CS342 Computer Security **Handout # 9**
Profs. Daniel Bilar and Lyn Turbak **September 30, 2006**
Wellesley College

Problem Set 3
Due: Midnight Friday, October 13

Overview:

The purpose of this assignment is to give you some hands-on experience with authentication (e.g., passwords), access control (e.g., permissions), and simple computer forensics.

Reading:

In order to do this assignment, you will need to have a firm understanding of permissions in Linux, especially the "set user ID" (a.k.a. setuid or SUID) and "set group ID" (a.k.a. setgid or SGID) bits. Here are some helpful references:

- *Hacking Linux Exposed: Linux Security Secrets & Solutions* by Brian Hatch, James Lee, and George Kurtz is a valuable reference for many Linux security issues, including permissions and the elevation of user privileges (the latter of which is especially helpful for Problem 2). There is a copy of this book in 121B. *Please leave this book in 121B so that it is available to everyone in the class.*
- Wayne Pollock's *Unix File and Directory Permissions and Modes* (`http://wpollock.com/AUnix1/FilePermissions.htm`) summarizes essential details of Unix permissions (upon which Linux permissions are based).
- Hao Chen, David Wagner, and Drew Dean's *Setuid Demystified* (`http://www.cs.berkeley.edu/~daw/papers/setuid-usenix02.pdf`) is a technical paper giving a detailed model of setuid/setgid in various flavors of Unix. Although most of the technical details are not relevant to this assigment, the first few pages are still helpful for understanding setuid/setgid.

Working Together:

There are two problems on this assignment, both of which are lab problems. You *must* work on these problems with your lab buddies. Moreover you should *not* consult with other teams on either of these problems; you need to figure everything out with your lab buddies.

Submission:

Each lab team should submit one hardcopy solution by sliding it under *Lyn's* door. No softcopy solution is necessary. For Problem 1, your hardcopy should contain written solutions to all five parts. For Problem 2, for each secret file you capture, your hardcopy should contain the contents of the file and a detailed description of what steps you took (and why you took them) to capture the file.

A Note About How to Work on this Assignment

This is a problem set that is best done a little bit at a time rather than all at once. You are likely to hit many brick walls along the way, and you'll need time to step back, reflect, and figure out how to circumvent them. Some of the steps you need to take (such as installing and running the *John the Ripper* password cracker in Problem 2) take time and cannot be done at the last minute. Finally, it is strongly recommended that you work on both problems at the same time. Waiting to finish Problem 1 before starting Problem 2 is a bad strategy because (1) it's a good idea to maximize your time to think about Problem 2 and (2) you may learn things doing Problem 2 that will help you with Problem 1.

Group Problem 1 [30]: Attack Forensics

a. You've been attacked — we know all of your root passwords! (We have sent each team an email containing its root password.) This problem is to figure out how we managed to get them. To help you out, we've left behind some information that a good hacker would not normally leave behind. Put on your detective hats, look for clues, and write up a description of exactly what we did to steal your root passwords. Linux's `find` and `grep` commands are particularly helpful for this task.

b. We could have made it much more difficult to find out what we did by covering our tracks more carefully. Explain the steps we could have taken to make your detective work tougher.

c. We did clean up *some* of our tracks. Are there clues you expected to find but did not?

d. Repair your system so that we cannot monitor your future password changes. Describe how you cleaned things up.

e. As competent hackers, we have left behind some other "backdoors" into your system so that we can still gain root access to your machines even after you clean up and change your root password. Can you find and remove them? Describe your efforts. How sure are you that you have found all the backdoors?

Group Problem 2 [70]: Treasure Hunt

Machine #3 in lab (`192.168.0.3`) has dedicated accounts for each of the four CS342 lab teams this semester:

- `team4`: machine #4 team (Sarah Abraham & Hitomi Yoneya)
- `team5`: machine #5 team (Heather Conner, Rocio DeLao, and Anita Yip)
- `team6`: machine #6 team (Eylul Dogruel and Ariel Katz)
- `team7`: machine #7 team (Vasumathi Raman and Rebecca Shapiro)

The passwords for these accounts are the same as your original root passwords on your team machines (i.e., the ones the instructors discovered for Problem 1).

Machine #3 also has accounts for the following twelve imaginary users:

1. Alice Wan (`awan`)
2. Bob Duomo (`bduomo`)
3. Cathy Dry (`cdry`)
4. Debbie Shi (`dshi`)
5. Ellen Quintus (`equintus`)
6. Fred Sexon (`fsexon`)
7. George Hepta (`ghepta`)
8. Helen Ochoa (`hochoa`)
9. Inna Nunez (`inunez`)
10. Jack Dekka (`jdekka`)
11. Karen Elva (`kelva`)
12. Loretta Douzette (`ldouzette`)

Somewhere in his/her home directory, each of these imaginary users has a "secret" file named *username*`-secret.txt` whose contents are intended to be private (readable only by the user and no one else). However, all of the users are Linux newbies who are not very knowledgeable about permissions and other security loopholes in Linux. So it turns out that each of their secret files can actually be read by other users who are both determined and clever.

Your goal is to collect the contents of as many of the twelve secret files as you can. For each such file, you should not only give the contents of the file, but also provide a detailed description of the steps you took to "capture" it and why you took these steps. For files you succeeded in capturing, you need only describe successful steps, not unsuccessful ones. For every secret file that you do not succeed in capturing, you should still provide a description of your attempts to capture it for partial credit.

Hints/Advice/Rules

- Do not do your work directly at the machine #3 console. Instead, work on your team machine and SSH to machine #3 as follows:

  `ssh -X -Y` *`team-name`*`@192.168.0.3`

  (Note: for some reason, this may take a while — a minute or so. Be patient.) The `-X -Y` flags allow you to open windows from machine #3 on your team machine via so-called *SSH tunneling*.

- Some of the files can be captured with very little work; others require significant work and/or cleverness. There is no relationship between the order of the users presented above and the difficulty of capturing their secret files. In particular, users earlier in the list don't necessarily have files that are easier to capture.

- The user's home directories contain files other than the secret file. In some cases, some of these other files are in useful for capturing the secret file.

- In order to capture some secret files, you need to crack the passwords of some users. To do this, you will need to take two steps:

  1. Gain access to the file `/etc/shadow`, which stores the hashed passwords of the users. This file is normally readable only by `root`, but the bungling sysadmins of machine #3 have accidentally managed to make it readable by any user knowledgeable about setuid/setgid.
  2. Install and run the *John the Ripper* password cracker (`http://www.openwall.com/john/`). The documentation that comes with the software tells you how to install and use it. You can use it in the simplest mode with the default wordlist to obtain the passwords you need. (Please let Lyn or Daniel know if you need help installing, configuring, or running *John*.)

- Each user has at least one web page in a `public_html` directory. Some of these pages contain information relevant to this problem. Although many of the user web pages are publicly readable by any user on machine #3, some can *only* be read via a web browser. Here are some things you need to know about web pages:

  – In an ideal world, you could access the home page for user *username* on machine #3 from your team machine by specifying the URL `http://192.168.0.3/~`*username* in a web browser on your team machine. However, as-yet-unresolved firewall issues currently prevent this from working. Instead, you need to connect to machine #3 via SSH (as described above) and execute `mozilla &` to launch a web browser on machine #3 itself. You can then use the expected URL in this web browser.

- Access to web directories can be controlled by a `.htaccess` file. E.g., see `http://www.javascriptkit.com/howto/htaccess.shtml` for documentation on `.htaccess` files.
- The web server runs as user/group `apache`. Including `apache` in a group gives the web server whatever permissions are given to the group.
- The HTML source of a `.html` file can be viewed using the `View>Page Source` menu item in Firefox.
- In an HTML file, text between `<!--` and `>` is a comment that is not displayed by the web browser.

- You should skim Chapter 8 of *Hacking Linux Exposed* ("Elevating User Privileges") for a description of various ways to elevate your privileges in Linux. You may find some of these helpful in this problem. Pay special attention to path attacks, symlink attacks, and user validation attacks.

- Several of the attacks involve using setuid/setgid programs. It's a good idea to figure out how to find all such programs accessible to your team on machine #3.

  Because such programs are notorious for causing security holes, Linux makes it difficult to create such programs. For example, Linux ignores the setuid/setgid bit for shell scripts. In order to get the effect of setuid/setgid for a script `foo_script`, the following steps can be taken:

  1. `foo_script` must begin with the declaration `#!/bin/bash -p`. The `-p` option is essential for the setuid/setgid behavior to be observed.
  2. Write a C program `foo.c` that invokes the `execv` function on `"foo_script"` and whatever arguments `foo_script` expects. `execv` replaces the current process with a process for computing the invoked command.
  3. Compile `foo.c` to the binary `foo` via `gcc -o foo foo.c`.
  4. Set the permissions on `foo` to be both (1) executable and (2) setuid via `chmod 4755 foo`.

  At the end of these steps, invoking the command `foo` will have the effect of executing `foo_script` with setuid priviliges.

  You do *not* have to perform any of these steps in this assignment. So why are we explaining this to you? Because some of the imaginary users have followed these steps to make setuid/setgid shell scripts. In several cases, you will need to examine the contents of the shell scripts in order to launch an exploit against the user.

- Accessing some secret files will require that you make changes to certain files/directories in the accounts of other users. Once you determine the secret, be sure to undo any changes that you make so that you leave the system exactly in the same state that you found it. Otherwise, you could (1) make it very easy for other teams to access the information you worked so hard to get or (2) make it impossible for other teams to access the information you found (this is unacceptable in this assignment, though not in the real world). *If you (1) make a change that you are unsure how to undo or (2) encounter a state that you suspect is incorrect, please contact Daniel or Lyn ASAP.*

- It is not expected that you should be able to become root on machine #3. If you are able to do this, we have done something terribly wrong! If you succeed in becoming root, you will get extra credit. However, even if you do become root, you must not use your root privileges to access any of the secret files; you must figure out a non-root way to capture the secrets.