Getting Started with SEBSD HOWTO

Last update: Nov 29, 2005

This document is a general introduction about Security Enhanced BSD (SEBSD). It is based on the experimental version of SEBSD, 20050624-6.0-SEBSD, and might be also applied to the future FreeBSD 6.0 with MAC framework and SEBSD enabled. The document is for people who want to start using or testing SEBSD. Some content here is derived from "Getting Started with SELinux HOWTO" by Faye Coker for SELinux. For more information about SEBSD and its latest status, see http://www.trustedbsd.org/sebsd.html.

---

1. Introduction

1.1 SEBSD

SEBSD is a port of the SELinux Flask and Type Enforcement implementations to run on top of the TrustedBSD MAC framework. SEBSD is occurring as part of an extension to the DARPA CBOSS contract at NAI Labs, and provides access to NSA FLASK and the Type Enforcement implementation as pluggable module on FreeBSD 5.0 or above.

The Flask architecture provides flexible support for a wide range of access control policies by separating the policy enforcement from the policy decision. Its implementation under Linux, SELinux, has achieved great success and been merged into official Fedora and other popular Linux distributions. For more information about Flask and SELinux, see http://www.nsa.gov/selinux.

1.2 Feedback

Any comments and suggestions on this document are welcome. Please email to wuyanjun <at> gmail <dot> com.

1.3 Disclaimer

This document is a user guide only. The author will take no responsibility for any damage cause by using it. We will try our best to update it and make it more correct. Anyway, we strongly recommend you install SEBSD and configure it well on a test machine before deploying it on a production server.

---

2. Overview

This section will discuss the following matters: on which condition you might consider SEBSD, some features provided by SEBSD, and several terms that will be frequently used in subsequent sections.

2.1 When should you consider SEBSD?

If you are a BSD fan, and at the meantime you are curious about OS security, then you might choose to use SEBSD as a starting point. It will provide you broad and deep knowledge.

If you are an administrator maintaining a BSD server in a security-critical company, and the server is used by multiple users and has high, complex security requirements, you might consider SEBSD as the basis of your security management.

2.2 Features of SEBSD

Comparing to the traditional access control mechanisms such as DAC (Discretionary Access Control) and Posix.1e Capabilities, SEBSD offers greater security for systems. It provides fine-grained mandatory access control mechanisms together with flexible configuration and management. Through IBAC(Identity Based Access Control), RBAC(Role Based Access Control) and TE (Type Enforcement) models, users are assigned predefined roles, files and processes belong to certain types. A user can access a file or a process only if one of his/her roles has permission to access the type of file/process.

This differs from regular Unix permission in that the user defined roles, or security contexts they are placed in, have limited access to files and other resources, but in a far more control fashion. For example, take a user's .rhosts file on a regular Unix System. If the owner makes it world writable (by mistake or by Trojans), then anyone can login and do lots of damage. Under SEBSD, you can control whether or not the user has the ability to change the permissions on their .rhosts file, and also prevent other people from writing to it even the owner has made it world writable.

A common question is how SEBSD permissions relate to standard Unix permissions. When you do a certain operation, the Unix permissions are checked first. If they allow your operation then SEBSD will check next and allow or deny as appropriate. But if the Unix permissions don't let you do it, the requested operation stops there and SEBSD checks will not be performed.

Another example is if there was an exploitable bug in /usr/bin/passwd which could run chmod 666 /etc/master.passwd. SEBSD permissions would still prevent anyone from inappropriately accessing the file.

2.3 Terminologies

The following terms will appear frequently in this document, and form the core concepts SEBSD.

2.3.1 identity

An identity under SEBSD is not the same as the traditional Unix UID. They can coexist together on the same system, but are independent and quite different from each other. Identities under SEBSD stands for part of a security context which will affect what domains can be entered, i.e. what essentially can be done. A SEBSD identity and a standard Unix login may have the same textual representation (and in most cases they do), however it is important to understand that they are two different things. Running the su command does not change the user's identity under SEBSD.

Example:
A normal user with the login name *wyj* runs the “id –M” command (under SEBSD):

```
wyj@wyj_sebsd$ id -M
sebsd/wyj:user_r:user_t
```

The identity portion of the security context in this case is "wyj". Now if *wyj* has su'ed to root and runs id, he will the security context is still wyj:user_r:user_t.

```
wyj@wyj_sebsd$ su -
Password:
root@wyj_sebsd# id -M
sebsd/wyj:user_r:user_t
```

So the identity remains the same, and has not changed to root. However, if identity *wyj* has been granted access to enter the staff_r role and does so (with sebsd_newrole command which will be covered later), and runs id command again, he will now see:

```
wyj@wyj_sebsd$ id -M
sebsd/wyj:staff_r:staff_t
```

So the identity remains the same but the role and domain (second and third fields respectively) have changed. Maintaining the identity in this manner is useful where user accountability is required. It is also crucial to system security in that the user identity will determine what roles and domains can be used.

2.3.2 domain

Every process runs in a domain. A domain directly determines the permissions a process has. A domain is basically a list of what processes can do, or what actions a process can perform on different types. Think of a domain like a standard Unix uid. Say root has a program and does a chmod 4777 on that program (making it setuid root). Anyone on the system, even the *nobody* user, can run this program as root thereby creating a security issue. With SEBSD however, if you have a process which triggers a domain transition to a privileged domain, if the role of the process is not authorized to enter a particular domain, then the program cannot be run.

Some examples of domains are sysadm_t which is the system administration domain, staff_t which is the user domain with access to the sysadm_r role, and user_t which is the general unprivileged user domain. *init* runs in the init_t domain, and *named* runs in the named_t domain.

2.3.3 type

A type is assigned to an object and indicates what actions can be performed on this object. The definitions for domain and type are the same, except domain applies to processes while type applies to objects such as files, directories or sockets, etc.

2.3.4 role

A role determines what domains can be entered. A user is assigned several roles in policy configuration files.

Example: In order to allow a user from the user_t domain (the unprivileged user domain) to execute the passwd command, the following should be specified in the relevant configuration file:

```
role user_r types user_passwd_t
```

It means that a user in the user role (user_r) is allowed to enter the user_passwd_t domain, i.e. he/she can run the passwd command.

2.3.5 security context

A security context has all the attributes that are associated with things like files, directories, processes, sockets and so on. It is made up of the identity, role and domain or type. You can check your own current security context by running “id –M” under SEBSD.

```
wyj@wyj_sebsd$ id -M
sebsd/wyj:user_r:user_t
```

The prefix “sebsd” means this is a SEBSD label and the string after “/” is the SEBSD

security context.

There is a very important distinction that needs to be made clear here, between domain and type, as both of them are in the third field, and it tends to cause a little confusion later on if you do not understand it from the start.

Processes have a domain. When you check the security context of a process (for an example, see the explanation of "transition" below), the final field is the domain such as user_passwd_t (if you were running the passwd command).

Objects such as files, directories, sockets etc have types. When you use the ls -Z command on a file for instance, the final field is the type, such as user_home_t for a file created in the home directory of a user in the user_r role.

The security context of a file can vary depending on the domain that creates it. By default, a new file or directory inherits the same type as its parent directory; however you can have other choices through policies.

Example: User *wyj* creates a file named test in his directory. He then runs the command ls -Z test and sees:

```
wyj@wyj_sebsd$ touch test
wyj@wyj_sebsd$ ls -Z test
sebsd/wyj:object_r:user_home_t test
```

He then creates a file in /tmp also called test, and runs the command ls -Z /tmp/test, the result is:

```
wyj@wyj_sebsd$ cd /tmp/
wyj@wyj_sebsd$ touch test
wyj@wyj_sebsd$ ls -Z test
sebsd/wyj:object_r:user_tmp_t test
```

In the first example, the security context includes the type "user_home_t" which is the default type for the home directory of an unprivileged user in the user_r role. After running the second ls -Z command, you can see that the type is user_tmp_t which is the default type that is used for files created by a user_t process, in a directory with a tmp_t type.

2.3.6 transition

A transition decision also referred to as a labeling decision, determines which security context will be assigned for a requested operation. There are two main types of transition. Firstly, there is a transition of process domain which is used when you execute a program of a specified type. Secondly, there is a transition of file type happened when you create a file under a particular directory.

Example:
For the second type of transition (transition of file type), refer to the example give in the "security context" section above. When running the ls -Z command you can see what the file types are labelled as (i.e. user_home_t and user_tmp_t in the above examples).

For transition of process domains, consider the following example. Run ssh as a non privileged user, or more specifically, from the user_t domain (remember you can use the id command to check your security context). Then run “ps -Z |grep ssh” and note what is listed for ssh. Assuming user *wyj* does this, he will see:

```
wyj@wyj_sebsd$ ps -Z|grep ssh
sebsd/wyj:user_r:user_t            496    p0    R+          0:00.00 grep ssh
sebsd/wyj:user_r:user_ssh_t        482    v2    I+          0:00.03 ssh 192.168.0.5 -l wyj
```

At the last line of the output listing, the ssh process is being run in the user_ssh_t domain because the executable is of type ssh_exec_t and the user_r has been granted access to the user_ssh_t domain.

2.3.7 policy

Policies are a set of rules governing things such as the roles a user has access to; which roles can enter which domains; and which domains can access which types with what mode. You can edit your policy files according to how you want your system set up.

Because the default location for SEBSD policy is /etc/security/sebsd/policy, and we will use it frequently hereafter, let us take $POLICY_HOME to make it short.

---

3. Installation

The following instruction will use 20050624-6.0-SEBSD to explain how to install a new SEBSD. It is available at http://www.trustedbsd.org/downloads. It is assumed you know how to install a normal FreeBSD system, including packages, distributions and ports, and how to configure, compile and boot a new SEBSD kernel. If you are not familiar with that knowledge, see FreeBSD handbook for details.

3.1 Basic Installation Step

Here we assume you have the installation CD on hand. This CD will install the complete operating system, including kernels, user applications, and complete kernel

source code. As a common FreeBSD sysinstall interface, a series of menus will prompt the user how to proceed.

a) At the main menu, select an installation method, typically, the standard installation is adequate. The remainder of these instructions assumes the standard installation option was selected.
b) The next menu displays the disk partition manager. As long as the installation machine will be dedicated to SEBSD, allow the partition manager to use the entire disk by selecting 'A'. Select 'Q' to exit the partition manager. The installation program may print a warning that this creates a dedicated machine. It will proceed to ask which boot manager to install; select 'BootMgr' to install the normal FreeBSD boot manager on this hard disk.

   Note: Take great care if you have multiple operating systems on the same disk and you want to keep them live together with SEBSD. For the issue, you'd better to ask experts for help unless you have enough experiences. What's more, remember the system should be installed on the primary partition as this is the general requirements of BSD system.
c) The next menu will label the disk to create swap space and individual file systems. Selecting 'A' will use the default values. Select 'Q' to proceed to the next menu.
d) The next menu selects the distributions to install. The 'Developer' option is recommended. X Window support is not included on this installation CD, and may be installed later. Likewise, the optional ports collection is not include on the SEBSD installation CD.
e) On the next screen, Select CD/DVD from the installation media menu.
f) Confirm installation. WARNING: With the configuration recommended in these instructions, all existing data on the hard disk will be destroyed!
g) SEBSD will be installed on the machine. Once complete, the installation program will ask a series of questions to help configure the new system. Answer these questions as appropriate.

Reboot the system when prompted. By default, the system will boot the MAC kernel and load the SEBSD security module (with the default policy). The file systems have not yet been labeled, and many warnings will be printed to the system console. If it is necessary to boot the generic FreeBSD kernel (without the MAC framework), comment out the following lines in /boot/loader.conf:

```
kernel="MAC"
sebsd_load="YES"
```

Alternatively, the kernel and modules to load may be selected from the FreeBSD boot loader. Refer to the FreeBSD handbook for more information on the boot loader.

3.2 Modify and Load Security Policies

The system comes pre-installed with a sample policy, but local changes might be

required. The policy source is located in /etc/security/sebsd/policy and the compiled (binary) version is installed in /etc/security/sebsd/policy/policy.bin by default. Only the binary version is loaded by the SEBSD module at boot time. An alternate location for the binary policy file may be specified at the boot loader or in /boot/loader.conf. The boot loader uses a symbolic link "policy.bin" in the same directory as policy.17, in case a different policy file version is necessary.

Since SEBSD uses the same policy language as SELinux, the SELinux report titled, "Configuring the SELinux Policy", (available at the SELinux project web site: http://www.nsa.gov/selinux/) can provide useful information. If you make changes to the policy source, you must re-install the modified binary policy:

```
cd /etc/security/sebsd/policy && make install
```

If changes were made to the policy, the modified version must be loaded into the kernel. The /sbin/sebsd_loadpolicy program can be used instead of a reboot:

```
/sbin/sebsd_loadpolicy /etc/security/sebsd/policy/policy.bin
```

or you can simply type “make load” to load the default policy file.

Note that policy.bin is installed by default as a symlink to another file. If you plan to generate your own policy file then you might need to adjust this.

3.3 Label the File System

Labeling file system is to set a security context for each persistent (on-disk) object (file, directory, etc). The security context is stored in the extended attribute. By default, extended attribute support was enabled during the install, but the individual files were not labeled. To label all file systems, login as root and run the following command:

```
cd /etc/security/sebsd/policy && make relabel
```

It will take quite a few minutes according to the number of files on the disk.

3.4 Permissive Mode and Enforcing Mode

Reboot the machine, so that applications can use the file labels and will be started in the correct domains. At this point, the machine will be running SEBSD with the sample policy. The sample policy is only an example and must be customized. Furthermore, the sample policy is not complete, so the system will print some access control warnings. By default, the system is configured in the development mode, or permissive mode; in this mode, access control failures are logged but not enforced. To toggle between enforcing mode and permissive mode, you can use sysctl to set the value of security.mac.sebsd.enforcing as follows:

To enable: sysctl security.mac.sebsd.enforcing=1
To disable: sysctl security.mac.sebsd.enforcing=0

Note that with the sample policy, only root running in the sysadm_r role is permitted to toggle the enforcement state.

If you would like the machine to default to enforcing mode at boot time, you may specify a default value for this sysctl in /etc/sysctl.conf. Uncomment the following line at the end of the file:

```
security.mac.sebsd.enforcing=1
```

There is also a useful switch for you to debug your policies. Turn on the verbose functionality by the following command:

sysctl security.mac.sebsd.verbose = 2 (or any positive integer larger than 1)

you will see many messages about SID assignment, file path and security context, etc.

---

4 Add a New Account

When the installation is completed, the thing you would like to do is to add a new account. This section will introduce how to create a new user and assign him the roles and default security context.

4.1 Create a New Account

Log in as root and you will see following message if you use the default policy:
Your default SEBSD context is root:sysadm_r:sysadm_t.
Do you want to choose a different one? [n]:

Here you need to choose the logging context for root. We will explain it later. Just press enter and log in with the default context root:sysadm_r:sysadm_t. Only in this context root can add a new user. Let us use “adduser” command to add a new user called *setest*:

```
root@wyj_sebsd# adduser
Username: setest
Full name: sebsd test
Uid (Leave empty for default):
Login group [setest]:
Login group is setest. Invite setest into other groups? []:
Login class [default]:
Shell (sh csh tcsh bash nologin) [sh]:
Home directory [/home/setest]:
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
```

```
Enter password:
Enter password again:
Lock out the account after creation? [no]:
Username       : setest
Password       : *****
Full Name      : sebsd test
Uid            : 1001
Class          :
Groups         : setest
Home           : /home/setest
Shell          : /bin/sh
Locked         : no
OK? (yes/no):yes
```

Now a new user is created successfully.

4.2 Assign Roles to User

We must assign roles to the new user. Here we expect he has at least user_r role. The relevant configuration file is /etc/security/sebsd/policy/user. Open it with vi, add the following line at the end of the file:

```
user setest roles { user_r };
```

It means the user *setest* is authorized to have the role user_r. If you want *setest* have more roles like sysadm_r, you can add the following line instead:

```
user setest roles { user_r sysadm_r };
```

To apply the modification, run “make load” under /etc/security/sebsd/policy directory by root with root:sysadm_r:sysadm_t context.

By default, a new user will have the role user_r, without necessarily specifying in users file. However, on the following conditions you must explicitly specify it in the configuration file:

1) the user has more roles than user_r;
2) the user is able to change his password.
3) have the SEBSD log explicitly contain the user name where applicable.

Next we will set the security context for *setest*.

4.3 Set Default Security Context for User

After adding *setest* in /etc/security/sebsd/policy/users, we need to assign the default security context for user’s login session. The relevant configuration file is appconfig/default_context. Open it and you will see a line like:

```
system_r:local_login_t    user_r:user_t
```

It means when a user logs in locally, the /bin/login program will run in local_login_t domain, and will then assign user with user_r role and let the user enter user_t domain.

If the line is like:

```
system_r:local_login_t    user_r:user_t staff_r:staff_t
```

It means the user is also authorized to enter staff_r role and staff_t domain.

There is another line like:

```
system_r:sshd_t user_r:user_t staff_r:staff_t
```

It means any user that logs in through ssh will enter user_r:user_t or staff_r:staff_t.

Recall the message appears in Section 4.1. In /etc/security/sebsd/policy/users, there is a line for root:

```
user root roles { staff_r sysadm_r ifdef(`direct_sysadm_daemon', `system_r') };
```

We can see root is authorized to have staff_r and sysadm_r roles, together with a conditional role system_r (Here we will not introduce the conditional expression). These two roles have the default types (specified in appconfig/default_type) staff_t and sysadm_t respectively. Hence the root user has two security contexts to choose when he logs in:

[1] root:sysadm_r:sysadm_t
[2] root:staff_r:staff_t

4.4 Label User's Home Directory

After adding the *setest* user, we need to label the user's home directory. If we forget to do so, the user will not be able to login to his own home directory in enforcing mode. The home directory for *setest* is /usr/home/setest. Use the following command to label the files and sub directories under it:

```
setfmac –R sebsd/setest:object_r:user_home_t /usr/home/setest
```

The command recursively labels all the files and directories under /usr/home/setest with security context setest:object_r:user_home_t. See setfmac(8) for detailed usage of setfmac.

Sometimes a process needs to access a user's home directory, but not to any files or subdirectories below. hence we should label /usr/home/setest itself with the different type, like the following:

```
setfmac sebsd/setest:object_r:user_home_dir_t /usr/home/setest
```

---

5 Choosing a Security Context

This section will introduce how user selects the security context and how to change one to the other. We also show the different rights the user has when he associates with the different security context.

The user with multiple roles authorized has two ways to choose his security context: through log in or through sebsd_newrole utility.

Following the instruction of Section 4, let us create a new user called *wyj*. Add a line to the /etc/security/sebsd/policy/users and assign *wyj* with two roles:

```
user wyj roles { user_r staff_r };
```

Remember to label *wyj*'s home directory as described in Section 4.4.

5.1 Choosing a Security Context When Log In

When the user *wyj* logs in, the following will prompt:

```
Your default SEBSD context is wyj:user_r:user_t.
Do you want to choose a different one? [n]: y
```

Press "y", then you will see:

```
[1] wyj:user_r:user_t
[2] wyj:staff_r:staff_t
Enter number of choice: 1
```

Here *wyj* can choose one context to login. Suppose he choose 1 to log in, then run "id –M", he will see:

```
wyj@wyj_sebsd$ id -M
sebsd/wyj:user_r:user_t
```

5.2 Change Security Context Through sebsd_newrole

SEBSD provides a utility called *sebsd_newrole*, by which a user can execute his shell with a new role, and then change to a new security context. The usage of *sebsd_newrole* is:

```
sebsd_newrole -r role [ -t type ] [ args ]
```

For details see sebsd_newrole(1).

User can change his current role to another role in case both are authorized and the role transition from the source to the destination is allowed.

For example, *wyj* logged in as wyj:user_r:user_t. Now he wants to change to wyj:staff_r:staff_t. Run the following command:

```
wyj@wyj_sebsd$ sebsd_newrole -r staff_r -t staff_t
Authenticating wyj.
Password:
Executing default shell (/usr/local/bin/bash) with context wyj:staff_r:staff_t
```

After that, run “id –M”:

```
wyj@wyj_sebsd$ id -M
sebsd/wyj:staff_r:staff_t
```

The security context of *wyj* has changed to wyj:staff_r:staff_t.

Now assume *wyj* wants to change to wyj:sysadm_r:sysadm_t in which the sysadm_r is not authorized to him. If he run the same command, he will see:

```
wyj@wyj_sebsd$ sebsd_newrole -r sysadm_r -t sysadm_t
Authenticating wyj.
Password:
Debug: error 9 (authentication error)
Error, incorrect password for wyj
```

The context transition fails.

5.3 Comparison of Rights in Different Roles

Then what’s the different in different roles? The answer is simple, different roles means different rights. We will use an example to illustrate it.

Suppose root has two roles, staff_r and sysadm_r. If root logs in as the staff_r role and enter /etc/security/sebsd/policy, then he runs “ls –Z”, he will see:

```
root@wyj_sebsd# ls -al
total 0
ls: .: Permission denied
```

He has no right to list the files in the directory.
Now he logs in again with sysadm_r, or by executing “*sebsd_newrole –r sysadm_r –t sysadm_t*”, and then runs “ls –Z”, he will see all the files with their security contexts. That’s because the type of /etc/security/sebsd/policy is policy_config_t, and only sysadm_t domain can access it, while the staff_r cannot enter sysadm_t.

---

6 Add a New User Domain

To make it more clearly about the relations between user, role and domain, this

section will describe how to add a new role called second_r and the corresponding domain second_t.

6.1 Editing Relevant Configuration Files

The first file to edit is domains/user.te. Add the following line to the file (either at the top or bottom):

```
full_user_role(second)
```

Note there is a comment like:

```
# if adding new user roles make sure you edit the in_user_role macro in
# macros/user_macros.te to match
```

So we jump to macros/user_macros.te. After finding the place of in_user_role macro, modify it to be:

```
undefine(‘in_user_role’)
define(‘in_user_role’, ‘
role user_r types $1;
role second_r types $1;
’ )
```

Finally, set the default type for second_r in appconfig/default_type by adding the line:

```
second_r:second_t
```

The macro full_user_role(second) declares the second_r, and also creates some related types, such as second_home_t and second_home_dir_t that have been used in previous section. The second_tmp_t is used to label newly created files in /tmp directory by second_t domain. The second_tmmfs_t is used to label shared memory created in tmpfs. And second_tty_device_t and second_devpts_t is used to label tty devices and pseudo tty devices.

6.2 Test the New User Domain

Let us use *setest* to test our new user domain. Firstly, edit the file users and adding the following line to assign *setest* only second_r role.

```
user setest roles { second_r };
```

Then run “make load” to load the new policy. After that remember to label *setest*’s home directory if you did not do it yet:

```
setfmac –R sebsd/setest:object_r:user_home_t /usr/home/setest
setfmac sebsd/setest:object_r:user_home_dir_t /usr/home/setest
```

But it should be noted some new files

Now *setest* can log in the system with second_r role. If he runs “id –M”, he will see his current security context as follows:

```
setest@wyj_sebsd$ id –M
sebsd/setest:second_r:second_t
```

---

7 Explanation of SEBSD Log Message

You must have seen many log messages printed on the console. You can display them again by running “dmesg” command. This section will explain the detailed meaning of the log messages.

Example: The user *wyj* with wyj:user_r:user_t tried to edit /etc/master.passwd by vim. The system got the following log:

```
avc: denied { read } for pid=530 comm=vim inode=1038362, mountpoint=/, scontext=wyj:user_r:user_t tcontext=system_u:object_r:shadow_t tclass=file
```

“avc: denied” means the operation is refused. SEBSD can log both denied event and grant event;
“{ read }” means the process tried to read the file. The contents of the brace { } contains the operation or operations that are requested to be done on the object.;
“for pid＝530” is the PID of the process (vim process here);
“comm=vim” shows the command being executed (vim here);
“inode=1038362” is the inode number of the file to read (/etc/master.passwd here);
“mountpoint=/” shows the mount point of the file system that the target file lies on (root directory of system here);
“scontext=“ shows the subject’s security context;
“tcontext=“ shows the security context of the target object;
“tclass=“ shows the security class of the target object (regular file here).

You can find more logs and analyze their means. These logs help the administrator to improve their configurations and provide proper security policy.

---

8 Resources

The materials about SEBSD are scarce, but there will be more and more available as SEBSD becomes popular.

8.1 Papers and Documents

You can find several papers and technical reports about TrustedBSD MAC framework and SEBSD. And the paper titled “Security-Enhanced BSD” is recommended if you already have some knowledge about SELinux.

8.2 Official Sites

FreeBSD: http://www.freebsd.org
TrustedBSD: http://www.trustedbsd.org
SELinux: http://www.nsa.gov/selinux

8.3 Mail Lists

trustedbsd-discuss@cyrus.watson.org is strongly recommended, where Robert Watson and Scott Long will give detailed answers to your questions about SEBSD.

You can visit http://www.trustedbsd.org/mailinglists.html and select proper mail lists about other components of trustedbsd to subscribe.

Also visit:
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/eresources.html#ERESOURCES-MAIL to subscribe more topics you are interested in.