The Simplified Mandatory Access Control Kernel

Casey Schaufler
casey@schaufler-ca.com

Mandatory Access Control

Computer systems employ a variety of schemes to constrain how information is shared among the people and services using the machine. Some of these schemes allow the program or user to decide what other programs or users are allowed access to pieces of data. These schemes are called discretionary access control mechanisms because the access control is specified at the discretion of the user. Other schemes do not leave the decision regarding what a user or program can access up to users or programs. These schemes are called mandatory access control mechanisms because you don’t have a choice regarding the users or programs that have access to pieces of data.

Bell & LaPadula

From the middle of the 1980’s until the turn of the century Mandatory Access Control (MAC) was very closely associated with the Bell & LaPadula security model, a mathematical description of the United States Department of Defense policy for marking paper documents. MAC in this form enjoyed a following within the Capital Beltway and Scandinavian supercomputer centers but was often sited as failing to address general needs.

Domain Type Enforcement

Around the turn of the century Domain Type Enforcement (DTE) became popular. This scheme organizes users, programs, and data into domains that are protected from each other. This scheme has been widely deployed as a component of popular Linux distributions. The administrative overhead required to maintain this scheme and the detailed understanding of the whole system necessary to provide a secure domain mapping leads to the scheme being disabled or used in limited ways in the majority of cases.

Smack

Smack is a Mandatory Access Control mechanism designed to provide useful MAC while avoiding the pitfalls of its predecessors. The limitations of Bell & LaPadula are addressed by providing a scheme whereby access can be controlled according to the requirements of the system and its purpose rather than those imposed by an arcane government policy. The complexity of Domain Type Enforcement and avoided by defining access controls in terms of the access modes already in use.

Smack Terminology

The jargon used to talk about Smack will be familiar to those who have dealt with other MAC systems and shouldn't be too difficult for the uninitiated to pick up. There are four terms that are used in a specific way and that are especially important:

> ***Subject***: A subject is an active entity on the computer system. On Smack a subject is a task, which is in turn the basic unit of execution.
>
> ***Object***: An object is a passive entity on the computer system. On Smack files of all types, IPC, and tasks can be objects.
>
> ***Access***: Any attempt by a subject to put information into or get information from an object is an access.
>
> ***Label***: Data that identifies the Mandatory Access Control characteristics of a subject or an object.

These definitions are consistent with the traditional use in the security community. There are also some terms from Linux that are likely to crop up:

> ***Capability***: A task that possesses a capability has permission to violate an aspect of the system security policy, as identified by the specific capability. A task that possesses one or more capabilities is a privileged task, whereas a task with no capabilities is an unprivileged task.
>
> ***Privilege***: A task that is allowed to violate the system security policy is said to have privilege. As of this writing a task can have privilege either by possessing capabilities or by having an effective user of root.

Smack Basics

Smack is an extension to a Linux system. It enforces additional restrictions on what subjects can access which objects, based on the labels attached to each of the subject and the object.

Labels

Smack labels are ASCII character strings, one to twenty-three characters in length. Single character labels using special characters, that being anything other than a letter or digit, are reserved for use by the Smack development team. Smack labels are unstructured, case sensitive, and the only operation ever performed on them is comparison for equality. There are some predefined labels:

> _ Pronounced “floor”, a single underscore character.
> ^ Pronounced “hat”, a single circumflex character.
> * Pronounced “star”, a single asterisk character.
> ? Pronounced “huh”, a single question mark character.

Every task on a Smack system is assigned a label. System tasks, such as init(8) and systems daemons, are run with the floor ("_") label. User tasks are assigned labels according to the specification found in the /etc/smack/user configuration file.

Access Rules

Smack uses the traditional access modes of Linux. These modes are read, execute, write, and occasionally append. There are a few cases where the access mode may not be obvious. These include:

Signals: A signal is a write operation from the subject task to the object task.
Internet Domain IPC: Transmission of a packet is considered a write operation from the source task to the destination task.

Smack restricts access based on the label attached to a subject and the label attached to the object it is trying to access. The rules enforced are, in order:

1. Any access requested by a task labeled "*" is denied.
2. A read or execute access requested by a task labeled "^" is permitted.
3. A read or execute access requested on an object labeled "_" is permitted.
4. Any access requested on an object labeled "*" is permitted.
5. Any access requested by a task on an object with the same label is permitted.
6. Any access requested that is explicitly defined in the loaded rule set is permitted.
7. Any other access is denied.

In Figure 1 a user barney has been assigned the Smack label Rubble. This user can read or execute the floor labeled system programs and data. He can also read from and write to the special device /dev/null, which has the star label. System processes running with the floor label do not have any access to Barney's data. A system process running with the hat label is allowed read access to the user's data but not write access.

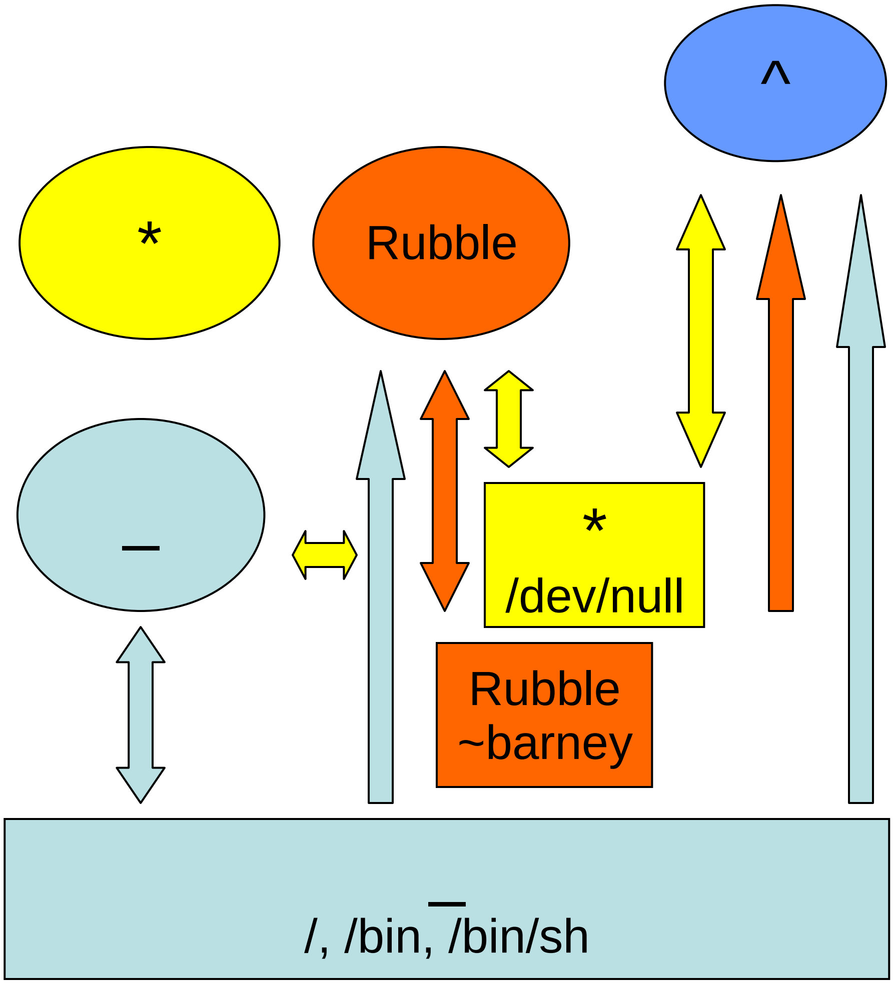

Figure 1 – Basic Smack Access Policy

With the basic rules system tasks running with the floor label are protected from user processes running with other labels. Figure 2 demonstrates the accesses allowed when two user labels are in use. In this example the `Java` labeled task can read and write the `Java` labeled data, the `MP3` labeled task has read and write access to the `MP3` labeled data, while the system floor labeled task has the same access to its floor labeled data. Both the `Java` and `MP3` tasks have read access to the floor system data. The two user tasks have no access to each other's data.

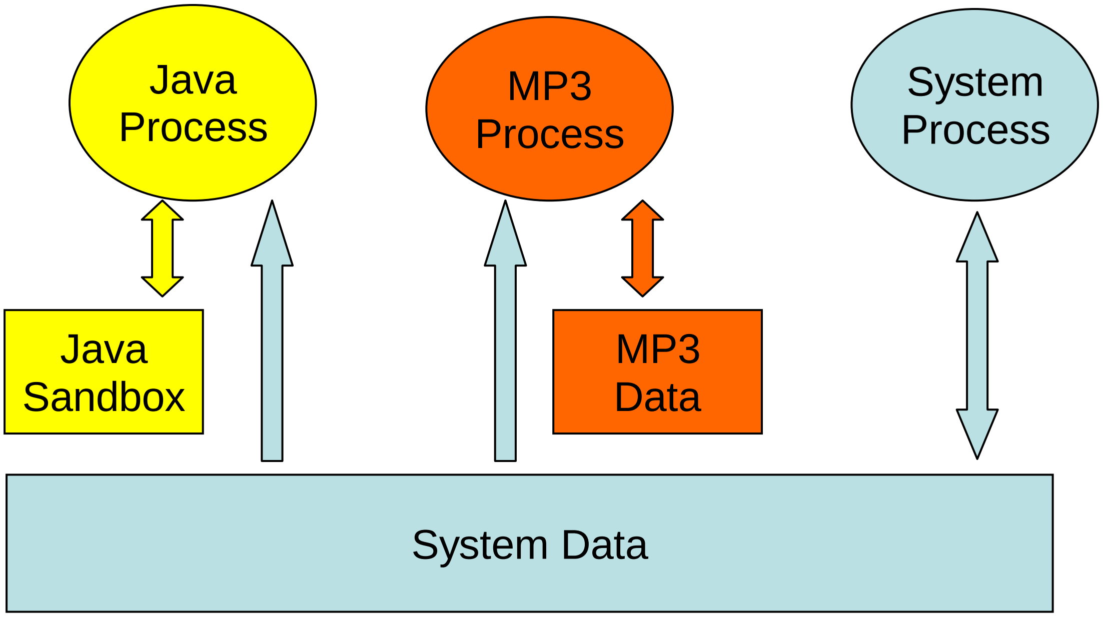

Figure 2 - Basic User Label Interactions

Smack Access Rules

With the isolation provided by Smack access separation is simple. There are many interesting cases where limited access by subjects to objects with different labels is desired. One example is the familiar spy model of sensitivity, where a scientist working on a highly classified project would be able to read documents of lower classifications and anything she writes will be "born" highly classified. To accommodate such schemes Smack includes a mechanism for specifying rules allowing access between labels.

Access Rule Format

The format of an access rule is:

```
subject-label object-label access
```

Where **`subject-label`** is the Smack label of the task, **`object-label`** is the Smack label of the thing being accessed, and **`access`** is a string specifying the sort of access allowed. The Smack labels are limited to 23 characters. The access specification is searched for letters that describe access modes:

a: indicates that append access should be granted.
r: indicates that read access should be granted.
w: indicates that write access should be granted.
x: indicates that execute access should be granted.

Access mode specifications can be in any order. Examples of acceptable rules are:

```
TopSecret Secret  rx
```

```
Secret    Unclass r
Manager   Game    x
User      HR      w
New       Old     rRrrr
Closed    Off     -
```

Examples of unacceptable rules are:

```
Top Secret     Secret    rx
TS/Alpha,Omega Overloard rx
Ace            Ace       r
Odd            spells    waxbeans
```

Spaces are not allowed in labels. The slash character "/" is not allowed in labels. Since a subject always has access to files with the same label specifying a rule for that case is pointless. Letters that do not specify legitimate access modes are not allowed.

Applying Access Rules

The developers of Linux rarely define new sorts of things, usually importing schemes and concepts from other systems. Most often, the other systems are variants of Unix. Unix has many endearing properties, but consistency of access control models is not one of them. Smack strives to treat accesses as uniformly as is sensible while keeping with the spirit of the underlying mechanism.

File system objects including files, directories, named pipes, symbolic links, and devices require access permissions that closely match those used by mode bit access. To open a file for reading read access is required on the file. To search a directory requires execute access. Creating a file with write access requires both read and write access on the containing directory. Deleting a file requires read and write access to the file and to the containing directory. It is possible that a user may be able to see that a file exists but not any of its attributes by the circumstance of having read access to the containing directory but not to the differently labeled file. This is an artifact of the file name being data in the directory, not a part of the file.

IPC objects, message queues, semaphore sets, and memory segments exist in flat namespaces and access requests are only required to match the object in question.

Process objects reflect tasks on the system and the Smack label used to access them is the same Smack label that the task would use for its own access attempts. Sending a signal via the `kill`() system call is a write operation from the signaler to the recipient. Debugging a process requires both reading and writing. Creating a new task is an internal operation that results in two tasks with identical Smack labels and requires no access checks.

Sockets are data structures attached to processes and sending a packet from one process to another requires that the sender have write access to the receiver. The receiver is not required to have read access to the sender.

Setting Access Rules

The configuration file /etc/smack/accesses contains the rules to be set at system startup. The contents are written to the special file /smack/load. Rules can be written to /smack/load at any time and take effect immediately. For any pair of subject and object labels there can be only one rule, with the most recently specified overriding any earlier specification.

In order to ensure that rules are written properly a program **smackload** is provided.

Task Attribute

The Smack label of a process can be read from /proc/<pid>/attr/current. A process can read its own Smack label from /proc/self/attr/current. A privileged process can change its own Smack label by writing to /proc/self/attr/current but not the label of another process.

File Attribute

The Smack label of a filesystem object is stored as an extended attribute named SMACK64 on the file. This attribute is in the security namespace. It can only be changed by a process with privilege.

Privilege

There are two capabilities used explicitly by Smack. CAP_MAC_ADMIN allows a process to perform administrative functions such as loading access rules. CAP_MAC_OVERRIDE exempts a process from all access control rules.

Smack Networking

As mentioned before, Smack enforces access control on network protocol transmissions. Usually a packet sent by a Smack process is tagged with its Smack label, however packets that would get the ambient label are sent without a tag. This is done by adding a CIPSO tag to the header of the IP packet. Each packet received is expected to have a CIPSO tag that identifies the label and if it lacks such a tag the network ambient label is assumed. Before the packet is delivered a check is made to determine that a subject with the label on the packet has write access to the receiving process and if that is not the case the packet is dropped.

CIPSO Configuration

It is normally unnecessary to specify the CIPSO configuration. The default values used by the system handle all internal cases. Smack will compose CIPSO label values to match the Smack labels being used without administrative intervention. Unlabeled packets that come into the system will be given the ambient label, and outgoing packets that would get the ambient label are sent unlabeled.

Smack requires configuration in the case where packets from a system that is not Smack that speaks CIPSO may be encountered. Usually this will be a Trusted Solaris™ system, but there are other, less widely deployed systems out there. CIPSO provides 3 important values, a Domain Of Interpretation (DOI), a level, and a category set with each packet. The DOI is intended to identify a group of systems that use compatible labeling schemes, and the DOI specified on the smack system must match that of the remote system or packets will be discarded. The DOI is 3 by default. The value can be read from /smack/doi and can be changed by writing to /smack/doi.

The label and category set are mapped to a Smack label as defined in /etc/smack/cipso.

A Smack/CIPSO mapping has the form:

```
smack level [category [category]…]
```

Smack does not expect the level or category sets to be related in any particular way and does not assume or assign accesses based on them. Some examples of mappings:

```
TopSecret 7
TS:A,B    7 1 2
SecBDE    5 4 6
RAFTERS   7 12 26
```

The “:” and “,” characters are permitted in a Smack label but have no special meaning.

The mapping of Smack labels to CIPSO values is defined by writing to /smack/cipso, and to ensure correct formatting the program **smackcipso** is provided.

In addition to explicit mappings Smack supports direct CIPSO mappings. One CIPSO level is used to indicate that the category set passed in the packet is in fact an encoding of the Smack label. The level used is 250 by default. The value can be read from /smack/direct and changed by writing to /smack/direct.

Socket Attributes

There are two attributes that are associated with sockets. These attributes can only be set by privileged tasks, but any task can read them for their own sockets.

> **SMACK64IPIN**: The Smack label of the task object. A privileged program that will enforce policy may set this to the star label.

SMACK64IPOUT: The Smack label transmitted with outgoing packets. A privileged program may set this to match the label of another task with which it hopes to communicate.

Packet Attributes

The Smack label that came with a network packet is obtained differently depending on the type of socket involved. Only a privileged process will ever need to do this, and then only if it is trusted to enforce the Smack access control rules.

For a `UDS` socket the label will match that of the file system object. It can be obtained by calling

```
fgetxattr(sock, "security.SMACK64", …).
```

The label of a `TCP` connection can be obtained by calling

```
getsockopt(sock, SOL_SOCKET, SO_PEERSEC, …)
```

The label used by a process should never change during a TCP session. It requires privilege to do so and a program that changes labels must do so with access control in mind.

The label of individual `UDP` packets must be dealt with as they come in, because there is no connection negotiated between the tasks. A program that wants to deal with incoming packets at multiple labels first needs to call

```
setsockopt(sock, SOL_IP, IP_PASSSEC, …)
```

and then parse the message headers with each packet received. The function `smackrecvmsg`() is available to provide the parsing. It can be used instead of `recvmsg`().

Writing Applications for Smack

There are three sorts of applications that will run on a Smack system. How an application interacts with Smack will determine what it will have to do to work properly under Smack.

Smack Ignorant Applications

By far the majority of applications have no reason whatever to care about the unique properties of Smack. Since invoking a program has no impact on the Smack label associated with the process the only concern likely to arise is whether the process has execute access to the program.

Smack Relevant Applications

Some programs can be improved by teaching them about Smack, but do not make any security decisions themselves. The utility `ls`(1) is one example of such a program.

Smack Enforcing Applications

These are special programs that not only know about Smack, but participate in the enforcement of system policy. In most cases these are the programs that set up user sessions. There are also network services that provide information to processes running with various labels.

File System Interfaces

Smack maintains labels on file system objects using extended attributes. The Smack label of a file, directory, or other file system object can be obtained using `getxattr`(2).

```
getxattr(“/”, “security.SMACK64”, value, sizeof
(value));
```

will put the Smack label of the root directory into value. A privileged process can set the Smack label of a file system object with `setxattr`(2).

```
rc = setxattr(“/foo”, “security.SMACK64”, “Rubble”,
strlen(“Rubble”), 0);
```

This will set the Smack label of `/foo` to `Rubble` if the program has appropriate privilege.

Socket Interfaces

The socket attributes can be read using `fgetxattr`(2). A privileged process can set the Smack label of outgoing packets with `fsetxattr`(2).

```
rc = fsetxattr(fd, “security.SMACK64IPOUT”, “Rubble”,
strlen(“Rubble”), 0);
```

This will set the Smack label “Rubble” on packets going out from the socket if the program has appropriate privilege.

```
rc = fsetxattr(fd, “security.SMACK64IPIN, “*”,
strlen(“*”), 0);
```

This will set the Smack label “`*`” as the object label against which incoming packets will be checked if the program has appropriate privilege.

Administration

Smack supports some mount options:

> **smackfsdef**=*label*: specifies the label to give files that lack the Smack label extended attribute.
> **smackfsroot**=*label*: specifies the label to assign the root of the file system if it lacks the Smack extended attribute.
> **smackfshat**=*label*: specifies a label that must have read access to all labels set on the filesystem. Not yet enforced.
> **smackfsfloor**=*label*: specifies a label to which all labels set on the filesystem must have read access. Not yet enforced.

These mount options apply to all file system types with the current exception of NFS.