Advanced Computer Security **Summer 2006**

Project #1

Goal

1. The goal of this assignment is to gain hands-on experience with the effect of buffer overflow bugs. All work in this project is done on a system called boxes (implemented using User-Mode Linux).

2. You are given the source code for two exploitable programs (`/tmp/target1` and `/tmp/target2`). These programs are installed as setuid root in the boxes system. Your goal is to write two exploit programs (`sploit1` and `sploit2`). Program `sploit[i]` will execute program `/tmp/target[i]` giving it certain input that should result in a root shell on the boxes system.

3. The skeletons for `sploit1` and `sploit2` are provided in the `sploit/` directory. Note that the exploit programs are very short, so there is no need to write a lot of code here.

The Targets

1. The targets/ directory in the assignment tarball contains the source code for the targets, along with a Makefile specifying how they are to be built.

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

The Exploits

The sploits/ directory in the assignment tarball contains skeleton source 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.

The 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).

gdb is your best friend in this assignment, particularly to understand what's going on. Specifically, note 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`.

A useful command to run gdb is to use the `-e` and `-s` command line flags; for example, the command '`gdb -e sploit3 -s /tmp/target3`' in boxes tells gdb to execute sploit3 and use the symbol file in target3. These flags let you trace the execution of the target3 after the sploit has forked off the execve process. When running gdb using these command line flags, be sure to 'run'

the program before you set any breakpoints; for our purposes, entering the command 'run' naturally breaks the execution at the first `SIGTRACE` before the target is actually exec-ed, so you can set your breakpoints when gdb catches the `SIGTRACE`. Note that if you try to set break points before entering the command 'run', you'll get a segmentation fault.

How to set up the Environment

1. You need to set up a Xwindows server on your machine. The two programs that you need to run are "Start→Hummingbird Connectivity 2006→Exceed" and "Start→Hummingbird Connectivity 2006→Exceed Tools→Xstart".

2. When Xstart runs, change the "method" to "Local Application", and in the command text box, type "vsh myth.stanford.edu". Click on the green run button (second last button). Enter your leland password when the prompt appears.

3. Once a terminal window opens, it asks for the terminal type. Type in "vt100". Once you get a command line, type "xterm &" and another terminal will appear; this new terminal is the one that you'll want to use from now on.

4. Once you're in the myth machine, get the setup script by typing

   ```
   wget http://crypto.stanford.edu/cs155/boxessetup.sh
   ```

   in the xterm.

5. Run the setup and follow the instructions printed out

   ```
   chmod u+x boxessetup.sh ; ./boxessetup.sh
   ```

   Once you've finished all the instructions, the environment is set and you're ready to start.

6. There are two accounts on these boxes, root and user. The passwords are the same as the usernames. The sploit files are in user's home directory and the targets are in /tmp.

7. vi and nano are the only two editors on boxes.

8. To create more virtual terminals, while root in boxes, do

   ```
   box:~# TERM=vt100 ; vi /etc/inittab
   ```

   and uncomment out lines:

   ```
   ##2:23:respawn:/sbin/getty 38400 vc/2
   ##3:23:respawn:/sbin/getty 38400 vc/3
   ```

   to spawn two extra console windows on the next reboot of boxes. The only two editors on the boxes environment are `nano` and `vi`. If you prefer other editors, write the code outside of boxes and then use 'nano' to paste the code into a text file in a boxes console.