

Specification of Simplified Policy Description Language(SPDL) ver 2.1.

Yuichi Nakamura *

November 13, 2006

Contents

1 Overview	3
1.1 Feature	3
1.2 Overview of SPDL configuration elements	3
1.3 Default deny rule	4
1.4 Terms	4
2 Structure of configuration by simplified language	4
2.1 Syntax of section	4
3 Import configuration from other file:include	5
4 Declare domain and role	5
4.1 Declare domain:domain	5
4.2 Declare role:role	5
5 Configure RBAC:user	5
6 Domain transition	6
6.1 Domain transition:domain_trans	6
6.2 Simplified domain transition:program	6
7 Access control to normal files:allow/deny	7
7.1 allow	7
7.2 deny	9
7.3 Priority of allow, deny when conflict happens	10
7.4 Special files	11
7.5 Notice about links	11
7.5.1 Treatment of symbolic links	11
7.5.2 Treatment of hard links	12

*himainu-ynakam@miomio.jp

8	Access control to devices:allowdev	13
8.1	allowdev(1)	13
8.2	allowdev(2)	13
9	Access control to files on misc file systems:allowfs	14
10	Access control to temporally file:allowtmp	15
10.1	Why allowtmp is necessary?	15
10.2	What is allowtmp?	16
10.3	Syntax and meaning	16
11	Access control to network:allownet	17
11.1	Port usage	18
11.2	Usage of RAW socket	18
11.3	Usage of Network Interface(netif) and IP address(node)	18
11.4	Inherit socket from other domain	19
12	Access control of process communication:allowcom	20
12.1	allowcom (IPC)	20
12.2	allowcom(Signal)	20
13	Access control other administrative access rights:allowpriv	20
13.1	allowpriv: POSIX capability	21
13.1.1	Capabilities that can not be configured	21
13.1.2	Configurable capabilities	21
13.2	allowpriv: related to kernel	23
13.3	allowpriv: related to SELinux operations	23
13.4	allowpriv: other privileges	24
13.5	denypriv	24
14	Access control of kernel key retention service:allowkey	25

This document describes syntax and meaning of SPDL configuration elements.

1 Overview

Simplified policy is written in syntax called Simplified Policy Description Language (SPDL). SPDL compiler (seedit-converter) converts SPDL into normal SELinux policy language.

1.1 Feature

The feature of SPDL is hiding labels, and reducing number of permissions.

- Hiding labels
SPDL does not use types to configure access control. You can use file name and port number to configure.
- Reduce number of permissions
There are too many permissions in SELinux, so SPDL reduces number of permissions by removing permissions and integrating permissions. Permissions that does not have security impact is removed. Permission removal is implemented by allowing that permission to all domains. Integrating permission means, treating set of permissions as one permission. For example `r` permission for file integrates SELinux permissions related to file read. For detail of what kind of permissions are removed, integrated see *Integrated/unsupported permissions in Simplified Policy* in http://seedit.sourceforge.net/doc/permission_integrate/.

1.2 Overview of SPDL configuration elements

Configuration elements are following.

- Giving domain to applications
To assign domain, we have to configure domain transitions.
SPDL has two elements to configure domain transition: *domain_trans* and *program*. *domain_trans* is syntax to configure domain transition, *program* is syntax for simplified configuration.
- RBAC
SPDL supports RBAC. *role* and *user* elements do that.
- Access control to file
allow/deny are SPDL elements that enables to configure access control to normal files. *allowdev* exists for device files, and *allowtmp* exists for temporally (dynamically created) files.
- Access control to network
allownet does this.

- Access control to IPC
allowcom does access control to IPC and signal.
- Access control to other privilege
Other important OS operations that is not restricted above can be configured by *allowpriv*.

1.3 Default deny rule

Domain and roles are denied all permissions unless allowed by SPDL.

1.4 Terms

- Domain
Domain is the same as domain in SELinux. It is attached to process by domain transition.
- Role
Role in SPDL is simplified. Role is identified with a domain for user shell. In SPDL, we describe access rights for role. In fact, it is giving access rights for user shell of the role. For example, when you give access right for *sysadm_r*, access right is given to *sysadm_t*(Domain for user shell of *sysadm_r*).
Note that in generated SELinux policy, all roles can type every types. There is no syntax corresponding to *role::x::types:y* in simplified language.
- Unconfined domain
unconfined domain is a domain that is configured to be allowed everything. If process is given unconfined domain, it is passes all SELinux permission checks, so it runs the same as in normal Linux.

To configure unconfined domain *allowpriv all;* is used.

2 Structure of configuration by simplified language

Configuration is composed of sections. In each section, access control for domains/roles are described. Section begins with { and ends with }.

2.1 Syntax of section

```
{ (begin of section)
domain/role (declare domain or role, one domain or role can be declared in one
section)
users (This can be used only for role)
domain.trans—program (Configure domain transition)
```

allow/deny—allow:xx (Describe access control for files, access control for resources other than file)
} (end of section)

3 Import configuration from other file:include

you can include configuration from other file by *include* statement. Syntax is following.

include *filename*; The include path can be specified by -I option of seedit-converter, by default it is /etc/seedit/policy/include.

4 Declare domain and role

4.1 Declare domain:domain

1. Syntax
domain *domain name* ;
2. Meaning
Declare domain. All configuration in a section is done for the declared domain.
3. Constraints
Domain name must end with *._t*. This statement can not be used twice in one section.

4.2 Declare role:role

1. Syntax
role *role name* ;
2. Meaning
Declare role. *role name* is associated to user by using *user* statement as shown below.
3. Constraints
role name must end with *._r*. This statement can not be used twice in one section.

5 Configure RBAC:user

1. Syntax
user *user name*;

2. Meaning
Define users who can use the role.
3. Example

```
{
role user_r;
user root;
user ynakam;
....

```

Above means user root and ynakam can use user_r.
4. Constraints
It can be used only section where role is declared.

6 Domain transition

6.1 Domain transition:domain_trans

1. Syntax
domain_trans *parent domain filename-of-entrypoint*;
2. Meaning
This defines how domain is assigned to process.
3. Example

```
{
domain httpd_t;
domain_trans initrc_t /sbin/httpd;
...

```

Above means that when process(domain: initrc_t) executes /sbin/httpd, /sbin/httpd runs as httpd_t domain.

4. Note
Dynamic domain transition can be configured by omitting entry point.
For example, {
domain httpd_t;
domain_trans initrc_t;

means, dynamic domain transition from initrc_t to httpd_t is allowed.

6.2 Simplified domain transition:program

1. Syntax
program *path-to-program*;

2. Meaning

By this, *path-to-program* is attached domain when launched from command line, and */etc/init.d* scripts. That is to say, allow domain transition from unconfined domain. However, domain transition from authentication program domain (such as domains for *su*, *login*, *sshd*) is not configured. Which domains are regarded as authentication_domain is configured in authentication_domain field in *converter.conf*.

3. Example

1)

```
{
domain httpd_t;
program /usr/sbin/httpd;
}
```

/usr/sbin/httpd is attached *httpd_t* domain when launched from command line and */etc/init.d* script.

4. Note

This element is intended to be used in relaxed policy. This will not mean nothing in more strict policy where there is not unconfined domain.

7 Access control to normal files:allow/deny

7.1 allow

1. Syntax

(a) `allow filename | label [r],[w],[x],[s],[o],[t],[a],[c],[e],[dx];`

2. Meaning

(a) Allow access to file.

3. Specifying filename

- Wildcard

For *filename directory/** and *directory*** can be used. For example, */var/** means, all files in */var* directory, and */var* directory. */var/*** means, all files under */var*, */var* directory and files/dirs in its child, child's child... directories.

- Home directories

File name that starts with *~* represents home directory (Not including */root*). For example, */public_html* means, */home/all users/public_html*. But in role configuration, it is different, it means home directories for users that can use role.

4. Meaning of permissions.

- r(Read)
Allows to read file
- w(Write)
Allows to write,create,delete file. Note that creation of device is not allowed unless *allowpriv devcreate* is described.
- x(eXecute)
Allows to execute file.
- s(Search)
Search file tree. i.e. Get contents of directory. For file,means nothing.

5. Example

```
{  
domain httpd_t;  
...  
allow /var/www/** r,s;  
....
```

httpd_t is allowed to read all files and directories under /var/www and its children.

6. Detailed configuration support

In addition to s,r,x,w permissions, permissions o,t,a,c,e can be used. Permission *w* is divided into those permissions.

- o: Overwrite
Allows only writing file, not allow create,delete.
- t: seTattr
Allow modify attribute of file.
- a: Append
Allow append to file.
- c: Create
Allow to create file.
- e: Erase
Allow to delete file

7. Domain execute permission

dx permission means Domain Execute. If domain is defined for the program, program is executed in new domain.

Example:

```
{  
domain httpd_t;  
program /usr/sbin/httpd;
```



```

        allow /var/www/cgi-bin/test.cgi r,s,dx;
    }
    {
domain cgi_t;
    program /var/www/cgi-bin/test.cgi;
    allow .....
    }

```

In this case, httpd_t domain have dx permission to test.cgi. Domain is defined below. So, test.cgi runs as different domain.

8. Limitation about home-directories
Deny statement for individual home directory does not work. For example,

```
deny /home/ynakam/public_html;
```

does not work.

7.2 deny

1. Syntax
deny *filename*;
2. Meaning
This is used to describe constraints for allow and, also used to cancel allow.
3. Example
 - (a) Example 1: Describe constraints

```

*In file constraints
deny /etc/shadow;

*In httpd_t.a
{
domain httpd_t;
include constraints;
allow /etc/* r,s;
}

```

By *include constraints*; configuration in file constrains is included .
So, the above configuration is the same as following.

```

{
domain httpd_t;
include constraints;
deny /etc/shadow;
}

```

```
allow /etc/* r,s;
}
```

This means, httpd_t have r,s permission to files in /etc. But can not access /etc/shadow. To allow access to /etc/shadow, *allow /etc/shadow r,s;* should be described explicitly. Deny is useful to prevent misconfiguration.

(b) Example 2: Cancel allow

```
{
domain httpd_t;
allow /etc/* r,s;
deny /etc;

allow /etc/* r,s; is cancelled by deny /etc;
```

7.3 Priority of allow, deny when conflict happens

1. OR operation(When allow conflicts)
When allow rule conflicts, OR operation is applied.

- Example

```
domain foo_t;
allow /var/** r;
allow /var/** s;
```

foo_t have r,s permission to under /var.

```
domain foo_t;
allow /var/run/* r;
allow /var/run/** w;
```

foo_t have r permission to in /var. But for sub-directory(/var/run/xxx etc), it has w permission.

- Conflict between child and parent

```
domain foo_t;
allow /var/** r;
allow /var/run/** w;
```

foo_t have r permission to under /var(including subdir). For /var/run, it has only w permission.

2. Cancel previous configuration(When allow/deny conflicts)
When allow and deny conflicts, configuration that appears later survives.

- Example

```
domain foo_t;
allow /foo/* r,s;
deny /foo/* ;

allow /foo/* r,s is cancelled.
```

```
domain foo_t;
deny /foo/* ;
allow /foo/* r,s;

deny /foo/* is cancelled.
```

```
domain foo_t;
allow /foo/bar/** r,s;
deny /foo/** ;

allow /foo/bar/** r,s is cancelled.
```

- Exception

However,

```
domain foo_t;
deny /foo/bar/**;
allow /foo/** r,s;
```

*deny /foo/bar/** is not cancelled.* To cancel deny, you have to describe allow for denied directory(in this case, *allow /foo/bar some_permission;*)

7.4 Special files

Access to following files are special.

1. /dev/tty* /dev/pts /dev/ptmx, /dev/vcs*,/dev/vcsa*
If you write allow for those file, this does nothing. Access control to these files must be done by allowdev.
2. /proc, /sysfs, /selinux, /dev/tmpfs
Allow to these files do nothing, because these files are mounted on filesystems that do not support xattr. See allowfs. For /selinux see allowpriv getsecurity.

7.5 Notice about links

7.5.1 Treatment of symbolic links

Configuration to file that contains symbolic link is ignored.

For example,

```
allow /etc/init.d/httpd r;
```

is ignored(init.d is symbolic link to rc.d/init.d).

7.5.2 Treatment of hard links

In Linux system, contents of file can be referred by multiple name using hard link. Hardlink is rarely used recent distro, but you have to note about this if you want to preserve security.

In SPDL, following rule exists about hard link.

If file has multiple hardlink, to access the file, you must specify originally existing file name.

For example, `/etc/shadow` and `/var/chroot/etc/shadow` is hardlinked, and `/etc/shadow` exists originally, to access contents of `/etc/shadow`, you have to use file name `/etc/shadow`. Configuration using `/var/chroot/etc/shadow` will be ignored. If some domain (assume `foo.t`) want to read `/var/chroot/etc/shadow`, you have to configure `allow /etc/shadow r;`

Next, there is a question, what is criteria of file *originally* exist? Following is answer.

In following, `/etc/shadow` and `/var/shadow` is assumed as hardlinked files.

1. If rule is described to one file, the file is treated as original.
Ex: `allow /etc/shadow r;` is described in some domain, but rules using filename `/var/shadow` is not described, `/etc/shadow` is treated as original.
2. If rules are described to multiple hardlinked files, the filename that name is the youngest is treated as original
Ex: `allow /etc/shadow r;` and `allow /var/shadow r;` are described in some domains, `/var/shadow` is treated as original, because `/var/shadow` is younger than `/etc/shadow`.
3. If rules are not described for hardlinked files, the directory names that hardlinks exist are compared. The file whose directory name is oldest is original.
Ex: `/etc/shadow`, `/var/shadow` do not appear in any domain. Then `/var/shadow` is treated as original. Because `/var` is older than `/etc`.

If you are not sure which hardlink is *original*, you can use all names. It means, you can describe

```
allow /etc/shadow r;  
allow /var/shadow r;
```

1 of 2 will be ignored, and do no harm.

Above treatment of hardlink is necessary to avoid a kind of *back door* of path name based configuration. Assume hard link to `/etc/shadow` is created by some trick under `/var/www/html`, without this behavior, apache web server can access contents of `/etc/shadow` via `/var/www/html/shadow`. To protect this, we must limit way to access hard link to 1.

<http://securityblog.org/brindle/2006/04/19> is good reference.

8 Access control to devices:allowdev

8.1 allowdev(1)

Device files must be handled carefully. Because device files are interface to kernel. When device file is linked to driver that handles critical information, read/write to such device will lead to leak of confidential information or break of system. Following allowdev statements restricts access to device files.

1. syntax

- (a) `allowdev -root directory;`

2. meaning

By default, when *allow* statement is described to file, access to device files are not allowed. The directory that contains devices must be described in advance, by `allowdev -root`.

3. Example

```
{
domain httpd_t;
allow /dev/* r,w;
```

In above, `httpd_t` can access normal files under `/dev`, but can not access device files.

```
{
domain httpd_t;
allowdev -root /dev;
allow /dev/* r,w;
```

In above, `httpd_t` can access both normal files and devices under `/dev`. However, in permission *w*, creation and remove devices are not granted unless *allowpriv devcreate* is described.

8.2 allowdev(2)

tty devices are device files `/dev/tty*`, pts devices are devices under `/dev/pts`. tty devices represents local login terminal, and pts devices represents terminal in X and ssh terminal. These devices are linked to terminal when user logs in, or open X/ssh terminal. If you can write other users terminal device files, you can write message to his terminal. In SELinux environment, tty/pts device files are given label according to login user's role. So tty/pts device files should be treated differently in SPDL.

1. syntax

- (a) allowdev -pts---tty---allterm open;
- (b) allowdev -pts---tty---allterm *role* [r],[w];
- (c) allowdev -pts---tty---allterm *role* admin;

2. meaning

-tty means, tty devices. -pts means, pts devices. -allterms means both tty and pts devices.

- (a) This is usually used in role section. Allow role to have its own tty/pts device. At the time of login, by login program, role's tty device file is given type *role prefix.tty_device.t*.
- (b) Allow to read/write *role*'s tty device.
- (c) Allow to change label of tty device, and rename, unlink.

3. Special role

- general
this means tty/pts before labeled(The type is devtty.t and tty_device.t, devpts.t, ptmx.t). Usually, access to these are harmless except *admin* permission.
- all
All other roles tty/pts
- vcs
This can be used only in allowdev -tty. Means vcs file(/dev/vcs*, /dev/vcsa*), these provide access to screen-shot of tty terminal.

9 Access control to files on misc file systems:allowfs

SELinux can do fine-grained access control to files on filesystems that support extended-attributes, such as ext3, ext2 and xfs. For such files, you configure access control using *allow* statement. In other filesystems, you should configure *allowfs* described in this section.

• Syntax

1. allowfs *name_of_filesystem* [s],[r],[x],[w];
For *name_of_filesystem* *tmpfs sysfs autofs usbfs cdfs romfs ramfs dosfs smbfs nfs proc proc_kmsg proc_kcore xattrfs* can be used.

• Meaning

1. Allow access to files in specified system. For example, *allowfs proc s,r;* means to grant s,r access to files on proc filesystem(/proc). When you see logs whose types are *filesystem_t*, you may have to use allowfs. This means, if you find log about *read access to sysfs_t is denied*, you may add *allowfs sysfs s,r;*
- Notice about *name_of_filesystem*
 - proc filesystem
Access control to proc file system is a little fine-grained. *proc_kmsg* means, /proc/kmsg, *proc_kcore* means /proc/kcore. *proc_pid_self* means process information of own process in /proc/pid/. *proc_pid_other* means process information for all others. *proc* means other files on /proc.
 - xattrfs
This means filesystem that supports extended-attribute, but not configured to use SELinux's label. For example, if you format USB memory as ext3 on non-SELinux machine. Next you mount the USB memory in SELinux machine, the files on it are recognized as xattrfs. You have to use *allowfs xattrfs permissions* in such case.
 - cdfs
This corresponds to iso9660 and udf filesystem.
 - dosfs
This corresponds to fat, vfat, ntfs.
 - smbfs
This corresponds to cifs and smbfs.

10 Access control to temporally file:allowtmp

10.1 Why allowtmp is necessary?

allowtmp is prepared to configure access control to temporally files. Before going detail, let's see why such configuration element is necessary. SELinux identifies files based on inode, not file name. File name based configuration does not work correctly when inode number changes or inode does not exist at the time of configuration (typically such files are temporally files). Such files exist under /var/run, /tmp, /var/tmp. For example, assume following configuration exists.

```
domain httpd_t
allow /var/run r,s;
allow /var/run/httpd.pid r,w,s;
```

At first, httpd_t have r,w,s permission to /var/run/httpd.pid. However, when httpd is restarted /var/run/httpd.pid is removed and created again. In this process, inode number is changed. When inode number changes, it inherits parent

directory's permission. i.e: httpd_t have r,s permission to /var/run/httpd.pid(the permission of /var/run). So to grant r,w,s permission to /var/run/httpd.pid, r,w,s permission should be given to parent directory(/var/run). However, in this configuration, httpd_t can r,w,s other daemons pid files under /var/run. In second example, when program creates files randomly under /tmp it is a problem. Assume program A(domain is a_t) and program B(domain is b_t) creates files whose names are random under /tmp. In such case, following configuration will be described.

```
{
domain a_t;
allow /tmp/** r,w;
}
{
domain b_t;
allow /tmp/** r,w;
}
```

This means, program A can access program B's temporally files, and program B can access program A's temporally files.

In above example, access control configuration can not be described for individual files, but for directory what such files belongs. If you think it is enough, following will not necessary :-).

10.2 What is allowtmp?

To resolve this problem, SELinux has a feature called file type transition. *allowtmp* is a feature to configure file type transition. In file type transition, when domain creates files under some directory, created file is given a label. The label can be named by policy. Following is example usage of *allowtmp*.

```
domain httpd_t;
allow /var/run r,s;
allowtmp -dir /var/run -name httpd_var_run_t; -(a)
allow httpd_var_run_t r,w,s; -(b)
```

In (a), when httpd.t create file under /var/run, it is labeled as *httpd_var_run.t*. And in (b), httpd.t can r,w,s access to the created file. To identify file using label name(httpd_var_run.t).

10.3 Syntax and meaning

1. Syntax

- (a) allowtmp -dir *directory* -name *label permission*;
- (b) allowtmp -fs *file system name* -name *label permission*; *permission* is the same as file permission and can be omitted.

2. Meaning

- (a) When domain create file under *directory* it is labeled as *label* and have permission to the created file specified by *permission*. *permission* can be omitted. When omitted, permission can be given by *allow*.
- (b) This is used to configure allowtmp under files that do not support extended attribute, currently, this can be used only for tmpfs.
- (c) About label
 - When *label* is *auto* , label is named automatically based on domain and directory. For example, domain is *hoge.t*, and *directory* is */var/*, label name is *hoge_var.t*.
 - When *label* is *all* or ***, it means all files under *directory* created using allowtmp.

3. Example

```
domain httpd_t ;
allowtmp -dir /var/run -name auto r,w;
```

Files created under */var/run* by *httpd.t* is labeled as *httpd_var_run.t* and *httpd.t* can *r,w,s* access to such files.

```
domain httpd_t
allowtmp -dir /var/run -name auto r,w;
domain named_t
allowtmp -dir /var/run -name auto r,w;
domain initrc_t;
allowtmp -dir /var/run -name all r,w;
```

Files created under */var/run* by *httpd.t* is labeled as *httpd_var_run.t* and *httpd.t* can *r,w* access to such files(*named.t* can not access). Files created under */var/run* by *named.t* is labeled as *named_var_run.t* and *named.t* can *r,w* access to such files(*httpd.t* can not access) *initrc.t* can *r,w* access to above files because *-name all* is specified. *-name all* is used to administrate files created by allowtmp.

11 Access control to network:allownet

allownet statements is prepared to configure network access control. It can configure access control to port , netif(Network Interface), node(IP address) and inheritance of socket.

11.1 Port usage

1. Syntax
allownet -protocol *protocol* -port *port number permission*;
protocol: tcp,udp can be specified, splitted by ,.
port number: number and -1023 and 1024- , and * can be described, splitted by ,.
permission: *client* or *server* splitted by , , can be described
2. Meaning
Allow permissions to be TCP/UDP server/client using port. Port number -1023 means, all unused ports under 1023. 1024- means all unused ports after 1024. * means all ports.
3. Note about udp server
If you describe allownet -protocol udp -port xxx server; The domain also behave as client to port number over 1024.
4. Example

```
domain httpd_t;  
# httpd_t can be server using port 80 and 443.  
allownet -protocol tcp -port 80,443 server;  
# httpd_t can use TCP/UDP 3306 service(MySQL) as client.  
allownet -protocol tcp,udp 3306 client;
```

11.2 Usage of RAW socket

Usage of RAW socket must be restricted, because RAW socket can be used for IP spoofing and eavesdropping.

1. Syntax
allownet -protocol raw use;
permission: *client* or *server* or * splitted by , , can be described.
2. Meaning
The domain is allowed to use RAW socket.

11.3 Usage of Network Interface(netif) and IP address(node)

Usage of netif/node is allowed by this. In default policy, it is allowed to all domains.

1. Syntax
 - (a) allownet -protocol *protocol* -netif *name of NIC permission*;
protocol: tcp,udp,raw and * can be specified, splitted by ,.
name of NIC: NIC name(such as lo,eth0,eth1) splitted by ,.
permission: *send* or *recv* splitted by , , can be described.

- (b) `allownet -protocol protocol -node address permission;`
protocol: tcp,udp,raw and * can be specified, splitted by ,.
address: *ipv4address/netmask* or * splitted by ,. Example: 192.168.0.1/255.255.255.0
. And * means all address.
permission: *send* or *recv* splitted by , can be described.

2. Meaning

- (a) Allows to send or receive packet to/from NIC.
- (b) Allows to send or receive packet to/from IP address.

3. Example

```

{
domain httpd_t;
allownet -protocol tcp use;
allownet -protocol tcp -port 80 server;
allownet -netif eth0 send,recv;
}
--> httpd_t can use tcp socket and be server using TCP 80 port.
And can send/recv packet to/from eth0.

```

11.4 Inherit socket from other domain

Following syntax allows using socket of other domain. It is rare to configure.

1. Syntax

- (a) `allownet -protocol protocol -domain domain use;` *protocol*, tcp,udp can be specified, splitted by ,.

2. Meaning

- (a) This enables to restrict inheriting socket from other domain. This configures from where the domain can inherit socket. When *domain* is *self*, the domain can use socket which is created by its own domain.

3. Example

```

domain foo_t;
# foo_t can inherit UDP socket from bar_t
allownet -protocol udp -domain bar_t;

```

12 Access control of process communication:allowcom

12.1 allowcom (IPC)

1. Syntax
allowcom -ipc|-unix|-sem|-msg|-msgq|-shm|-pipe *to domain* [r],[w];
2. Meaning
Allow to communicate with *to domain* by specified IPC.
If *to domain* is *self*, this means IPC within domain. If *to domain* is * the domain can IPC to every domain. -ipc is allowing all kinds of IPCs, it is simplified configuration. If you want to specify specific kind of ipc, you can use following. -unix is unix domain socket, -sem is semaphore, -msg is message, -msgq is message queue, -shm is shared memory, -pipe is pipe.

12.2 allowcom(Signal)

1. Syntax
allowcom -sig *to domain* [c],[k],[s],[n],[o];
2. Meaning
Allow to send signal to *to domain*. [c] is sigchld, [k] is sigkill, [s] is sigstop, [n] is signull, [o] is other signals.

13 Access control other administrative access rights:allowpriv

- Syntax
allowpriv *string*;
string is name of privilege. It is described in next section.
- Meaning
Allow access rights represented by *string*.

Next, what can be configured for *string* can be categorized into following.

- POSIX capability
- Related to kernel
- Related to SELinux operations
- Others

13.1 allowpriv: POSIX capability

Strings that begin with *cap_* is POSIX capability. You can see detailed meaning by `man capabilities`.

13.1.1 Capabilities that can not be configured

Following POSIX capabilities can not be configured, because it can be configured in different place.

- `CAP_NET_BIND_SERVICE`
This restricts usage of wellknown ports, but by `allownet`, you can configure better. So this is omitted.
- `CAP_MKNOD`
This is allowed in `allowpriv devcreate`.
- `CAP_AUDIT_WRITE`
Operations that is restricted by this is the same as `allowpriv audit_write`,so this is omitted.
- `CAP_AUDIT_CONTROL`
Operations that is restricted by this is the same as `allowpriv audit_control`, so this is omitted.

13.1.2 Configurable capabilities

- `cap_sys_pacct`
Configures kernel accounting(see `acct(2)`).
- `cap_sys_module`
Allows to install kernel module.
- `cap_net_admin`
Allow capability `CAP_NET_ADMIN`(Such as administrate NIC, route table).
- `cap_sys_boot`
Allow capability `CAP_SYS_BOOT`. This means allow the usage of `reboot` system call.
- `cap_sys_rawio`
Allow capability `CAP_SYS_RAWIO`.This means usage of `ioperm`, `iopl` system call and access to `/dev/mem`.
- `cap_sys_chroot`
Allow to use `chroot`.
- `cap_sys_nice`
Allow capability `CAP_SYS_NICE`. This means process scheduling.

- `cap_sys_resource`
Allow capability *CAP_SYS_RESOURCE*. This means usage of `rlimit` etc.
- `cap_sys_time`
Allow capability *CAP_SYS_TIME*. This means modify system clock.
- `cap_sys_admin`
The same as POSIX capability *CAP_SYS_ADMIN*. This permissions overlaps other permissions, so if you allow this, not so serious problem. By denying this, it can restrict `sethostname` and some `ioctl` operations.
- `cap_sys_tty_config`
The same as capability *CAP_TTY_CONFIG*. Change keyboard configuration, and usage of `vhangup` call.
- `cap_ipc_lock`
Allow capability *CAP_IPC_LOCK*. This means to lock memory.
- `cap_dac_override`
- `cap_dac_read_search`
- `cap_setuid`
- `cap_setgid`
- `cap_chown`
- `cap_setpcap`
- `cap_fowner`
- `cap_fsetid`
- `cap_linux_immutable`
- `cap_sys_ptrace`
- `cap_lease`
- `cap_ipc_owner`

- cap_kill

13.2 allowpriv: related to kernel

Configures privileges to communicate and administrate kernel. Following strings can be used.

1. netlink
Allows to communicate with kernel by netlink socket.
2. klog_read
Allows to read kernel messages by syslog(2) call. Usually it is required to use dmesg command.
3. klog_adm
Allows to change configuration of kernel message output.
4. audit_read
Allows to read status and configuration of kernel audit subsystem.
5. audit_write
Allows to send log message to audit subsystem in kernel.
6. audit_adm
Change configuration of kernel audit subsystem.

13.3 allowpriv: related to SELinux operations

Allow privileges to administrate SELinux.

1. relabel
Allow to relabel all files. You must also allow getsecurity and allowpriv search.
2. part_relabel
Allow to relabel files that the domain can write. You must also allow getsecurity.
3. setfscreate
This is necessary only applications that use SELinux API(setfscreatecon).
4. getsecurity
Allow to get security policy decisions, by accessing /selinux.
5. setenforce
Allow to toggle enforcing/permissive mode.
6. load_policy
Allow to load policy to kernel.

7. setseccomp
Change performance parameter of SELinux via /selinux/avc
8. getseccomp
Get security information(such as domain, stored in /proc/pid/attr) of other processes.

13.4 allowpriv: other privileges

Allow other privileges.

1. quotaon
Allow to quotaon.
2. mount
Allow to mount device.
3. unlabeled
Allow full access to unlabeled files(Files labeled as unlabeled_t).
4. devcreate
Allow to create device files in directory that the domain can write. Without this, a process can not create device file on a directory even it is configured writable.
5. setattr
Allow to setattr to files that the domain can s access. Without this setattr permission is granted in w permission.
6. search
Allow s permission to all files.
7. read
Allow r permission to all files.
8. write
Allow w permission to all files.
9. all

13.5 denypriv

This can be used to cancel allowpriv configuration.

14 Access control of kernel key retention service:allowkey

This feature is included at version 2.1 or later.

After Linux 2.6.18, new feature *kernel key retention service* is included. By the feature, each process can obtain key. For detail of key retention service, please refer to kernel document Document/keys.txt (You can look at the copy at <http://free-electrons.com/kerneldoc/latest/keys.txt>). allowkey controls access to key. This feature is effective only for FC5 or later. Cent OS does not have kernel key subsystem, so allowkey means nothing.

1. Syntax

allowkey *domain permissions*;

For permissions, you can use following.

v: View. Look attribute of key.

r: Read. Read contents of key.

W: Write. Write contents of key.

s: Search. Search keyrings.

l: Link. Permits key or keyrings to be linked to.

t: Set Attribute: Set attribute of key.

For detail of permission, see Document/keys.txt.

2. Meaning

Allow access to keys retained by *domain*.

For example,

```
allowkey login_t v,r;
```

means, allow view and read access to keys, obtained by process whose domain is login_t.