**CS155: Computer Security** **Spring 2003**

# Project #2

Due: Thursday, May 15th, 2003.

**Summary** In this project, you will write Josh, the journaling operator shell. Josh is a setuid-root shell that allows users to undertake a subset of root's capabilities.

Your starting point is OSH, a minimal shell developed by Gunnar Ritter.

You may work alone, or in pairs. You should not collaborate with others outside your group.

You will use the Boxes system again. Remember to test your Josh in a `closedbox`.

This assignment specification is quite long, but much of it is intended to clarify points of confusion raised by previous years' students. Please be sure to read it carefully before beginning to code.

**The Problem** As discussed in class, the classical Unix permissions model does not manage privilege delegation effectively. To perform routine administrative tasks (such as cleaning out the `/tmp` directory), one must possess the root password, and can therefore perform other, undesirable actions (such as reading users' mail).

This limitation manifests itself most acutely in large Unix installations, where there are typically many administrators tasked with various duties.

Various systems have been devised to overcome this limitation, and allow unprivileged users to undertake some subset of root's capabilities. A commonly-used example is Sudo (`http://www.sudo.ws/`).

We will take an approach more like that taken by OSH, the Operator Shell (`http://www.engarde.com/~mcn/osh.html`, mirrored on the class Website at `http://cs155/osh.html`). You might want to read the SANS III conference paper linked from the OSH home page, and mirrored on the class Website at `http://cs155/osh.sansIII.ps`.

**Our Solution** We will develop a new shell, called Josh — the Journaling operator shell — which will enable privilege delegation in the Unix environment.

In Unix, the command interpreter is not part of the kernel, and is in fact modular and interchangeable. The command interpreter program is called a shell; it reads user requests, parses them, and makes the system calls required to fulfill the requests on the user's behalf. Seventh Edition Unix (Jan 1979) shipped with the Bourne Shell, `/bin/sh`; 2BSD (May 1979) shipped with Bill Joy's C Shell (`/bin/csh`). The KornShell (`/bin/ksh`) was first released in 1983 and was for some time an at-cost add-on for Unix System V.

Today, people typically use either `bash`, a Bourne-shell reimplementation, or `tcsh`, a C Shell enhancement.

None of these shells is designed to be run setuid-root.

**Starter Code** Writing a shell is an excellent example of a Simple Matter of Programming (`http://tuxedo.org/jargon/html/entry/SMOP.html`). Besides the `fork-exec-open-pipe` infrastructure required for implementing shell pipelines, there's a considerable amount of parsing and bookkeeping that must be taken care of.

To keep you from having to start your project by implementing a shell — an excellent CS193u project, by the way — we provide you with OSH, the Old Shell (no relation to the Operator Shell). OSH is a feature-for-feature-compatible reimplementation of the Sixth Edition shell (May 1976) by Gunnar Ritter (`http://omnibus.ruf.uni-freiburg.de/~gritter`).

While OSH lacks some features we expect in modern shells (notably backticks and control structures) it packs quite a few features into 837 lines of C. (Bash is about 100,000 lines.)

**Recommended Reading** For some practical ideas about security dos and don'ts, read David Wheeler's "Secure Programming for Unix and Linux HOWTO" (`http://www.dwheeler.com/secure-programs/`), which is linked from the course Website.

For Unix programming in general, there is no better source than W. Richard Stevens' *Advanced Programming in the UNIX Environment* (Addison-Wesley, 1992, ISBN 0-201-56317-7). Go buy it now.

You might also want to refer to the source of various software components with which Josh will interact, such as the Linux kernel (`http://www.kernel.org/`) or the GNU C Library (`http://www.gnu.org/software/libc/`).

**Using Boxes** We retain the Boxes distribution from `pp1`, as installed on the Linux machines in Sweet Hall. Since you will be doing Unix systems programming this time, we make available the `manpages-dev` package, omitted in the Boxes distribution, which contains man pages for Linux syscalls and C library routines. You can obtain this package from the CS 155 Website at

```
http://cs155/manpages-dev_1.48-2_all.deb
```

and install in in Boxes by saying, as root,

```
box:~# dpkg -i /path/to/manpages-dev_1.48-2_all.deb
```

(where you don't type the root prompt "`box:~#`" and where the actual path to the package replaces "`/path/to`").

Note that the filesystem image contains the Expect utility (`http://expect.nist.gov/`), to allow you to test your shell out non-interactively, if you wish.

The wrapper interface to Boxes remains the same. You should check out the FAQ in the Boxes distribution, and the additional information posted to the newsgroup for basic information on getting Boxes up and running.

**Step One: Secure the Perimeter** Now we describe the tasks you must undertake in creating Josh. Of course, you need not stick to the order in which we present them.

Though OSH, your starter shell, is good code, it was not written to be run setuid root.

You should start by familiarizing yourself with the structure of the shell, auditing it with security in mind, and thinking about how best to extend it.

One thing to note: OSH currently searches the working directory for a program before searching the path for the program. This is dangerous (e.g., in the case when `/tmp` contains a malicious executable named "`ls`"). You should modify OSH so it executes programs in the working directory only if `$PATH` includes "`.`" or an empty component, or if a program is invoked as, e.g., "`./program`".

Another thing to note: Previous years' students have identified a buffer overflow in `substvars()`, a buffer overflow in `striparg()`, an array overflow in `pcmd()`, and other questionable code elsewhere. Be sure either to audit carefully or otherwise to take steps to mitigate the effects of vulnerabilities in OSH.

**Step Two: Executables** Josh allows users to run some programs that otherwise they could not. The file that controls this behavior is `/etc/josh_exec`. This file (which should be installed `root:root`, mode 600) has entries, one per line, in the following format:

```
userid:progpath
```

(Without the initial indentation.) Here, "`userid`" is a user's login name; `progpath` is some absolute path to a program. This path must be absolute in that it must begin with a "`/`". The path could, however, contain components that are "`..`" or symlinks.

Any program listed in `josh_exec` for a particular user should be executed as root, not as the user, whenever it is invoked. For example, if `/etc/josh_exec` includes the line

```
alice:/bin/kill
```

Then any `/bin/kill` invocation by Alice should run as root, even though Alice might have the Unix permissions to run `/bin/kill` as herself, user `alice`. This holds however Alice runs `/bin/kill`: via the absolute path, via a relative path, or via `$PATH`, but not if Alice runs a different program named `kill`, e.g., `/home/alice/local/bin/kill`.

Josh should allow users to run programs not listed in `josh_exec`; these programs will be run with as the user and with the user's ordinary permissions, precisely as if Josh had not been setuid.

Note that your Josh will have to figure out which program is actually invoked when a user types a non-absolute file name, or relies on the value of `$PATH` to find the program.