must include the names and netids of both team members as a comment at the top of the file, and you must use that subject exactly.

Submitting solutions in this competition does **not** eliminate the need to submit a complete solution, as described next.

Submission

1. The submission deadline is Thursday, February 10, 10:00 pm.

2. You should submit your solution through CMS. You should form a CMS group with your partner and submit your solution as that group.

3. Your submission must include a file named "ID" that contains a line for each member of your team, structured as follows:

   ```
   mrc26 Clarkson Michael
   fbs2 Schneider Fred
   ```

4. You should submit a gzipped tarball named `sploits.tar.gz` containing the source files and Makefile for building your exploits, as well as an ID file. All the exploits inside should build without error if the "`make`" command is issued. There should be no directory structure: all files in the tarball should be in its root directory. Running "`make dist`" in the `code/sploits` directory should produce such a tarball for you—but check to make sure.