System Administration for Beginners

Week 10 Notes

April 10, 2007

1 Introduction

1.1 Coming Soon

Next week, before lecture, all the vservers will be reset and a fresh install of Debian GNU/Linux will be loaded. Some of you may have noticed that the installations are rather bare, so, using Aptitude, I will pre-configure them with the packages using the filter `(~prequired|~pimportant|~pstandard)`. With the exception of `setools`, you should have a good foundation for starting your final project.

1.2 Lectures and Assignments

We will be covering more security this week in lecture. As we approach the end of the semester, I do not have any set material to cover (as my predecessors before me did not). At the beginning of this course, I asked for any topics you may want to learn about. I have taken those into consideration and will present lectures on advanced topics of system administration that I feel are interesting, which may include: web design/scripting, cryptography, advanced networking, etc.

If there is a topic that interests you (many of the course survey responses were blank regarding this), feel free to let me know in person or in an email and I will consider it as a topic to cover.

Thus, the rest of the class period will be yours to work on your project. There will not be any labs regarding the topics I will be talking about, although you will get a chance to implement and explore these topics in detail as part of the final project. Also, I may assign homework and quizzes that inquire about your understanding of the material.

1.3 Submissions

What does all this mean? If you haven't been reading anything else, then this is probably the most important part. Since your vservers will be reset next week, all prior lab work and homework are due before next week's lecture.

2 Security

2.1 Web Server Security

Last week, I explained that, because Apache runs with the permissions of a regular user, you have to explicitly grant Apache access to files and directories that you want to be accessible. In the case of configuration files for Internet applications, this basically meant granting everybody read access to the file, a very insecure setup. We solved this problem of liberal permissions by using access control lists to specifically grant Apache read access, but such a solution becomes tedious to manage when you have many files.

2.1.1 suExec and cgiwrap

An ideal solution would allow Internet applications to run as the user who owns them. `suExec`, a special feature built into Apache, provides this ability; for other web servers, there is a comparable tool called `cgiwrap`.

When Apache encounters a script that can be handled by `suExec`, it temporarily becomes the user who owns the script under a mechanism similar to that used by `su` and executes the script. In that way, Apache temporarily has access to the all the files and directories to which the user has access, and thereby needs no special read or write access to the file.

Many hosting providers use `suExec` or `cgiwrap` to allow Internet applications to run in a secure manner. Even though Apache runs as non-privileged user, it does have a lot of access, since applications and files not managed by `suExec` or `cgiwrap` must grant access to Apache. Consequently, if Apache is compromised through one of these files, an attacker will have access to all the files. On the other hand, is Apache is compromised through a script managed by `suExec` or `cgiwrap`, only that user's account will be affected.

2.1.2 The Performance Compromise

Not all hosting providers use `suExec` or `cgiwrap` to manage Internet applications. In fact, by default, if you install Apache and PHP, Apache will run PHP through a special module called `mod_php` that is not managed by `suExec` or `cgiwrap`. The reason for this is that suExec and cgiwrap are not very efficient. Every time a person requests a script managed by `suExec` or `cgiwrap`, Apache has to spawn a special copy of itself running as the user who owns the script. For high-traffic websites, this means hundreds of Apache processes many be running on a system. Furthermore, due to various technical reasons, `suExec` and `cgiwrap` cannot cache copies of a script in memory, which means a script has to be processed by the system every time it is run – modules like `mod_php` can cache scripts and reduce the load placed on a system.

As a system administrator, you need to decide if the trade-off in performance is worth the extra security. On a system with many different users, the added security of `suExec` and `cgiwrap`, which limits compromises in Internet applications to the account that caused them, is well worth it. On other systems, where

there is only one user, or a set of related, trusted users, the performance gains offered by not using `suExec` or `cgiwrap` may be worth the decrease in security.

2.2 Auditing Log Files

To assist system administrators, most server operating systems and server daemons maintain extensive logs of everything that occurs. These logs are extremely useful for security auditing and noticing potential problems before they become actual problems.

On Unix-like systems, logs are stored in the `/var/log` directory. Most logs are named after the information they contain. For example, `auth.log` contains information about successful and unsuccessful login attempts to the system, `daemon.log` contains general information produced by server daemons, and `user.log` contains information about user actions. Most log files are plaintext files which can be read with any text editor or pager, and most can be parsed using grep or a programming language.

To beginning system administrators, the information contained in log files can be overwhelming; by default, most operating systems log as much information as possible. While you can change settings and instruct the system to only log critical errors, it is not a good idea; oftentimes, seemingly useless information in the log files can provide the context necessary to understand why a certain error has occurred.

Many tools are available for summarizing log information. Generally, though, you use a different tool for each type of log file, since operating systems and server daemons tend to use different log formats.

2.2.1 logcheck

`logcheck` is a popular tool used by system administrators to summarize log information and extract important entries in log files. `logcheck` basically goes through each log file and, using pattern recognition software, attempts to locate important entries, which it emails to system administrators. Important log entries, by default, include failed attempts at logging into the system, hardware failure, and problems with certain server daemons. What constitutes an important log entry can be set by editing the configuration files for `logcheck`; `logcheck` comes with a few sample configuration files for servers, workstations, and regular user computers.

2.2.2 webalizer

By default, most web servers maintain two types of log: a log of all successful visits to a website, and a log of all unsuccessful visits and errors produced by the web server daemon. Apache calls these files `access.log` and `error.log`, respectively. The latter log is in a format that logcheck can understand, but the former log can not be parsed by `logcheck`.

One tool for parsing web server access logs is `webalizer`. You may recognize this program as a statistics script for web servers, but it also useful for system administration. The information produced by `webalizer` can give system administrators an idea of the amount of web traffic a certain website uses, the type of visitors who visit the website, and popular files on the website. This latter statistic, the most popular files on a website, is often used by system administrators to determine if a website is hosting any illegal or pirated files. webalizer also gives system administrators an idea of the bandwidth consumed by a website, which is useful for billing purposes.

2.3 Patches

Humans, especially programmers, are not perfect. People will make mistakes when they write programs, and sometimes these mistakes can be exploited by malicious users. Most of the time these mistakes result in relatively harmless errors, such as causing a server daemon to shutdown, but occasionally these mistakes allow a person to gain unauthorized access.

Hackers and security companies spend a considerable amount of time reading through the code of popular programs to locate these mistakes. Generally, if a hacker discovers a mistake that results in a security hole, he'll write a program that will *exploit* or take advantage of the mistake, and post it on the Internet for other hackers to use. When a security company finds a security hole, they generally issue a *security advisory*, which basically informs people about the existence of the security hole and the severity of it. Sometimes, they'll even post a *proof of concept* program that exploits the hole to prove that the security hole exists. In either case, users of the affected program are immediately at risk.

Software vendors usually respond to exploits and security advisories by issuing a *patch*, or fix for the security hole. The time between the release of an exploit or advisory and the release of a patch generally depends upon the severity of the security hole, but some commercial software vendors are notorious for denying the existence of a security hole and refusing to issue a patch or delaying the release of a patch so that it may be integrated into a larger collection of patches, often termed a *service pack*. In the case of such software vendors, system administrators may find it necessary to disable a service or program until a patch is issued or use a workaround if one is available.

When a patch is issued by a software vendor, it's the responsibility of a system administrator to test the patch to ensure that it works and to deploy it to all the affected systems in a timely manner. Failure to stay current with patches is probably the main reasons why systems are hacked.

Nowadays, most vendors of server operating systems provide tools for automatically downloading and installing patches. For Microsoft Windows, there's Windows Update. For Sun Microsystems Solaris, there's Sun Update Manager. For Debian Linux, the APT set of package tools will automatically download and install updates. Most other Linux distributions have their own set of tools, and if you plan on deploying a certain distribution, it would be important to investigate it's software update tools prior to making your final decision – if a

distribution lacks tools for automatically updating a system or makes it a difficult, it'll only create more hassle for you and encourage you to fall behind with patches.

2.4 Automating Tasks

System administrators perform many repetitive tasks. For example, a system administrator may use find every morning to check to if there are files or directories on the system with 777 permissions. Remembering to and actually performing such tasks becomes tedious, as a system administrator may eventually develop a long checklist of tasks that must be on an occasional, daily, weekly, monthly, or other periodic time-frame. Thankfully, there many tools available to automate such tasks.

2.4.1 Scripting

Rather than typing out the syntax of a set of commands every time you want to perform a repetitive task, you can put all the commands, one per a line, in a file and tell the system to execute the file. This process is called scripting.

For example, to delete all files that end in `.tmp` and `.aux` in a certain directory, you would use the following command:

```
rm *.tmp *.aux
```

Rather than executing than typing out this command every time you wanted to a cleanup a directory, you put it in a file named `cleanup.sh` (the convention is to name script files with a `.sh` extension). To execute the script, you provide the name of your shell and the name of the script file:

```
bash cleanup.sh
```

This basically tells the system to launch a new bash shell, execute all the commands in the file `cleanup.sh`, and close the shell. If any of the commands produces output, it will be displayed on your screen.

Rather than providing the name of your shell every time you want to execute a script file, you can provide the name of the shell inside the script file and execute the script file directly. To provide the name of the shell, you must insert a special line at the beginning of your script file with the full path to your shell, prefixed by a pound and exclamation mark. In the case of `bash`, the default shell on Linux:

#!/bin/bash

Therefore, the full contents of cleanup.sh become:

#!/bin/bash rm *.tmp *.aux

Now you may execute `cleanup.sh` directly. If you are in the same directory as `cleanup.sh`:

```
./cleanup.sh
```

Please note that you must have execute permissions on `cleanup.sh`.

2.4.2 cron

While creating script files solves the problem of having to type a set of commands over and over again, it does not solve the problem of having to remember to execute the script on a periodic basis. Thankfully, another tool is provided for that purpose: `cron`, a special daemon on the system that can execute commands at regular intervals.

To add an item to `cron`:

```
crontab -e
```

`crontab` will open up your cron file in your default editor and allow you to make modifications to the file. Each user has their own cron file, and each line in the file is composed of six whitespace-delimited fields, the first five of which are numeric:

- minute
- hour
- day of the month
- month of the year
- day of the week (Sunday is 0, Saturday is 6)

The last field is the command to execute. You may provide the name of a script you have written, but remember to provide the full path to the script. Also, any output generated by the file will be sent to the account of the cron file you are editing; in other words, if you are editing root's crontab, the output will be sent to root.

2.5 Minimum User Access

Sometimes you want to grant a user access to a system, but do not wish to grant them the ability to execute commands on the system. For example, you may want to allow a user to upload and download files to a certain account, but you may not want to grant them shell access. Such an arrangement is common on web servers, where users only need to be able to upload new web pages.

To accomplish this restriction, you may use `scponly`, a special shell that only allows users to connect using `sftp` and `scp`. When adding users, you may pass an additional parameter to `adduser` to specify `scponly` as their shell. Users added in such a manner will only be able to log into the system with `sftp` or `scp`. To change the shell of pre-existing users, you many use `vipw`.