CS155: Computer Security **Spring 2003**

Project #3

Due: Tuesday, June 3rd, 2003.

Summary In this project, you will perform a security audit of another group's Josh. You will produce a short audit document analyzing the other group's decisions, algorithms, and code in implementing Josh.

Additionally, if you find exploitable vulnerabilities of any sort within the Josh code you are auditing, you are encouraged to implement exploits for these vulnerabilities and include them with your audit.

The Josh to audit was sent to you by e-mail as a uuencoded tarball. If you did not receive a Josh to audit, contact the TA by e-mail.

You may recover the tarball using `uuedecode`. The `AUDIT_ID` file within the tarball identifies the group whose Josh you are auditing.

The Audit In a file called `AUDIT`, you are to produce a security audit document.

You should comment on the organization of the Josh you are audting.

For each of the tasks involved in Josh, you should document and analyze the choices which the group made in writing Josh, both the algorithms and implementation used.

As a reminder, the Josh tasks are security hardening, `josh_exec`, `josh_access`, the editor, journaling, and the extra credit; refer to the Project Two handout.

Wherever you see a potential security vulnerability, you should be sure to comment on it and its implications.

You need not write too much; 1500 words should be sufficient.

Exploit Code If you can find a security vulnerability in the Josh you are auditing, and are able to exploit it, you should write up an automated exploit for the vulnerability and include it with your submission.

We have reproduced the section "What Constitutes a Security Vulnerability" from Project Two at the bottom of this document.

As an attacker, you have a great deal of latitude in attacking Josh. You may, for example, set `/etc/josh_access` and `/etc/josh_exec` to any reasonable contents; you may create users and set their `~/.josh_*` files; you may install files where you wish. (The `Makefile` is the right place for this.)

To script interaction with Josh, you may find `expect` handy. Refer to its manpage for more information.

Deliverables As in the first programming assignment, you will use the online Leland submit script, `/usr/class/cs155/bin/submit`. This is `bfp2`.

Please do not submit from a Linux machine (e.g., the raptors). The sagas seem to work best.

The directory which you submit must contain the audit document, with filename `AUDIT`, and all exploit code. If you include executable code, you should also provide a `Makefile` for compiling it.

Along with your audit and exploits, you must include file called `ID` which contains, on a single line for each person in your team, the following: your SUID number; your Leland username; and your name, in the format last name, comma, first name. An example:

```
$ cat ./ID
3133757 binky Clown, Binky The
$
```

If you work alone, `ID` should have exactly one line. If you work with a partner, `ID` should have exactly two lines.

Do not include any identifying information in any file except `ID`.

You may also want to include a `README` file with details of your design, comments about your experiences, or suggestions for improving the assignment.

How You Will Be Graded As always, all testing will take place in a `closedbox` environment, with Josh installed setuid root as `/usr/local/bin/josh`.

You will be graded on the completeness and insightfulness of your security audit document.

In addition, any exploits you find and automate will raise your score, and lower (somewhat) the audited group's score.

Your comments will be provided to the group whose code you audit. (This, again, is why it is important that you not identify yourselves in any file other than `ID`.)

What Constitutes a Security Vulnerability (This section duplicated from Project Two.)

In an application like Josh, a security vulnerability is anything that allows a player in the system to obtain additional privileges. We list some possibilities below; this is not meant to be an exhaustive list.

Obviously, if a buffer overflow in Josh allows untrusted user Alice to obtain a root shell, that's a vulnerability.

If Alice can obtain read or write access to a file, or execute access to a program beyond what is granted to her by root, that's a vulnerability.

If Alice can obtain access to more of Bob's files than she should, that's a vulnerability.

If Bob can configure his files so that, as Alice runs Josh, she is tricked into giving Bob some subset of her privileges, that's a vulnerability.

If the system administrator, in going over the Josh journal files, is misled about the activities that took place on the system, or is tricked into giving up more privileges, that's a vulnerability.

Denial-of-service attacks may rise to the level of security vulnerabilities if they are serious: for example, if Bob can use Josh to disable Alice's login. (Attacks prevented by effective use of ulimits obviously don't count.)

In demonstrating an exploit, you may assume any reasonable configuration for the various files Josh uses to determine access; but, obviously, an exploit that derives from misconfiguration is not an exploit against Josh.