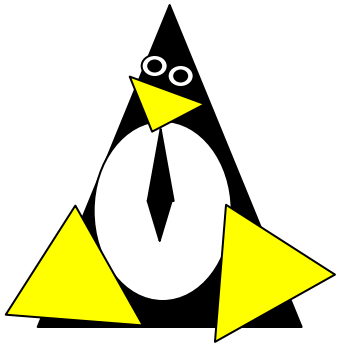

# System Services and Smack

Casey Schaufler
casey@schaufler-ca.com

## *Smack*

The Simplified Mandatory Access Control Kernel (Smack) provides a Mandatory Access Control (MAC) mechanism designed to be as simple as possible while retaining the flexibility required to meet modern system security needs.

## 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.

## System Services

System services are typically provided on well known network ports. Clients of these services connect to these well known ports to send requests and receive data or other responses. Services may require authentication or other protocol constraints but this is solely at the discretion of the service. The program providing the service is not affected by the security characteristics of the client. In fact it is generally very difficult for a server to determine the security characteristics of the client.

## The Smack System Services Approach

Smack is designed to address the security characteristics of processes. Other systems, including SELinux and AppArmor, are oriented around the security characteristics of programs. This is an important distinction for several reasons, but the most important is that a process based approach is better suited to the traditional separation and isolation requirements of a MAC system.

Smack recognizes three distinct models for providing services. These distinctions are based on the kind of information the server provides to the client, the security

implications of the service performed, and on how cognizant of MAC the server program may be.

The *single-label* server provides services to clients running with the same Smack label as the server. This is the "out of the box" case, where all processes run with the floor ("_") label and all files are defaulted to the same. Server applications that have not been made aware of Smack will usually be used in this way.

The *multiplexed* server uses multiple instances of single-label servers to provide the same service to clients running at different labels. A Smack cognizant utility, `smackpolyport`, is available to distribute requests for a service based on the Smack label of the clients making the requests. A special case of multiplexed servers allows a server that is not Smack cognizant to provide a service to any client.

The *Smack cognizant* server has been coded to identify the Smack label of incoming requests and enforce Smack policy itself. At this early stage in the development of Smack there are an understandably small number of Smack cognizant server programs.

## *Single Label Servers*

A single label server is useful is cases where the information being managed by the server is homogeneous with respect to its Smack label and there is no desire to share the information with processes at other labels. One such application could be a group database server.

The server need simply be started with the correct Smack label. One way this can be accomplished is for a user running at the desired label to invoke the server. Another is to use the (as yet unimplemented) `-l` (`--smack`) option of the `start-stop-daemon` utility.

**Single Label Server for one Smack label**

```
# cat /etc/init.d/cupsys

…

PIDFILE=/var/run/cups/$NAME.pid
DESC="Common Unix Printing System"
SMACK=Rabble

…

        start-stop-daemon --start -quiet -oknodo -pidfile "$PIDFILE" -
name "$NAME" --smack "$SMACK"

…
```

## *Multiplexed Servers*

A multiplexed server involves several instances of single label servers.

The `smackpolyport` utility provides mechanism to determine which requests are handled by which server. This utility is a general purpose Smack cognizant server whose function is to redistribute requests based on the Smack label of the connecting client. This allows the system to continue to use the well known port associated with the service to provide the service. It is of course also possible to eschew the well known port paradigm and advertise the label/port pairings for the service.

The `smackpolyport` utility deals strictly with ports, it does not invoke applications. A separate invocation is required for each client port. The simplest use is to provide a single server for all clients, regardless of their Smack labels. This is completely appropriate for certain kinds of system data.

**Single Server for all Smack labels**

```
# echo `cat /proc/self/attr/current`

# smackpolyport --client 3306 --master 12000:Public &

…

% echo `cat /proc/self/attr/current`
Public
% mysql --port=12000 shared-db &
```

Somewhat more complicated is the situation where the same service is provided with different servers. In this example two databases are served at different labels.

**Separate Server for each Smack label**

```
# echo `cat /proc/self/attr/current`

# smackpolyport --client 3306 --server 12000:Reptile --server
12001:Mammal &

…

% echo `cat /proc/self/attr/current`
Reptile
% mysql --port=12000 scale-db &

…

% echo `cat /proc/self/attr/current`
Mammal
% mysql --port=12001 fur-db &
```

Finally, there is the case where some labels will use a special service, while most will use common data.

**Separate Server for certain Smack labels**

```
# echo `cat /proc/self/attr/current`
_
# smackpolyport --client 3306 --master 12000:Public --server
12001:HoneyPot --server 12002:Expert &

…

% echo `cat /proc/self/attr/current`
Public
% mysql --port=12000 shared-db &

…

% echo `cat /proc/self/attr/current`
HoneyPot
% mysql --port=12001 misleading-db &

…

% echo `cat /proc/self/attr/current`
Expert
% mysql --port=12002 unfiltered-db &
```

## *Smack Cognizant Servers*

A Smack cognizant server is aware of MAC and makes its own decisions based on the MAC label of the client making requests.
A server application can be made Smack cognizant using a few simple functions. Some thought needs to go into the security implications of the services provided, and the detail involved in that analysis will depend on the data involved.

The client port needs to be configured to allow incoming requests with labels other than that of the server process. This is done by setting the input label of the socket to the star ("*") label.

**Single server socket for all Smack labels**

```
/*
 * Create the service socket for clients to connect to.
 */
if ((soc = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        perror("listen socket");
        exit(1);
}

sin.sin_family = AF_INET;
sin.sin_port = htons(CLIENT_PORT);
sin.sin_addr.s_addr = INADDR_ANY;

if (bind(soc, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        perror("listen bind");
        exit(1);
}

/*
 * Give the service socket the star label, allowing anyone
 * to talk to it.
 */
if (fsetxattr(soc, "security.SMACK64IPIN", "*", 2, 0) < 0) {
        perror("listen fsetxattr");
        exit(1);
}
```

Once the socket has been thus configured the burden of enforcing policy falls squarely on the program. The label of the client on an incoming connection can be obtained after the connection is accepted.

**Changing socket Smack labels**

```
slen = strlen(smack) + 1;
/*
 * Set the socket labels to match that of the server
 * to ensure communications. This requires CAP_MAC_ADMIN.
 */
if (fsetxattr(soc, "security.SMACK64IPIN", smack, slen, 0) < 0) {
        perror("server fsetxattr");
        exit(1);
}

if (fsetxattr(soc, "security.SMACK64IPOUT", smack, slen, 0) < 0) {
        perror("server fsetxattr");
        exit(1);
}
```

Actual Smack label checks that match those enforced by the kernel require that the current set of loaded rules be used. This can be accomplished using the `smackaccess` function provided in the `smack-util` package. This function reads and stores the rules from `/smack/load` and uses the same comparison scheme as the kernel code.

**Checking access based on Smack labels**

```
if (smackaccess(sub_smack, obj_smack, "rx") < 0)
        fprintf(stderr, "No read or execute access\n");

if (smackaccess(sub_smack, obj_smack, "wa") < 0)
        fprintf(stderr, "No write or append access\n");
```