Linux Security HOWTO

5. Files and File system Security

A few minutes of preparation and planning ahead before putting your systems on-line can help to protect them and the data stored on them.

- There should never be a reason for users' home directories to allow SUID/SGID programs to be run from there. Use the `nosuid` option in `/etc/fstab` for partitions that are writable by others than root. You may also wish to use `nodev` and `noexec` on users' home partitions, as well as `/var`, thus prohibiting execution of programs, and creation of character or block devices, which should never be necessary anyway.

- If you are exporting file-systems using NFS, be sure to configure `/etc/exports` with the most restrictive access possible. This means not using wild cards, not allowing root write access, and exporting read-only wherever possible.

- Configure your users' file-creation `umask` to be as restrictive as possible. See Section 5.1.

- If you are mounting file systems using a network file system such as NFS, be sure to configure /etc/exports with suitable restrictions. Typically, using `nodev', `nosuid', and perhaps `noexec', are desirable.

- Set file system limits instead of allowing `unlimited` as is the default. You can control the per-user limits using the resource-limits PAM module and `/etc/pam.d/limits.conf`. For example, limits for group `users` might look like this:

  ```
  @users     hard  core    0
  @users     hard  nproc   50
  @users     hard  rss     5000
  ```

  This says to prohibit the creation of core files, restrict the number of processes to 50, and restrict memory usage per user to 5M.

  You can also use the /etc/login.defs configuration file to set the same limits.

- The `/var/log/wtmp` and `/var/run/utmp` files contain the login records for all users on your system. Their integrity must be maintained because they can be used to determine when and from where a user (or potential intruder) has entered your system. These files should also have `644` permissions, without affecting normal system operation.

- The immutable bit can be used to prevent accidentally deleting or overwriting a file that must be protected. It also prevents someone from creating a hard link to the file. See the `chattr`(1) man page for information on the immutable bit.

- SUID and SGID files on your system are a potential security risk, and should be monitored closely. Because these programs grant special privileges to the user who is executing them, it is necessary to ensure that insecure programs are not installed. A favorite trick of crackers is to exploit SUID-root programs, then leave a SUID program as a back door to get in the next time, even if the original hole is plugged.

  Find all SUID/SGID programs on your system, and keep track of what they are, so you are aware of any changes which could indicate a potential intruder. Use the following command to find all SUID/SGID programs on your system:

  ```
  root#  find / -type f \( -perm -04000 -o -perm -02000 \)
  ```

```
root# find / -type f \( -perm -04000 -o -perm -02000 \)
```

The Debian distribution runs a job each night to determine what SUID files exist. It then compares this to the previous night's run. You can look in `/var/log/setuid*` for this log.

You can remove the SUID or SGID permissions on a suspicious program with `chmod`, then restore them back if you absolutely feel it is necessary.

- World-writable files, particularly system files, can be a security hole if a cracker gains access to your system and modifies them. Additionally, world-writable directories are dangerous, since they allow a cracker to add or delete files as he wishes. To locate all world-writable files on your system, use the following command:

  ```
  root# find / -perm -2 ! -type l -ls
  ```
  and be sure you know why those files are writable. In the normal course of operation, several files will be world-writable, including some from `/dev`, and symbolic links, thus the `! -type l` which excludes these from the previous `find` command.

- Unowned files may also be an indication an intruder has accessed your system. You can locate files on your system that have no owner, or belong to no group with the command:

  ```
  root# find / \( -nouser -o -nogroup \) -print
  ```

- Finding `.rhosts` files should be a part of your regular system administration duties, as these files should not be permitted on your system. Remember, a cracker only needs one insecure account to potentially gain access to your entire network. You can locate all `.rhosts` files on your system with the following command:
  ```
  root# find /home -name .rhosts -print
  ```

- Finally, before changing permissions on any system files, make sure you understand what you are doing. Never change permissions on a file because it seems like the easy way to get things working. Always determine why the file has that permission before changing it.

5.1. Umask Settings

The `umask` command can be used to determine the default file creation mode on your system. It is the octal complement of the desired file mode. If files are created without any regard to their permissions settings, the user could inadvertently give read or write permission to someone that should not have this permission. Typical `umask` settings include `022`, `027`, and `077` (which is the most restrictive). Normally the umask is set in `/etc/profile`, so it applies to all users on the system. The resulting permission is calculated as follows: The default permission of user/group/others (7 for directories, 6 for files) is combined with the inverted mask (NOT) using AND on a per-bit-basis.

Example 1:

file, default 6, binary: 110 mask, eg. 2: 010, NOT: 101

resulting permission, AND: 100 (equals 4, r__)

Example 2:

file, default 6, binary: 110 mask, eg. 6: 110, NOT: 001

resulting permission, AND: 000 (equals 0, ___) Example 3:

directory, default 7, binary: 111 mask, eg. 2: 010, NOT: 101

resulting permission, AND: 101 (equals 5, r_x)

Example 4:

directory, default 7, binary: 111 mask, eg. 6: 110, NOT: 001

resulting permission, AND: 001 (equals 1, __x)

```
		# Set the user's default umask
		umask 033
```

Be sure to make root's umask `077`, which will disable read, write, and execute permission for other users, unless explicitly changed using `chmod`. In this case, newly-created directories would have 744 permissions, obtained by subtracting 033 from 777. Newly-created files using the 033 umask would have permissions of 644.

If you are using Red Hat, and adhere to their user and group ID creation scheme (User Private Groups), it is only necessary to use `002` for a `umask`. This is due to the fact that the default configuration is one user per group.

5.2. File Permissions

It's important to ensure that your system files are not open for casual editing by users and groups who shouldn't be doing such system maintenance.

Unix separates access control on files and directories according to three characteristics: owner, group, and other. There is always exactly one owner, any number of members of the group, and everyone else.

A quick explanation of Unix permissions:

Ownership - Which user(s) and group(s) retain(s) control of the permission settings of the node and parent of the node

Permissions - Bits capable of being set or reset to allow certain types of access to it. Permissions for directories may have a different meaning than the same set of permissions on files.

Read:

- To be able to view contents of a file
- To be able to read a directory

Write:

- To be able to add to or change a file
- To be able to delete or move files in a directory

Execute:

- To be able to run a binary program or shell script
- To be able to search in a directory, combined with read permission

Save Text Attribute: (For directories)

The "sticky bit" also has a different meaning when applied to directories than when applied to files. If the sticky bit is set on a directory, then a user may only delete files that the he owns or for which he has explicit write permission granted, even when he has write access to the directory. This is designed for directories like `/tmp`, which are world-writable, but where it may not be desirable to allow any user to delete files at will. The sticky bit is seen as a `t` in a long directory listing.

SUID Attribute: (For Files)

This describes set-user-id permissions on the file. When the set user ID access mode is set in the owner permissions, and the file is executable, processes which run it are granted access to system resources based on user who owns the file, as opposed to the user who created the process. This is the cause of many "buffer overflow" exploits.

SGID Attribute: (For Files)

If set in the group permissions, this bit controls the "set group id" status of a file. This behaves the same way as SUID, except the group is affected instead. The file must be executable for this to have any effect.

SGID Attribute: (For directories)

If you set the SGID bit on a directory (with `chmod g+s directory`), files created in that directory will have their group set to the directory's group.

You - The owner of the file

Group - The group you belong to

Everyone - Anyone on the system that is not the owner or a member of the group

File Example:

```
-rw-r--r--  1 kevin  users         114 Aug 28  1997 .zlogin
1st bit - directory?             (no)
 2nd bit - read by owner?         (yes, by kevin)
  3rd bit - write by owner?        (yes, by kevin)
   4th bit - execute by owner?      (no)
    5th bit - read by group?         (yes, by users)
     6th bit - write by group?        (no)
      7th bit - execute by group?      (no)
       8th bit - read by everyone?      (yes, by everyone)
        9th bit - write by everyone?     (no)
         10th bit - execute by everyone?  (no)
```

The following lines are examples of the minimum sets of permissions that are required to perform the access described. You may want to give more permission than what's listed here, but this should describe what these minimum permissions on files do:

```
-r--------  Allow read access to the file by owner
--w-------  Allows the owner to modify or delete the file
            (Note that anyone with write permission to the directory
             the file is in can overwrite it and thus delete it)
---x------  The owner can execute this program, but not shell scripts,
             which still need read permission
---s------  Will execute with effective User ID = to owner
--------s-  Will execute with effective Group ID = to group
-rw------T  No update of "last modified time".  Usually used for swap
             files
```

```
---t------  No effect.  (formerly sticky bit)
```

Directory Example:

```
        drwxr-xr-x  3 kevin  users         512 Sep 19 13:47 .public_html/
        1st bit - directory?             (yes, it contains many files)
         2nd bit - read by owner?         (yes, by kevin)
          3rd bit - write by owner?        (yes, by kevin)
           4th bit - execute by owner?      (yes, by kevin)
            5th bit - read by group?         (yes, by users
             6th bit - write by group?        (no)
              7th bit - execute by group?      (yes, by users)
               8th bit - read by everyone?      (yes, by everyone)
                9th bit - write by everyone?     (no)
                 10th bit - execute by everyone?  (yes, by everyone)
```

The following lines are examples of the minimum sets of permissions that are required to perform the access described. You may want to give more permission than what's listed, but this should describe what these minimum permissions on directories do:

```
dr--------  The contents can be listed, but file attributes can't be read
d--x------  The directory can be entered, and used in full execution paths
dr-x------  File attributes can be read by owner
d-wx------  Files can be created/deleted, even if the directory
             isn't the current one
d------x-t  Prevents files from deletion by others with write
             access. Used on /tmp
d---s--s--  No effect
```

System configuration files (usually in `/etc`) are usually mode `640` (`-rw-r-----`), and owned by root. Depending on your site's security requirements, you might adjust this. Never leave any system files writable by a group or everyone. Some configuration files, including `/etc/shadow`, should only be readable by root, and directories in `/etc` should at least not be accessible by others.

SUID Shell Scripts

SUID shell scripts are a serious security risk, and for this reason the kernel will not honor them. Regardless of how secure you think the shell script is, it can be exploited to give the cracker a root shell.

5.3. Integrity Checking

Another very good way to detect local (and also network) attacks on your system is to run an integrity checker like `Tripwire`, `Aide` or `Osiris`. These integrety checkers run a number of checksums on all your important binaries and config files and compares them against a database of former, known-good values as a reference. Thus, any changes in the files will be flagged.

It's a good idea to install these sorts of programs onto a floppy, and then physically set the write protect on the floppy. This way intruders can't tamper with the integrety checker itself or change the database. Once you have something like this setup, it's a good idea to run it as part of your normal security administration duties to see if anything has changed.

You can even add a `crontab` entry to run the checker from your floppy every night and mail you the results in the morning. Something like:

```
                # set mailto
                MAILTO=kevin
                # run Tripwire
                15 05 * * * root /usr/local/adm/tcheck/tripwire
```

will mail you a report each morning at 5:15am.

Integrity checkers can be a godsend to detecting intruders before you would otherwise notice them. Since a lot of files change on the average system, you have to be careful what is cracker activity and what is your own doing.

You can find the freely available unsusported version of `Tripwire` at http://www.tripwire.org, free of charge. Manuals and support can be purchased.

`Aide` can be found at http://www.cs.tut.fi/~rammer/aide.html.

`Osiris` can be found at http://www.shmoo.com/osiris/.

5.4. Trojan Horses

"Trojan Horses" are named after the fabled ploy in Virgil's "Aenid". The idea is that a cracker distributes a program or binary that sounds great, and encourages other people to download it and run it as root. Then the program can compromise their system while they are not paying attention. While they think the binary they just pulled down does one thing (and it might very well), it also compromises their security.

You should take care of what programs you install on your machine. RedHat provides MD5 checksums and PGP signatures on its RPM files so you can verify you are installing the real thing. Other distributions have similar methods. You should never run any unfamiliar binary, for which you don't have the source, as root. Few attackers are willing to release source code to public scrutiny.

Although it can be complex, make sure you are getting the source for a program from its real distribution site. If the program is going to run as root, make sure either you or someone you trust has looked over the source and verified it.

Prev Home Next
Local Security Password Security and Encryption