CS155: Computer Security **Spring 2002**

Project #2

Due: Monday, May 27th, 2002.

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.

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`). You might want to read the SANS III conference paper linked from the OSH home page.

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 addon 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 The Boxes distribution has been upgraded to the latest User-Mode Linux patch. This patch should hopefully address some of the instability that was encountered during the first programming project.

The filesystem image has been upgraded to the latest from Debian. We have added 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 posted to the newsgroup for basic information on getting Boxes up and running.

Step One: Secure the Perimiter 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.

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. 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 kill invoked by alice should run as root.

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.

Your Josh should ignore any entries in `josh_exec` that do not name executable programs.

Step Three: File Access An administrator will need to read some files that are not readable by all users, and possibly to write to other files that are not writable by all users.

Josh will open these files on behalf of the administrator.

Josh decides whether to open these files based on a configuration file, `/etc/josh_access`. This file (which should be installed `root:root`, mode 600) has entries, one per line, in following format:

```
userid:filepath:perms
```

Here, `userid` is a user's login name; `filepath` is some absolute file name; and `perms` is of the form `[+-](r|w|rw)`. A + grants a positive right; a - takes away a previously granted right. For example, the entry

```
alice:/etc/motd:+w
```

Means that Alice is allowed to write to the file `/etc/motd`. (Read access need not be granted, because everyone can read `/etc/motd` by default.)

Entries are cumulative. Thus the two lines

```
bob:/etc/ssh/ssh_host_key:+r
bob:/etc/ssh/ssh_host_key:+w
```

Together mean that Bob can both read and write to the `ssh_host_key` file.

Negative rights are only meaningful in canceling a previous access grant: If a user has read or write permissions to some file without Josh, then a negative `josh_access` entry should not prevent her from accessing the file.

If the filepath specifies a directory, then the read/write permission is to apply recursively to all files under that directory. For example, the lines