Filesystem Labeling in SELinux

By James Morris

Abstract

With NSA Security-Enhanced Linux now integrated into the 2.6 kernel and making its way into distributions, an increasing number of people likely will be installing SELinux and experimenting with it. Given this increasing user base, this article takes a closer look at filesystem labeling under SELinux. This is an intermediate-level article, though a brief review of some SELinux concepts is provided in the next section.

November 2004

Reprinted with permission of Linux Journal
Resources for this article: www.linuxjournal.com/article/7689

Table of Contents

SELinux Overview: Labeling and Access Control

In SELinux, important objects, such as tasks, inodes and files are assigned a security context, a label that encapsulates the security attributes associated with an object. Under standard SELinux, this label is a colon-separated ASCII string composed of values for identity, role and type.

These labels are assigned by a kernel component called the security server, using rules loaded into the security policy database. The security context label on my workstation' s/etc/shadow file is:

```
system_u:object_r:shadow_t
```

where system_u and object_r are generic identity and role values used for files. shadow_t is the type of the file, an attribute that determines how the file can be accessed. A process, for example, would be labeled like this:

```
root:staff_r:staff_t
```

The SELinux identity here is root, assigned by SELinux as a more permanent form of identity than the standard UNIX identity. The role is staff_r, indicating that the process has all of the permissions assigned to that role. The type assigned to the process is staff_t. For a process, the type attribute indicates how it is allowed to access objects and interact with other processes. The type attribute of a process often is referred to as a domain.

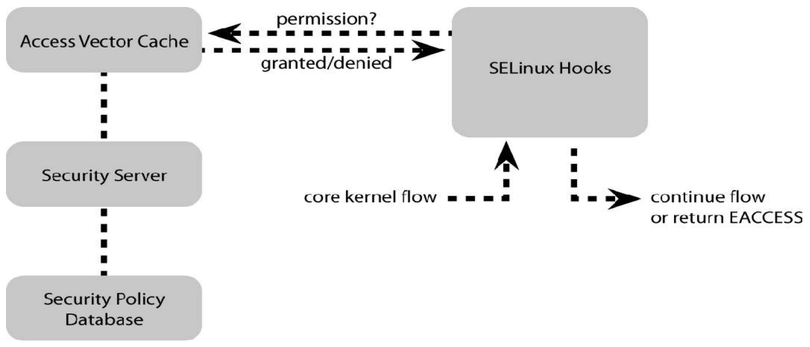
Access Control Decisions

SELinux uses these security context labels to make access control decisions between processes and objects, but how does this happen? SELinux has hooks located at strategic points within the core kernel code, such as the point where a file is about to be read by a user. These hooks allow SELinux to break out of the normal flow of the kernel to request extended access control decisions. Access control decisions usually are made between a process (for example, cat) and an object (for example, /etc/shadow) for a specific permission (read).

Decision requests are sent to the access vector cache (AVC), which passes requests through to the security server for interpretation. The security server consults the security policy database and determines a result, which is cached in the AVC and returned to the SELinux hook.

The SELinux hook then allows the flow to continue or return EACCES, depending on the decision result. Security context labels assigned to processes and objects are used to make these access control decisions.

The illustration below shows a simplified view of this process.



The following code demonstrates how security context labels are used on a real system, as seen by a user:

```
$ id -Z
root:staff_r:staff_t

$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

The audit log records the following:

```
avc:  denied  { read } for  pid=13653 exe=/bin/cat
name=shadow dev=hda6 ino=1361441 scontext=root:staff_r:staff_t
tcontext=system_u:object_r:shadow_t tclass=file
```

Translation: the cat program, labeled with the security context `root:staff_r:staff_t`, was denied permission to read a file `labeled system_u:object_r:shadow_t`.

SELinux knows nothing about the meaning of cat or /etc/shadow. It is concerned only with their respective security context labels, the class of the target object (in this case, it's a file) and the permission being requested.

An important aspect of SELinux design is that labels encapsulate all security attributes of an object, and they are interpreted only by the security server in the kernel and by libselinux in user space. The rest of the kernel code and user space merely pass labels around as opaque data. New security attributes can be added to labels without having to recompile applications or redesign core SELinux code.

Extended Attributes

On a typical Linux disk-based filesystem, each file is identified uniquely by an inode containing critical metadata for the file, including UNIX ownership and access control information. When the kernel references a file, its inode is read from disk into memory. A standard UNIX permission check simply uses the information present within the inode. SELinux extends standard UNIX security and uses security context labels to make extended access control decisions.

Linux implements Extended Attributes, also called EAs or xattrs. These are name/value pairs associated with files as an extension to normal inode-based attributes. EAs allow functionality to be added to filesystems in a standardized way so that interfaces to the attributes are filesystem-independent. Examples of EA functionality are access control lists (ACLs), storage of character-set metadata alongside file data and SELinux security context labeling.

EAs are stored within namespaces, allowing different classes of EAs to be managed separately. ACLs are stored in the system.posix_acl_access and system.posix_acl_default namespaces. SELinux security context labels are stored in the security.selinux namespace. See attr(5) for more information on EAs under Linux.

EA Security Labeling

EAs can be managed manually with the getfattr(1) and setfattr(1) utilities. For example, to view the SELinux security context label of a file:

```
$ getfattr -n security.selinux /tmp/foo
getfattr: Removing leading '/' from absolute path names
```

```
# file: tmp/foo
security.selinux="root:object_r:sysadm_tmp_t\000
```

Notice the specification of the EA security namespace. A wrapper utility called getfilecon(1) is provided for use with SELinux. It saves you from having to specify the EA namespace, and it has a cleaner output.

The use of text-based labels ensures that meaningful, human-readable security attributes are stored along with the file data. These labels can be preserved or translated if the filesystem is mounted on a different system, possibly with a different security policy. A counter example is the way the owner of a file is stored as a numeric UID in the file's inode. The UID typically is mapped to a meaningful value by way of /etc/passwd; it may not have the same meaning on a different system.

For a filesystem to support SELinux security context labels, it needs EA support and a handler for the EA security namespace. Such filesystems currently include ext3, ext2, XFS and ReiserFS; the last uses an external patch. In addition, the devpts filesystem has a dummy security handler that allows EA-based access to the in-kernel labels of ptys.

So, when are files labeled? During an SELinux system installation, the setfiles(8) utility typically is used to label all of the files in filesystems that support EA security labeling. Package management tools such as RPM also may label files during installation, while system administrators often need to set security contexts manually with chcon(1) or setfilecon(1).

Evolution of SELinux Filesystem Labeling

The first release of SELinux in 2000 used a different mechanism for labeling filesystems than is used by the extended attributes approach discussed in this article. Persistent security IDs (PSIDs), integer representations of security context labels, were stored in an unused field of the ext2 inode. Mapping files on each filesystem were used by SELinux to look up a file's PSID by inode and then to map the PSID to a security context label.

This approach had the disadvantage of needing to modify each filesystem separately to support PSIDs. Thus, it was not a good general solution for extended security in the upstream kernel.

With the LSM Project, a generalized access control framework was implemented for the Linux kernel. As no filesystem-specific hooks are used in LSM, SELinux moved away from the modified filesystem approach and stored PSIDs in a normal file next to the mapping files. This allowed SELinux to be used purely as an LSM application with no kernel patching. It also allowed labeling to work for more filesystems, but it was not optimal in terms of performance and consistency. Accessing files from within the kernel remains generally problematic.

As part of the process of merging SELinux into the mainline kernel, with more feedback from the community, SELinux moved to the current filesystem labeling model based on extended attributes. Extended attributes provide applications with a standard API, eliminating the need for custom system calls to manipulate security labels, while filesystems also can be used similarly by other security modules, even on the same filesystem, using the separation provided by EA namespaces.

File Creation

When a file is created, a matching rule in the security policy typically describes how to assign a label based on the security contexts of the parent directory and the current task. Here's an example:

```
$ id -Z
```

```
root:staff_r:staff_t
$ ls -dZ /tmp
drwxrwxrwt+ root  root system_u:object_r:tmp_t /tmp
$ touch /tmp/hello
$ getfilecon /tmp/hello
/tmp/hello      root:object_r:staff_tmp_t
```

In this case, the security policy contains a rule that states files created by staff_t in a directory labeled tmp_t must be labeled with the type staff_tmp_t. If there is no explicit rule, files are labeled with the context of the parent directory.

Privileged applications can override the above-stated rule by writing a security context to / proc/self/attr/fscreate. This security context then is used to label any newly created files. The setfscreatecon(3) library function encapsulates this operation.

Unlabeled files may exist if a filesystem has not been labeled properly before use or if files are created on a filesystem when SELinux is not enabled. In case of the latter, the SELinux kernel internally assigns a default context to unlabeled files for AVC calls, but it does not attempt to relabel them on disk. To restore a security context label manually, use restorecon(8).

An fsck-like utility is being developed for managing the scenario where unlabeled files have been created. To be run on boot, this utility will ensure that all files are labeled correctly before the system enters multiuser mode.

Labeling Behaviors

In the preceding section, we discussed file labeling for filesystems that both support EAs on disk and have handlers for the EA security namespace. When such a filesystem is mounted normally, it is said to use xattr labeling behavior.

When a filesystem is initialized by SELinux, such as when it is being mounted, a log message is generated that reads:

```
SELinux: initialized (dev hda6, type ext3), uses xattr
```

The uses `xattr` clause means the filesystem uses the xattr labeling behavior described above.

Many filesystems do not support EAs, and of those that do, not all have security namespace handlers. For on-disk filesystems, it may be that nobody has done the coding work yet or that EAs simply don' t make sense for legacy filesystems such as vfat.

A proliferation of pseudo-filesystems have developed under Linux. Filesystems are becoming an increasingly favored user-kernel API mechanism. The most obvious of these is procfs, which is an interface between user space and various kernel components. Due to the long history of procfs, it has accumulated a lot of cruft, and new user-kernel filesystem APIs are encouraged to be implemented by way of separate filesystems. These filesystems are kernel resident and have no intrinsic EA support. Examples include usbfs, sysfs and selinuxfs.

Such non-EA cases are managed with a variety of labeling behaviors, according to rules in the security policy for each filesystem type.

The transition SIDs labeling behavior is used for devpts, tmpfs and shmfs filesystems. Files in these filesystems are labeled on demand in the kernel, based on the security contexts of the current task and a security context specified for the filesystem in policy.

devpts is a special-case transition SIDs filesystem. It provides EA API access to ptys by way of a dummy EA security handler. Privileged applications, such as sshd, use this feature to relabel ptys, overriding the transition SID labels.

The task SIDs labeling behavior simply labels the file with the same security context as the current task. It is used for pipes and sockets created in the pipefs and sockfs filesystems, respectively.

The genfs_contexts labeling behavior is used for filesystems unsuited to xattr, transition SIDs and task SIDs labeling. In the security policy, security context labels are assigned to filesystem/pathname pairs. The purpose of the pathname component is to allow finer-grained labeling of the filesystem. This feature is important for procfs in particular, which is a jumble of readable and writable kernel data, including the sysctl interface.

Most non-EA filesystems use genfs_contexts labeling, usually with the entire filesystem set to a single security context. Common examples include sysfs, vfat, nfs and usbdevfs.

Mountpoint Labeling

A new feature included with the 2.6.3 kernel is mountpoint labeling, also referred to as context mounts. The main purpose of this is to allow the security context of an entire filesystem to be specified by using a mount option. Mountpoint labeling can be applied to any type of filesystem and overrides its normal labeling behavior.

A specific use of mountpoint labeling is to allow different NFS mounts to be labeled separately at mount time. It also is useful for general ad hoc mounting of filesystems that do not support EA security labeling and for mounting EA-labeled filesystems labeled elsewhere. The latter may be important in forensic work, for example.

Legacy filesystems with no labels also may need to be mounted under an SELinux-enabled OS. Even though the filesystem type supports EA security labeling, we may not want to add persistent security context labels to these filesystems. Mountpoint labeling allows us to assign kernel-resident labels that are not written to disk.

As mountpoint labeling is a new feature and is not widely documented, let' sdiscuss it in a little more detail.

When SELinux is enabled in the kernel, three new mount options are provided for mountpoint labeling:

- context: causes every file on the filesystem, and the filesystem itself, to be labeled with the specified security context. The /proc/self/attr/fscreate API discussed above is ignored for the filesystem. This overrides existing labeling behavior, changing it to mountpoint labeling. Filesystem labels are read-only to the user with this option, although policy-specified labeling transitions still operate on filesystems with EA security labeling support.

- fscontext: sets the label of the aggregate filesystem (that is, the filesystem itself) to the specified security context. This allows finer-grained control of filesystems by allowing their labels to be set on a per-mount basis rather than on a per-fs type basis specified in a policy. As the context option also implements this functionality, the two options cannot be used together. This option works only for filesystems with EA security labeling support. Aggregate filesystem security contexts are used for access control decisions made during file creation within a specific filesystem, mounting and unmounting of filesystems, accessing filesystem attributes and relabeling the filesystem itself.

- defcontext: sets the default security context for unlabeled files, instead of the value specified in the policy. As with the fscontext option, it works only for filesystems with EA labeling support and is not valid if context has been specified, as it too implements this functionality.

In the kernel, SELinux parses and strips out the security mount options during mount(2), passing normal options through to filesystem-specific code. Normal filesystems do not need to be aware of the security options, thus, they do not need to be modified. This is possible because most filesystems use text name/value pairs for mount options, which SELinux is able to manipulate easily.

Filesystems with binary mount option data, including NFS, SMBFS, AFS and Coda, need to be handled as special cases. Of these, only NFSv3 is supported at this stage of SELinux development.

Here's an example of how the context option operates, as it is likely to be the most widely used of the three mount options. A floppy disk with log files has arrived on our desk, and we'd like to mount it on our SELinux box and run some log analysis software on it. Due to the way policy is configured, these files need to be labeled system_u:object_r:var_log_t for the log analysis software to work properly. Mounting in this fashion also can help provide a sandbox for the data on the floppy, allowing SELinux to protect the OS and the contents of the floppy from each other.

Let's mount the disk:

```
$ mount -v -t vfat
-o context=system_u:object_r:var_log_t
/dev/fd0 /mnt/floppy
/dev/fd0 on /mnt/floppy type vfat
(context=system_u:object_r:var_log_t)
```

What does the audit log say?

```
SELinux: initialized (dev fd0, type vfat), uses mountpoint labeling
```

This message looks promising. Next, we verify that the files on the disk are labeled as expected. Normally, you would use getfilecon(1), but getfattr(1) has more explicit error messages:

```
$ getfattr -n security.selinux /mnt/floppy/access_log
/mnt/floppy/access_log: security.selinux: Operation not supported
```

What is going on here? An ls -Z also shows that the file has a null security context:

```
$ ls -Z /mnt/floppy/access_log
-rwxr-xr-x+  root  root  (null)
/mnt/floppy/access_log
```

The vfat filesystem on the floppy does not have EA support, and its security context labeling occurs purely within the kernel. It turns out that this in-kernel labeling is working correctly, but the user-space tools are not able to view the labels in the EA API. This is a limitation of the current EA implementation that has yet to be resolved elegantly.

However, there's a sneaky way to see what the labels on the files are by using the audit log, which always records the security context of a target object when logging an access message.

The use of getfattr(1) caused the following audit record to be generated:

```
avc:  denied  { getattr } for  pid=12354 exe=/usr/bin/getfattr
```

```
path=/mnt/floppy/access_log dev=fd0 ino=132 scontext=root:staff_r:staff_t
tcontext=system_u:object_r:var_log_t tclass=file
```

So, the file is labeled correctly (system_u:object_r:var_log_t), per the context mount option passed to the mount command.

Future Work

Although it is possible to assign security context labels to NFS mounted filesystems, they operate only locally for access control decisions within the kernel. No labels are transmitted across the network with files. Work has been advancing in this area, with SELinux-specific modifications being made to the NFSv2/v3 protocols and code. Further down the track, NFSv4 integration is expected to involve labeling over the wire by way of named attributes, which are part of the more extensible NFSv4 specification. This would allow both the NFS client and server to implement SELinux security for networked files. Support for other networked filesystems also would be useful, as would interoperability with Trusted BSD' sSELinux port.

Backup and Restoration

One of the many tasks that change for system administrators using SELinux is backup and restoration. When creating an archive, how will the security context labels be preserved within the archive? The answer is to use the highly flexible star(1) utility, which has extended attribute support.

To manipulate archives with security context labels, use the xattr option. When creating archives, you also need to specify the exustar format. For example:

```
$ star -xattr -H=exustar -c -f cups-log.star /var/log/cups
```

creates an archive of the /var/log/cups directory, retaining security context labels on the files.

To extract, simply use the xattr option:

```
$ star -xattr -x -f cups-log.star
$ ls -Z var/log/cups/
-rw-r--r--+ root     sys      system_u:object_r:cupsd_log_t
error_log
-rw-r--r--+ root     sys      system_u:object_r:cupsd_log_t
error_log.1
```

As you can see, the security context labels have been preserved.

James Morris (jmorris@redhat.com) is a kernel hacker from Sydney, Australia, currently working for Red Hat in Boston. He is a kernel maintainer of SELinux, Networking and the Crypto API; an LSM developer and an Emeritus Netfilter Core Team member.