Role Based Access Control and Secure Shell—A Closer Look At Two Solaris™ Operating Environment Security Features

Thomas M Chalfant,
Product Technical Support, Americas,
Enterprise Server Group,
Sun Microsystems, Inc.

Sun BluePrints™ OnLine—June 2003

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 817-3062-10
Revision 1.0, 6/4/03
Edition: June 2003

Please

Adobe PostScript

Role Based Access Control and Secure Shell

In the course of supporting customers we often get a chance to become familiar with system configurations and common practices used by many of our customers. Our customers are increasingly becoming more security conscious and are focusing more and more time implementing better security practices. To aid the customer in adopting better security practices, this article introduces and explains two security features in the Solaris™ operating environment. The first is Role Based Access Control and the second is Secure Shell. The goal is to provide you with enough information to make an effective decision to use or not use these features at your site as well as to address configuration and implementation topics.

Role Based Access Control (RBAC)

Role Based Access Control first became a visible feature in Trusted Solaris™ software and was later incorporated into the Solaris 8 operating environment. The Solaris 9 operating environment continues to improve upon the prior implementation. The feature set discussed in this article is that of the Solaris 9 12/02 release. With small differences, most of the information discussed can also be applied to prior Solaris releases.

RBAC takes a step back from the traditional UNIX administration model with an all powerful superuser called root, and augments this with a model that understands the security principle of least privilege. That is, not every person administering a system needs the all powerful root password to do their job, and users should only be given the minimal amount of privilege actually necessary to accomplish their given responsibilities. Their unique responsibilities become known as roles. Within the RBAC model, many roles can be created, each with its own specific capabilities

tailored to meet your security policy. This helps to provide a system management policy according to organizational needs rather than a necessity to grant the powerful root password outside the confines of a strict group of users.

A role has all the attributes of a normal user account including a name, unique UID, home directory, password, along with a specific set of unique capabilities. Users first log in as themselves, then assume a role by running the switch user (`su`) command. The login shell for a role is one of the profile shells: `/bin/pfsh`, `/bin/pfcsh`, or `/bin/pfksh`. These profile shells understand RBAC and implement the role capabilities as defined in the RBAC configuration files. Some common roles are the ability to shut down a host, mount a file system, and start or restart privileged daemons. Examples for implementing capabilities such as these are provided later in the article.

There are three roles that can easily be configured:

- Primary administrator—A powerful role that is equivalent to root.
- System administrator—A less strong role for administration that is not related to security. This role does not allow the user to set passwords.
- Operator—A junior administrator role for operations such as backups, restores, and printer management.

Other roles can be configured as required. Examples are included in the section "RBAC Examples" later in this article.

RBAC Configuration Files

There are several database files important to the configuration of RBAC. They can be located on the specific host or they can be located in the naming services databases via NIS, NIS+, or LDAP. As usual, the source and the search order comes from the file `/etc/nsswitch.conf`. This article does not attempt to provide all the details available in man pages for these files, but will specify the more useful information to get you started towards a quick understanding of RBAC configuration.

`/etc/user_attr`

The extended user attributes database defines users and roles, and most importantly, which users can assume which roles. It also specifies the profile used when assuming a particular role. The format of the file is:

```
user:qualifier:res1:res2:attr
```

where:

user	The name of the user as specified in the passwd(4) database.
qualifier	Reserved for future use.
res1	Reserved for future use.
res2	Reserved for future use.
attr	An optional list of semicolon-separated (;) key=value pairs that describe the security attributes to apply to the object upon execution. There are five valid keys: auths, profiles, roles, type, and project. Zero or more keys can be specified depending upon the desired result.

As you can see, only the user and attr fields are significant. The user field specifies a valid user in the /etc/password database whether this entry specifies a role or a user. Roles are assigned their own specific UID, and this user must have a valid home directory. Every user assuming this role will use this home directory. However, roles are differentiated from users in this file via the type key=value pair. An example will be provided later to clarify this. The profiles key=value pair relates to an entry by that value in the /etc/security /prof_attr and /etc/security/exec_attr databases. Note that while the auths key=value pair is listed here, the preferred use is in the /etc/security/prof_attr database file. This is because abstracting authorizations via execution profiles rather than the user attributes database can provide for better adaptability to changing security needs. Further, note that although profiles can be assigned directly to users, this practice is discouraged because users could make inadvertent mistakes by misuse of their privileges at an inappropriate time. Refer to the man page user_attr(4) for more details.

/etc/security/prof_attr

The execution profile description database links together the commands and authorizations needed to perform a specific function. If no specific authorizations are required, this file simply defines a profile to be used by a role. The format of this file is:

```
profname:res1:res2:desc:attr
```

where:

profname	The name of the profile. Profile names are case-sensitive.
res1	Reserved for future use.
res2	Reserved for future use.
desc	A long description. This field should explain the purpose of the profile, including what type of user would be interested in using it. The long description should be suitable for displaying in the help text of an application.
attr	An optional list of semicolon-separated (;) key=value pairs that describe the security attributes to apply to the object upon execution. There are three valid keys: help, profs, and auths. Zero or more keys can be specified depending upon the desired result.

Again this file looks complicated, but in reality is very simple. profname is a profile name used by the /etc/user_attr database described above. profs simply refers to other profiles described by other lines in this file, and serves as a way to nest the definition of a given profile. The auths key=value pair relates to an entry by that value in the /etc/security/auth_attr database. Refer to the man page prof_attr(4) for more details.

/etc/security/exec_attr

The execution profiles database defines the commands and security attributes that will be executed by a profile shell. Remember it is ideal to assign roles to users, profiles to roles, and specific commands and authorizations to profiles. The format of this file is:

```
name:policy:type:res1:res2:id:attr
```

where:

name	The name of the profile. Profile names are case-sensitive.
policy	The policy that is associated with the profile entry. It is always the text suser.
type	The type of object defined in the profile. It is always the text cmd.
res1	Reserved for future use.

`res2`	Reserved for future use.
`id`	A string that uniquely identifies the object described by the profile. For a profile of type `cmd`, the `id` is either the full path to the command or the asterisk (*) symbol, which is used to allow all commands. An asterisk that replaces the file name component in a path name indicates all files in a particular directory. To specify arguments, the path name should point to a shell script written to execute the command with the desired arguments.
`attr`	An optional list of semicolon-separated (;) key=value pairs that describe the security attributes to apply to the object upon execution. The list of valid key words depends on the policy enforced. The following key words are valid: `euid`, `uid`, `egid`, and `gid`. Zero or more keys can be specified depending on the desired result.

The commands `euid` and `uid` contain a single user name or a numeric user ID. Commands designated with `euid` run with the effective UID indicated, which is similar to setting the `setuid` bit on an executable file. Commands designated with `uid` run with both the real and effective UIDs. Setting `uid` might be more appropriate than setting the `euid` on privileged shell scripts.

The commands `egid` and `gid` contain a single group name or a numeric group ID. Commands designated with `egid` run with the effective GID indicated, which is similar to setting the `setgid` bit on a file. Commands designated with `gid` run with both the real and effective GIDs. Setting `gid` may be more appropriate than setting `guid` on privileged shell scripts. Refer to the man page `exec_attr`(4) for more details.

/etc/security/auth_attr

The authorization description database describes authorizations or rights that a user assuming a role will obtain. This is a method used by security aware programs, such that they change their behavior based upon the rights of users. If users have the appropriate rights, they are granted the particular restricted behavior. The Solaris™ Management Console suite of applications utilizes this functionality extensively. Refer to the man page `auth_attr`(4) for more details.

RBAC Examples

Setting Up a Shutdown Role

The first example shows the setup for allowing a user called `user1` to assume a role called `shutter` with an execution profile called `Shut`, which allows `user1` to perform the `/usr/sbin/shutdown` command only. This command ordinarily requires a real UID of 0 to perform this operation.

Here is the execution profile definition:

```
# grep Shut /etc/security/exec_attr
Shut:suser:cmd:::/usr/sbin/shutdown:uid=0;gid=1
```

Here is the execution profile description:

```
# grep Shut /etc/security/prof_attr
Shut:::System Shutdown:
```

Here are the user and role accounts definition:

```
# egrep "shutter|user1" /etc/passwd
shutter:x:9999:10::/export/home/shutter:/bin/pfsh
user1:x:5555:10::/export/home/user1:/bin/ksh
# egrep "shutter|user1" /etc/shadow
shutter:eqtR7XtSJg4zY:11268::::::
user1:awUD9lVUjuWUk:11268::::::
```

Here is the assignment of role `shutter` to user `user1`:

```
# grep shutter /etc/user_attr
shutter::::type=role;profiles=Shut
user1::::type=normal;roles=shutter
```

Note – Note: you may have to stop and restart name service cache daemon (nscd) to have this new role take effect.

```
# /etc/init.d/nscd stop
# /etc/init.d/nscd start
```

Log in to the system as user user1

```
$ ssh home1 -l user1
user1@home1's password:
Last login: Thu Mar 27 20:57:29 2003 from 192.168.2.22
Sun Microsystems Inc. SunOS 5.9 Generic May 2002
$ id
uid=5555(user1) gid=10(staff)
$ cat /etc/shadow
cat: cannot open /etc/shadow
```

User user1 does not have sufficient permissions to view the contents of /etc/shadow.

```
$ /usr/sbin/shutdown
/usr/sbin/shutdown: Only root can run /usr/sbin/shutdown
```

Equally, user1 does not have sufficient permissions to run /usr/bin/shutdown. So try the new role!

```
$ su - shutter
Password:
$ id
uid=9999(shutter) gid=10(staff)
$ cat /etc/shadow
cat: cannot open /etc/shadow
```

Role `shutter` still does not have permissions to view `/etc/shadow` even though it was assigned a real and effective UID of 0. Note, however, this permission was granted only for running the `/usr/sbin/shutdown` command. This is exactly as defined in the configuration files, honoring the concept of least privilege. Now let's try the shut down and see if it is successful under the new role.

```
$ /usr/sbin/shutdown
Shutdown started. Mon Mar 31 23:05:38 EST 2003
Broadcast Message from root (pts/6) on home1 Mon Mar 31
23:05:39...
The system home1 will be shut down in 1 minute
```

The shutdown continues.

With this simple example you can see how RBAC can provide very specific enhanced capabilities to a role without giving away anything over and above that specific granted permission.

Making root a Role

The next example shows how to set up root to be a role. This eliminates all direct root logins, and forces all users to first log in as themselves before assuming the role of root.

Find the root role in the `/etc/user_attr` file. By default it will look like the following:

```
$ grep root /etc/user_attr
root::::type=normal;auths=solaris.*,solaris.grant;profiles=All
```

Edit the file and change the type from normal to role. Also locate or add the user who should be able to assume the role of root. You should end up with the following:

```
$ grep root /etc/user_att
root::::type=role;auths=solaris.*,solaris.grant;profiles=All
user1::::type=normal;roles=shutter,root
```

As user `user1`:

```
$ roles
shutter,root
```

Editing RBAC Databases

The following commands can be used to edit the RBAC databases.

`smexec`(1M)	Manage entries in the `exec_attr` database
`smmultiuser`(1M)	Manage bulk operations on user entries
`smuser`(1M)	Manage user entries
`smprofile`(1M)	Manage profiles in the `prof_attr` and `exec_attr` databases
`smrole`(1M)	Manage roles and users in role accounts
`useradd`(1M)	Add new users
`usermod`(1M)	Change user files
`rolemod`(1M)	Change role files
`roledel`(1M)	Remove roles
`roleadd`(1M)	Add new roles
`smc`(1M)	Start the Solaris Management Console

With the knowledge already gained regarding the format of the underlying files used by RBAC, along with some basic system administration skills, these commands can be quickly put to use in scripts, etc., to avoid manual editing of the RBAC databases. Note that `smc` provides a very nice graphical user interface covering the functionality of RBAC configuration. Also, `smc` provides a method to invoke many system management tools that adhere to RBAC authorizations.

RBAC and Logging

RBAC is fully compatible with Solaris basic security model (BSM) auditing. The actions of a role are attributable to the user who assumed the role. The audit records include the identity of the user, the role, and the effective ID used for policy overrides. The audit event mask of the user is augmented by that of the role.

Sudo

Other methods to achieve the same goal as RBAC have achieved popularity. One such method is a program called Sudo, which is available on the Solaris Software Companion CD. So, the question to answer is why use RBAC if Sudo is so popular and made easily available by Sun Microsystems™? There are several reasons, actually, the first of which is support. Sun Microsystems supports RBAC, but only provides Sudo for your convenience on an AS IS, unsupported basis. This is a distinction worth noting. See the Sudo Freeware Legal Disclaimer in the "References" section of this document for more information. Another reason to prefer RBAC is that it is an integrated part of the OS. Further, RBAC configuration files can be included in the site's naming service such as NIS, NIS+, or LDAP, allowing for simple global management of RBAC roles within an enterprise. Finally, RBAC allows a finer grain control over the real and effective UIDs and GIDs for which a command executes under a role, specifically supporting the least privilege concept.

Secure Shell

Secure Shell is a program for logging into and executing commands on a remote machine. It is intended to replace less secure programs such as `rlogin`, `rsh`, `rcp`, and `telnet`. Its goal is to provide secure encrypted communications between two hosts over an insecure network. X11 connections and arbitrary TCP ports can also be forwarded over the secure channel. Another useful feature is session compression.

Secure Shell protects against:

- IP spoofing, where a remote host sends out packets that pretend to come from another trusted host. `ssh` even protects against a spoofing host on the local network pretending it is your router to the outside.
- IP source routing, where a host can pretend that an IP packet comes from another trusted host
- DNS spoofing, where an attacker forges name server records
- Interception of cleartext passwords and other data by intermediate hosts
- Manipulation of data by people in control of intermediate hosts
- Attacks based on listening to X authentication data and spoofed connection to the X11 server

The ssh program performs host and user identification using public/private key cryptography. Someone hostile who has taken over the network can force ssh to disconnect, but cannot decrypt or play back the traffic, nor hijack the connection. Additionally, as of the release of the Solaris 9 operating environment, ssh is now an integrated tool.

Given all the above, why aren't more sites using Secure Shell? Perhaps it is just a matter of education. The next few sections should help to rectify the situation. However, due to the large number of things you can do with ssh, this article is not able to explain them all in detail. Examples provided in a later section will demonstrate a few more popular uses.

Secure Shell Configuration Files

Since ssh uses many different configuration files, this can make it seem overly complex. This article does not list all the different files; refer to the man pages for ssh(1) and sshd(1M). Some specific files will be discussed as they are encountered during usage examples in the next section. One file you are sure to encounter on first use of ssh is:

`$HOME/.ssh/known_hosts` or `/etc/ssh_known_hosts`

The ssh program automatically maintains a list of all known hosts it has ever been used with. The system administrator can predefine hosts for you in the `/etc/ssh_known_hosts` file or new hosts will be added to the user's `$HOME/.ssh/known_hosts` file automatically during use. After using a given host, future invocations check these files for the host. If the host has changed, a warning is provided and password authentication is disabled. This prevents certain types of attacks from being successful.

Secure Shell Examples

The following sections illustrate some popular uses for Secure Shell.

Simple ssh From One Host to Another

This example demonstrates use of ssh for telnet replacement.

```
$ ssh home1
user1@home1's password: xxxxx
Last login: Thu Mar 27 20:57:29 2003 from home2
Sun Microsystems Inc. SunOS 5.9 Generic May 2002
$
```

Using ssh to Forward the X11 Protocol

This example demonstrates use of ssh to run an X11 based application on a remote host and display the application on the local host's display. This example assumes you are sitting at a system running an X11 server.

```
$ ssh -f home1 /usr/openwin/bin/xterm
user1@home1's password: xxxxx
$
```

The xterm appears. The -f option tells ssh to background itself after prompting for any required authentication information. Note that you might need the -X option if X11 forwarding has not been enabled in the ssh client configuration file. Further, note that X11 forwarding must be enabled in the sshd server configuration file as well. The ssh program takes care of setting up your DISPLAY environment variable appropriately along with configuring appropriate X11 server authorizations on the local hosts.

Identity Key Generation

One of the features of ssh is its ability to utilize public/private key cryptography for both server and user identification. If you have been curious while reading this paper and tried ssh on a system, you have already seen an example of server identification. When an ssh client connects to a server, the server presents its public host key and the client compares it to its own copy stored in the known_hosts file as discussed in the configuration file section. If there is a difference, the user must

decide the appropriate action to take. Like hosts, users can have public/private key cryptographic identities. This section discusses the user identification facet of Secure Shell.

```
$ ssh-keygen -t rsa
Enter file in which to save the key(/home/user1/.ssh/id_rsa):
Generating public/private rsa key pair.
Enter passphrase(empty for no passphrase): xxxxx
Enter same passphrase again: xxxxx
Your identification has been saved in /home/user1/.ssh/id_rsa.
Your public key has been saved in /home/user1/.ssh/id_rsa.pub.
The key fingerprint is:
md5 1024 d5:87:b0:23:33:ce:71:b3:f0:55:b6:b0:0d:00:77:74 user1@home1
```

This generates an RSA identity stored in user1's `.ssh` directory. To use the identity, add the contents of the `id_rsa.pub` file to the remote host's `~/.ssh/authorized_keys` file.

A subsequent `ssh` to this host will look somewhat different since it will attempt to use the RSA identity rather than the normal user password. You are then prompted for the passphrase as follows:

```
$ ssh home2
Enter passphrase for key '/home/user1/.ssh/id_rsa': xxxxx
Last login: Thu Apr 10 16:01:06 2003 from home1
Sun Microsystems Inc. SunOS 5.9 Generic May 2002
$
```

Authentication Agents

The above example was fine, but you still need to provide your passphrase rather than your password each time to initiate a connection. Using an `ssh` authentication agent, you can implement a form of single sign-on and you will not have to provide the passphrase every time you run `ssh`.

Assuming you have set up RSA identities as shown above, do the following on the client host:

```
$ eval 'ssh-agent'
Agent pid 5737
$
```

When the ssh agent runs outside the eval, it outputs shell variables for use by future invocations of ssh. Running inside the eval allows the shell variables to take effect on the current shell and future subshells. These shell variables tell the future ssh invocations to utilize this agent. The next step is to add keys for use by this agent.

```
$ ssh-add
Enter passphrase for /home/user1/.ssh/id_rsa: xxxxx
Identity added: /home/user1/.ssh/id_rsa(/home/user1/.ssh/id_rsa)
$
```

Now retry an ssh like before:

```
$ ssh home1
Last login: Thu Apr 10 16:04:27 2003 from home1
Sun Microsystems Inc. SunOS 5.9 Generic May 2002
$
```

Note that no password or passphrase was required, which provides our single sign-on capability. It is possible to run the agent on a different host than the client host if a more secure system is available, but this is left as an exercise for the reader.

TCP Port Forwarding

Consider the following scenario. You are currently logged into host home1 which does not have internet access. It can, however, access a remote host home2 which, in turn, can access the internet via an http proxy on yet another host http_proxy_host on port 8080. The following ssh command will set up a tunnel to home2, and tell it to port forward a local port from home1 over the tunnel and to the http proxy.

From host home1:

```
$ ssh -L
8080:http_proxy_host:8080 home2
user1@home2's password: xxxxx
$
```

Now from your browser on host home1, set your proxy to localhost:8080 and you'll really use the proxy server accessible from home2.

Summary

From the explanations and examples provided in this article, hopefully you will see the benefits of RBAC and Secure Shell and begin to utilize them within your managed environment. This article represents a whirlwind tour of the features of RBAC and Secure Shell, and cannot portray all the features and configuration details in this short forum. The following references will help you to explore these topics in greater detail.

References

- Sun Microsystems Sudo Freeware Legal Disclaimer
  `http://wwws.sun.com/software/solaris/freeware/index.html#disclaimer`
- Sudo Homepage
  `http://www.courtesan.com/sudo/sudo.html`
- `ssh` Frequently Asked Questions
  `http://www.employees.org/~satch/ssh/faq/ssh-faq.html`
- RBAC in the Solaris™ Operating Environment
  `http://wwws.sun.com/software/whitepapers/wp-rbac/wp-rbac.pdf`
- System Administration Guide: Security Services, document # 816-4883-10
  `http://docs.sun.com`
- Noordergraaf, Alex and Watson, Keith. “Solaris™ Operating Environment Security: Updated for the Solaris 9 Operating Environment,” Sun BluePrints Online, December 2002:
  `http://www.sun.com/blueprints/1202/816-5242.pdf`
- National Institute of Standards and Technology on RBAC
  `http://csrc.nist.gov/rbac/`

Acknowledgments

I would like to thank my peers at Sun Microsystems for their time and effort spent in review of this document. I would also like to thank my management for allowing me to step outside the normal topics for my group and produce a document that hopefully is useful to a larger population of users.

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html`