

CS 378 - Network Security and Privacy

Fall 2007

Project #2

Due: 3:30pm CST, November 13, 2007

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 Sciences department code of conduct can be found at <http://www.cs.utexas.edu/users/ear/CodeOfConduct.html>

Late submission policy

This project is due at the **beginning of class** on **November 13**. All late submissions will be subject to the following policy.

You start the semester with a credit of 4 late days. For the purpose of counting late days, a "day" is 24 hours starting at 2pm 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 four assignments 1 day late, submit one assignment 4 days late, *etc.* After your 4 days are used up, no late submissions will be accepted and you will automatically receive 0 points for each late assignment.

Project (75 points)

Objective

The purpose of this project is to give you hands-on experience with implementing buffer overflow exploits. All work on this project must be done on a system called Boxes (implemented using User-Mode Linux), which is available in `/projects/cs378.shmat/boxes`

In `/projects/cs378.shmat/proj2/targets` directory, you are given the source code for seven exploitable programs (`target1.c`, ..., `target7.c`). These programs are all installed as `setuid root` in the Boxes system. You have to write seven exploit programs (`exploit1`, ..., `exploit7`). The `exploit[i]` program will execute `target[i]`, giving it a certain input that should result in a root shell on the Boxes system.

The `/projects/cs378.shmat/proj2/splaits` directory contains the skeletons for exploit programs. Exploit programs are very short, so there is no need to write a lot of code.

Read help files `README`, `BOXES-FAQ`, and `HINTS` in `/projects/cs378.shmat/proj2/`

Boxes environment

You must test your exploit programs within a system called Boxes, which is provided to you in `/projects/cs378.shmat/boxes` directory. Boxes, based on User-Mode Linux, allows you to boot a fully functional Linux system as a user process on another Linux machine. It should run on x86 GNU/Linux machines running a recent 2.4-series kernel. See `/projects/cs378.shmat/proj2/BOXES-FAQ` for more information.

We recommend that you test your exploits in a virtual machine booted with a “closedbox” kernel, lest you accidentally damage your host account. You can use the `ssh` daemons running in the image to transfer files from `openboxes` (with `hostfs` access) to `closedboxes`.

It is recommended that you develop your code on the host machine, or at least keep frequent backups. The User-Mode Linux kernel is mostly stable, but can occasionally crash.

Note that if you are keeping your `cow` file in `/tmp` on the host machine, it may be deleted without your knowledge. In this case, it is **extremely** important that you back up your code on to the host machine.

Targets

The `/projects/cs378.shmat/proj2/targets/` directory contains the source code for the targets, along with a `Makefile` specifying how they are to be built. There are **seven** targets.

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

Exploits

The `/projects/cs378.shmat/proj2/splaits/` directory contains 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.

Your assignment

You are to write exploits, one per target. Each exploit, when run in the Boxes environment with its target installed `setuid-root` in `/tmp`, should yield a root shell (`/bin/sh`). You can use `whoami` to tell whether you are root.

Hints

Read Aleph One's "Smashing the Stack for Fun and Profit" **carefully**. Read `scut`'s "Exploiting Format String Vulnerabilities." Both are linked from the reference section of the course website.

To understand what's going on, it is helpful to run code through `gdb`. In particular, notice the "disassemble" and "stepi" commands. You can instrument your code with arbitrary assembly using the `__asm__()` pseudofunction. Your code **must** run within Boxes.

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.

Deliverables

You will need to submit the source code for your exploits, along with any files (`Makefile`, `shellcode.h`) necessary for building them. You will submit your files using the `turnin` command on the UTCS system. The grader is `anand` and the homework name is `buffer`.

Along with your exploits, you must include file called `ID` which contains, on a single line, the following: your UTEID number(s); your UTCS username(s); and your name(s), in the format last name, comma, first name. An example:

```
$ cat ./ID
3133757 binky Clown, Binky The
3133758 mickey Mouse, Mickey
$
```

You may want to include a `README` file with comments about your experiences or suggestions for improvement.