Design Of Micro-Kernel Based Security Server Framework with RBAC Policy Server

Won Goo Lee[1], Hee Gyu Lee[2], Kwang Jo Kim[3], Jae Kwang Lee[4]

[1]Dept. of Computer Engineering, University of Hannam, 133 Ojong-Dong, Taeduk-Gu, Taejeon, 306-791, KOREA
jklee@netwk.hannam.ac.kr
[2]Dept. of Computer Engineering, University of Hannam, 133 Ojong-Dong, Taeduk-Gu, Taejeon, 306-791, KOREA
june@netwk.hannam.ac.kr
[3]Dept. of School of Engineering, Information and Commuications Univ., 58-4 Hwaam-Dong, Yuseong-Gu, Daejeon. 305-732, KOREA
kkj@icu.ac.kr
[4]Dept. of Computer Engineering, University of Hannam, 133 Ojong-Dong, Taeduk-Gu, Taejeon, 306-791, KOREA
jklee@netwk.hannam.ac.kr

Abstracts. This paper presents the design and implementation of the secure LINUX micro-kernel using Policy Module based-on RBAC(Role-Based Access Control) mechanism. The LINUX micro-kernel supports system scalability in operating system level which is indispensable to append hardware architecture and gives single system image to roles. Also, this micro-kernel manages system configuration information and hardware resources that which may be efficiently different in each module. The LINUX micro-kernel supports inter-process communication independent to the location of processes and includes RBAC policy module..

1 Introduction

Most of existing system is needed to subsystem to process important and private information, each countries is establishing reliable standards, researching secure OS, and developing trusted systems. After 1980, kernel-level secure OS has been published users and others. Recently, secure kernel approach is design and implementation secure kernel applying micro-kernel mechanism. In comparison to the traditionally used integrated kernel, the micro-kernel based operating system has slower speed. But, micro-kernel based OS and portability point of view. By using RBAC policy providing the more strengthening access control functionality, the very secure OS can be constructed. Thus, this paper presents the design and implementation of the secure LINUX micro-kernel using Policy Module based-on RBAC(Role-Based Access Control) mechanism. The LINUX micro-kernel supports system scalability in operating system level which is indispensable to append hardware architecture and gives single system image to roles. Also, this micro-kernel manages system configuration information and hardware resources which may be

efficiently different in each module. The LINUX micro-kernel supports inter-process communication independent to the location of processes and includes RBAC policy module.

2 Related Works

Nowadays, a distributed environment approach is a necessary activity to share resource and this is more important to strengthen information security. Entities for this access activity were divided into subject and object according to the activation. Generally, subject access to object in according to access method. Here, access control is to control access of subject based status, position and role of subject. this section represent access control model approaching the resource.

2.1 Security Policies : MAC Policy vs. RBAC Policy

Here describe the Mandatory Access Control model commonly known as the Bell-La Padula model. There is a set of classifications, (e.g. top secret, secret, confidential, classified,) which is totally ordered. There is, in addition, a set, of categories that is unordered. The combination of a classification and a subset of the categories is called a security level. Security levels are partially ordered and form a lattice.
Every subject and object in the system must be tabelled by a security level. We will distinguish between trusted and untrusted subjects. Trusted subjects can be relied on not to compromise security; all other subjects are untrusted.
In the case of subjects, the label is called the clearance, and for an object, the label is called the security classification. We will denote the security label by X(s) or X(O).
To ensure secrecy, the following two mandatory rules must be followed:

- Simple Security Property: Subject s can read object, 0 only if X(s) > X(o).
- *-property: Untrusted subject s can write object 0 only if X(s) 5 X(o).

The Simple Security Property is sometimes referred to as the no read up rule, and the *- property is known as the no write down rule.
In the discussion in the next section, we will first assume that the security levels are totally ordered, i.e. that, there exist, n security levels X1,. . , X, such that X1 > X2 > > X,. Subsequently, we will reexamine the results with a security lattice which is not totally ordered.
Differently, Role-based access control(RBAC) is a promising alternative to traditional discretionary access control(DAC) and mandatory access control(MAC). The central idea of RBAC is that permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the roles permissions. RBAC is policy neutral in that the precise policy being enforced is a consequence of how various components of RBAC-such as role hierarchies, constraints and administration of user-role and role-permission assignment-are configured. Originally, The concept of RBAC began with multi-user and multi-application on-line systems pioneered in the 1970s. The central notion of RBAC is that permissions are associated with roles,

and users are assigned to appropriate roles. This greatly simplifies management of permissions. Roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. Although the basic concept of RBAC has been around for some time, it is only recently that security researchers have studied it rigorously. There are different aspects to RBAC that are dealt with in different ways in different systems. This makes it difficult to arrive at a consensus definition of RBAC.
A major attribute of RBAC is that it is policy neutral. RBAC provides support for several important security principles (notably least privilege, privilege abstraction and separation of duties), but does not dictate how, or even if, these should be put into practice. The precise policy enforced in RBAC is a consequence of the detailed configuration of various components of RBAC, such as role hierarchies, constraints and administration of user-role and role-permission assignment.

2.2 Applied Policy Model: RBAC Model vs. ACLs Model

The section compares simple Role Based Access Control (RBAC) models and access control lists (ACL). RBAC has several advantages over ACLs. Even a very simple RBAC model affords an administrator the opportunity to express an access control policy in terms of the way that the organization is viewed, i.e., in terms of the roles that individuals play within the organization. With RBAC, it is not necessary to translate a
natural organizational view into another view in order to accommodate an access control mechanism.
ACLs(Access Control Lists) typically associate an object with a list of users and groups. Associated with each user or group in an ACLs for an object is a set of operations which may be performed on that object. An operation on the object may be perfomed by a user if that user or a group to which that user belongs is listed in the ACLs associated with the object and that operation is associated with that user or that group. PASC P1003.le(formerly know as POSIX.6) and Windows NT are examples of specifications which define ACLs mechanisms.
Consider an ACLs mechanism, ACL, where only groups are permitted as entries in the ACLs. ACL may have an arbitrary number of groups and there are no restrictions on a users membership in any group or several groups. To describe an access control policy using ACL, an administrator:

1. Creates groups of individuals according to their responsibilities (i.e., every member of a group has the same responsibilities).
2. Associates with these groups the permissions necessary for the individuals in the group to carry out their responsibilities.

To describe an access control policy using RBAC, an administrator:
1. Creates roles based on the responsibilities necessary to meet the goals of the organization.

2. Associates these roles with the permissions necessary to carry out these roles.
3. Associates these roles to individuals.

ACL is equivalent to RBAC from the point of view that any access control policy described using ACLG can be described by RBAC and any access control policy described by RBAC can be described by ACL. This equivalence can be shown by creating a one-to-one correspondence between the groups in the access control policy described using ACL and the roles in the access control policy described using RBAC.
Given an access control policy described using ACL, the equivalent access control policy can be constructed using RBAC by associating a role with the user if that user is a member of the group which maps to that role. Conversely, given an access control policy described using RBAC, the equivalent access control policy can be constructed using ACL by making the user a member of the group which maps to the role if that user is associated with that role. Appendix A provides a more precise description of how these constructions can be accomplished. Note that:

- The functions in Appendix A used to define an access control policy based on RBAC or ACL express the capability of RBAC and ACL as a means to represent access control policy. Moreover, these functions mirror the actions taken by an administrator to create the access control policy using RBAC or ACL.

The constructions used to show the equivalence only depend on user/role, role/permission, user/group, and group/permission associations in the access control policy representations. How the permissions are represented is immaterial to the constructions as long as the permission representations are the same in both RBAC and ACLs.

Windows NT is an example of a network operating system which supports a group ACL mechanism that includes the functionality of ACL. It is possible to implement RBACM in such an environment by simply creating tools to administer the RBAC metaphor using the ACL mechanism provided. To accomplish this, the groups of ACL become the roles of RBAC. Such tools can usually be implemented as privileged applications that:

- only require processing during the Administration phase (see section 1.2).
- only require no change to existing privileged system or kernel processes which provide processing during the Session and Enforcement phases.

Not only is it usually possible to implement RBAC in such a manner given an ACLs mechanism which supports ACL, it is also usually possible to implement the role hierarchy, Static Separation of Duty, and Cardinality features of the NIST Mode1

3. Role-Based Framework for Proposed Security Policy Server

3.1 Proposed Security Server Architecture

Secure Linux system consists of micro-kernel, Linux server, and security policy server as shown Fig. 1. This system supports the strengthen policy and mechanism, also embed reference monitor or policy server. Otherwise secure kernel approach is design and implementation secure kernel applying micro-kernel mechanism. In comparison to the traditionally used integrated kernel, the micro-kernel based operating system has slower speed. But, micro-kernel based OS and portability point of view. Each unit and memory of a processor must be initialized by using the boot information so that the multi-level OS can actively run the function of the system..

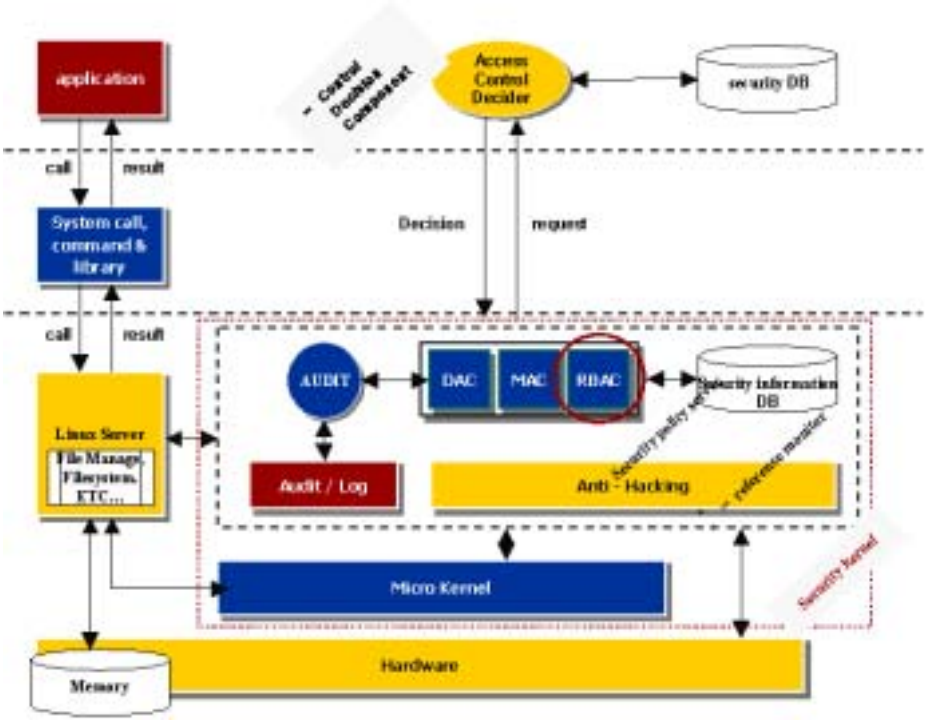

Fig. 1 The entire architecture of proposed security policy server

3.2 Proposed Access Model

Fig. 2 shows the available path of information flow to secure system applied the proposed BLP mechanism.. that is, this mechanism can be define security

requirements to protect illegal information effluence out system dealing data with different security level. Such basic features are the followings.

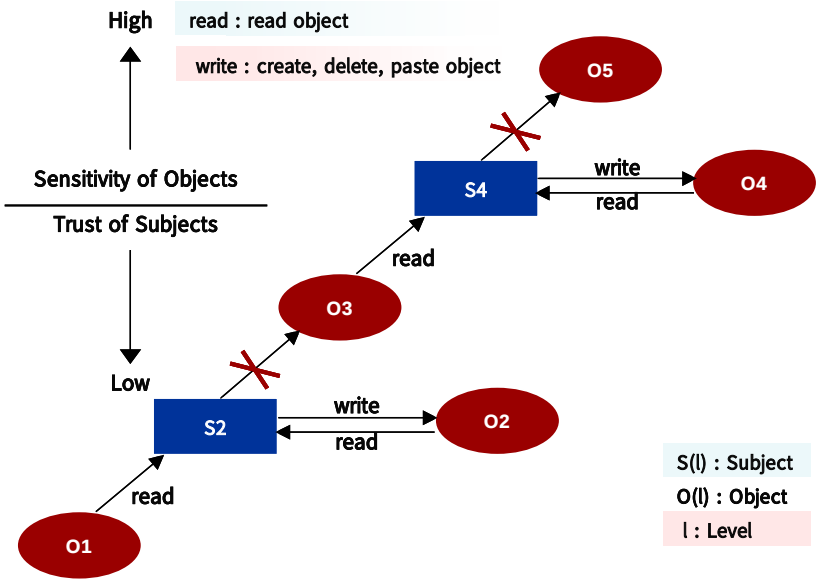

Fig. 2 Proposed BLP model

① if only C(S) >= C(O), subject S(i) must be read object O(i).
② if only C(S) <= C(O), subject S must be write object O

On orginal BLP model, Although C(S) <= C(O), subject S may be write object O. But, as above Fig. 3, Write operation is universal operation including Create operation, Append operation, and Delete operation. However, it isn't capable that low-level subject can be append and delete higher-level object and To use Write operation is limited, because security problems is occurred

3.3 Proposed RBAC Policy Model

RBAC policy system is security system designed by RBAC mechanism based PC embedded Linux operating system in Intranet. Also, can be widely perform management of data admitted permission, with management capability. Otherwise, the system can be consistently have a relative information with management tool in itself. Thus, this needs a various sets and function. We design RBAC module applying on secure OS on the basis of RBAC96 model. On components of secure OS, the most components of them is management tool, the tool store user-role, role-role relationship into Database and manage them such as below Fig. 3. thus information in database for each others must be consistently maintained

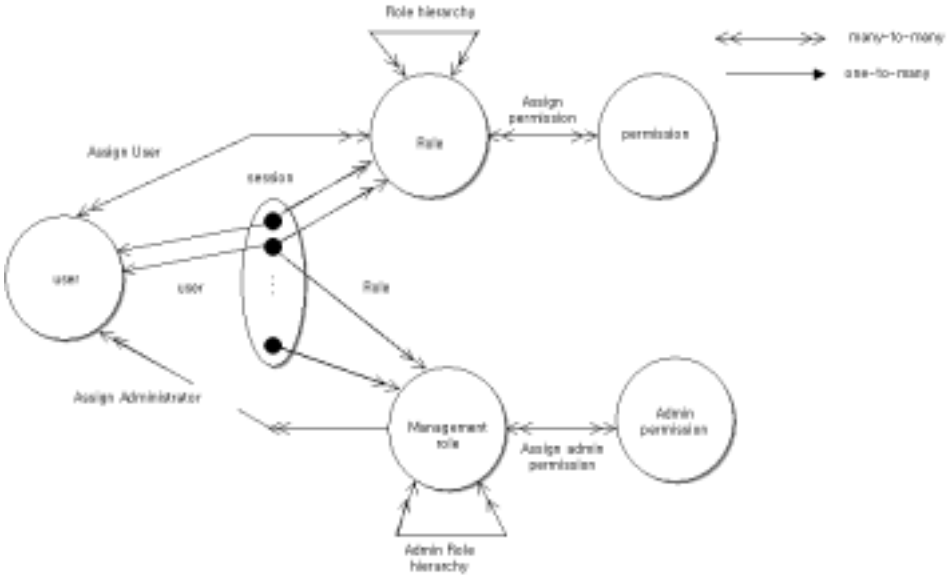

Fig 3. Proposed RBAC policy model

4. A Framework for Proposed Secure Micro-Kernel with RBAC Module

Micro-kernel based approach gives more benefits compared with integrated kernel based approach. The LINUX micro-kernel is designed for effective boot mechanism, system configuration information handling, resource management, dynamic kernel virtual address allocation and hardware specific functions like as interrupts. It also supports a consistent IPC mechanism transparent with the process location.

As Fig. 4, proposed security server perform security policy in harmony of three security policy : MLS(Multi-level Security), Type Enforcement, Identity-based Access Control, Dynamic Role-based Access Control.

Access decision provided by security server is different from the existing security server in three aspects; constructing MLS(Multi-Level System), supporting DRBAC(Dynamic Role-Based Access Control), and supporting Identity-based Access Control.

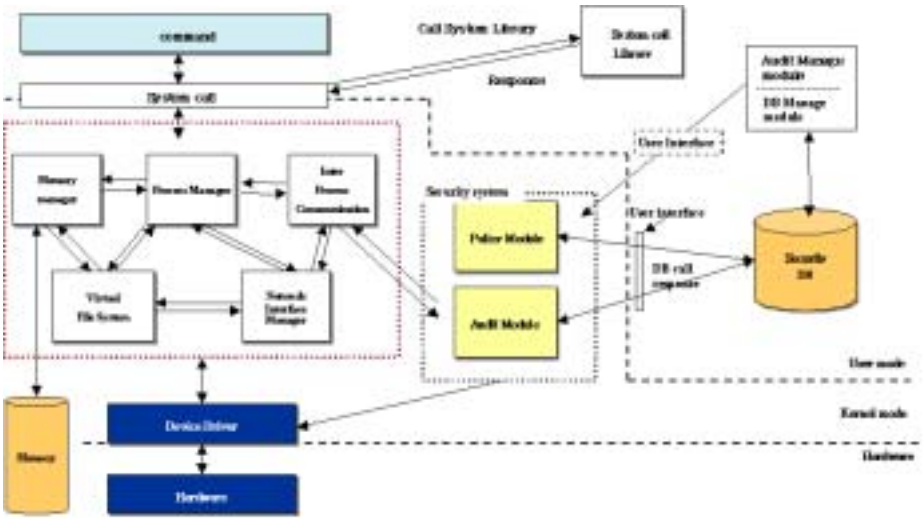

Fig. 4 The structure of Security Sever

4.1 A Framework for Proposed Secure OS

Fig. 5 shows secure operation system framework. Linux kernel consists of five module: task manager, file system, network manager, device driver, and memory manager. Kernel is resource manager, manage resources: physical resource and abstract resource. The major three level is divided into user level, kernel level, hardware level. User level consists of common command(e.g, ls, ps, mount, etc.) and compiled results. Hardware level consists of physical resources(CPU, memory, and disk, etc.,). Kernel level is between user level and hardware level, manage hardware, provide user level services. Kernel communicate hardware by using device interface and interrupt processing mechanism, also communicate user level application by using system call interface.

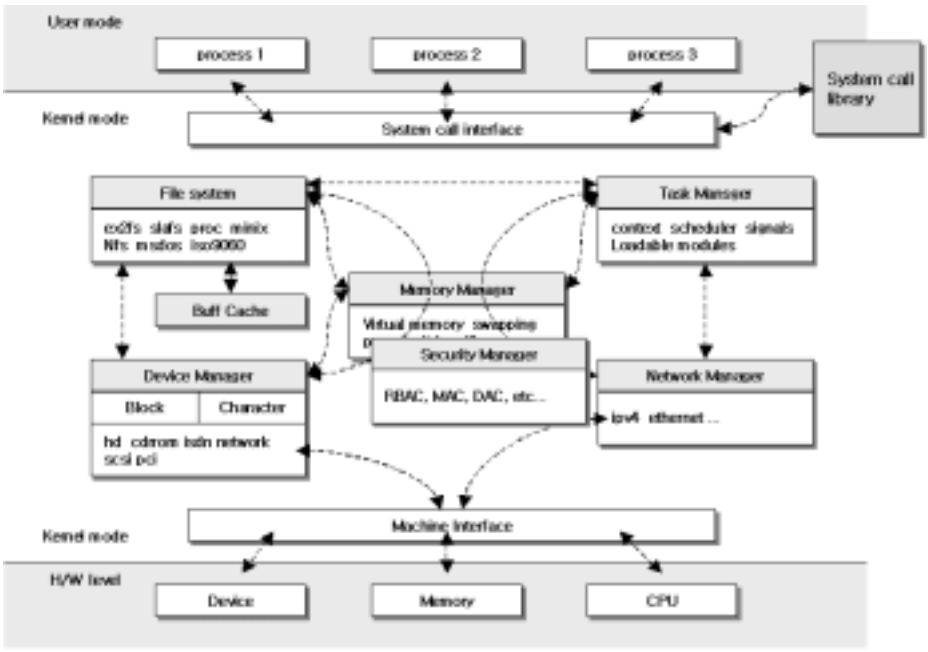

Fig. 5 Secure OS Framework

4.2 A Framework for each module on Micro-Kernel

The structure of each File and Directory represent at below Fig. 6. all file system represent a rbac_device_list_item_t structure, they were associated at doubly linked list by device_list_head. Here, device_list_head is pointer, curr is temporary pointer and indicate the latest item. Count indicate the number of item consisting of this lists. On rbac_device_list_item_t stucture, id is device number existing on file system, head_table is a array of rbac_file_dir_list_head_t structure and the head of rbac_file_dir_list. Each item include ACI for a file and a directory, the structure of ACI is rbac_file_dir_aci_t. Each file or directory are distinguished by the number of inode, are stored into 30 lists by using hash function to faster search inode needed.

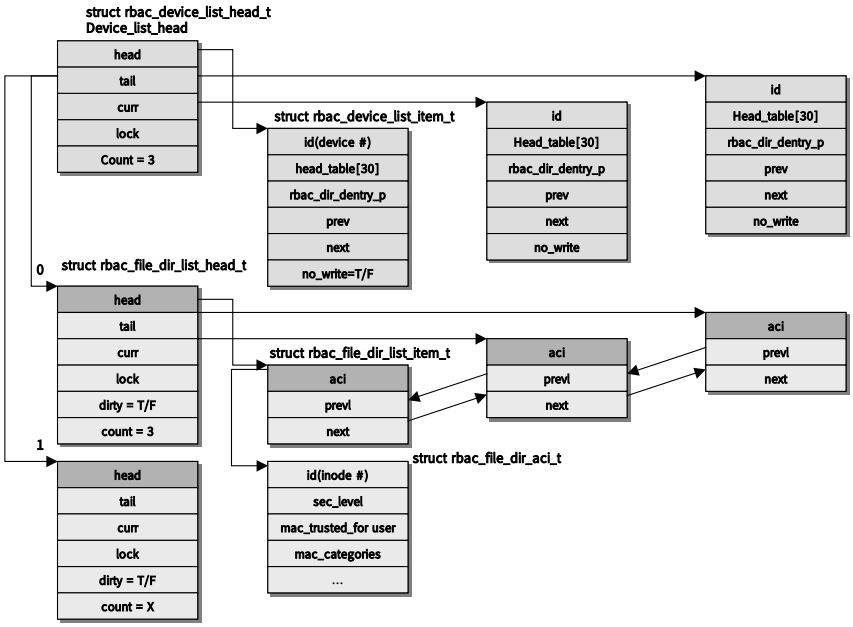

Fig. 6 The structure of File & Directory on Micro-Kernel

ACI on processes are stored into process_list_head, doubly linked list. Each items are distinguished into the number of process and include ACI on a process.. The structure of Device Special File and IPC is similarity of The structure of File.

Signal processing flow represents at Fig. 7. Constraint Condition for task to process signal is that task can be transfer si\gnal into other tasks, receive the signal on coming it, and call function processing signal.

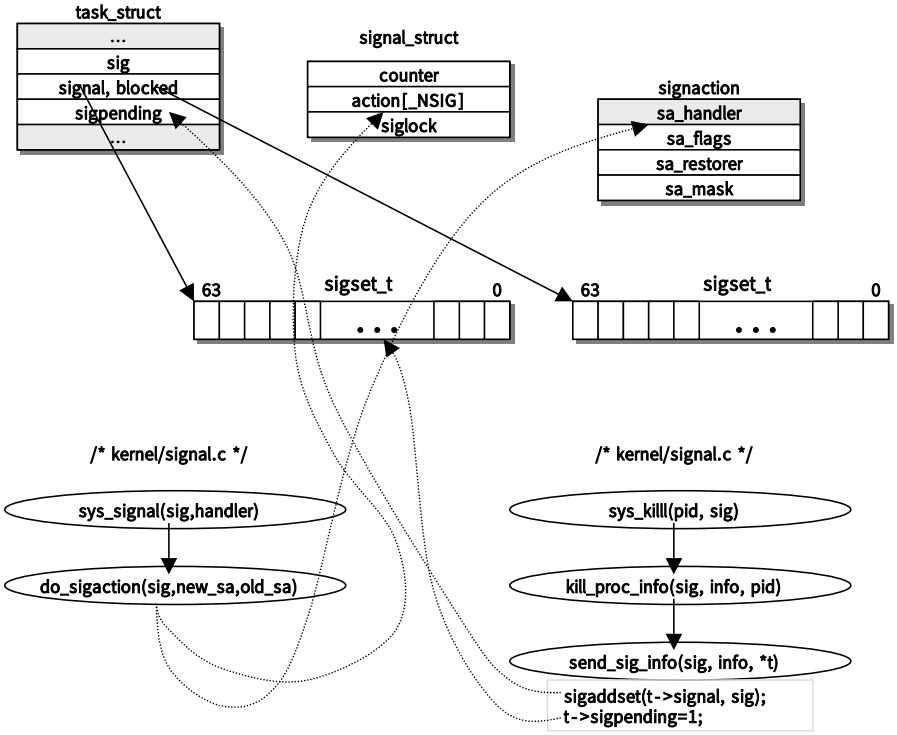

Fig. 7 Variables related Signal Processing on Micro-Kernel

5. A Simple Embedding for Proposed RBAC Module

5.1 Kernel Initialization

Linux kernel start at start_kernel() function(init/main.c), is initialized by processing kernel option and needed daemon with kernel_thread() by the start_kernel() function. rsbac_parse_koptions() function(rsbac/help/debug.c) process kernel option and runtime option related RBAC, their source codes represent at below

asmlinkage void __init start_kernel(void)

...
if (**rbac_parse_koptions(command_line)**)
continue;
...

...
boolean __init rbac_parse_koptions(char * line)

...
int init(void * unused)

lock_kernel();
do_basic_setup();
rbac_init();
...
execve(/sbin/init);

Here, init() function(init/main.c) initialize parts of kernel and is operated by kernel_thread, run init process, and perfume rbac initialization

5.2 RBAC Initialzation

RBAC initialization is accomplished by rbac_init() function(rbac/help/debug.c). the function initializes each lists, constructs by reading ACI stored into file, initializes each policy module, and generates 'rbacd' kernel thread. The function codes consisting of lists codes represent at below

struct rbac_device_list_item_t * add_device_item(kdev_t kdev)

new_item_p = kmalloc(struct **rbac_device_list_item_t,** GFP_KERNEL);
...
new_item_p->rbac_dir_dentry_p = NULL;
for (i = 0; i < 30; i++)
...

int read_fd_lists(struct **rbac_device_list_item_t** * device_p, kdev_t kdev)

...

struct rbac_process_list_item_t* add_process_item(struct rbac_process_aci_t aci)

...._register(/proc/rbac-info, versions);

void __init rsbac_init_debug(void)

if rbac was not initialized yet
read and set rbac_debug_adf[] from file /rbac/dbugadf;
proc_register(/proc/rbac-info, log_levels);

proc_register(/proc/rbac-info, debug);
proc_register(/proc/rbac-info, rmsg);
proc_register(/proc/rbac-info/backup, dbugadf);

for each **rbac_debug_***
if set, log a message with printk();

5.3 ACI Function

On RBAC module, three function : rsbac_get_attr(), rsbac_set_attr(), rsbac_remove_target() [rsbac/data_structures/aci_data_structures.c] search and revise ACI. As call the function, a common data structure represent at below.

enum **rsbac_target_t**
T_FILE, T_DIR, T_DEV, T_IPC, T_SCD,
T_USER, T_PROCESS, T_FD, T_NONE
;
union **rsbac_target_id_t**
...
;
enum **rsbac_attribute_t**
A_owner, A_pseudo, A_security_level,
A_mac_categories, A_object_category,
A_data_type, A_system_role,
A_current_sec_level, A_mac_curr_categories, ...
;
union rsbac_attribute_value_t

5.3.1 rbac_get_attr()

rbac_get_attr() load the security attribute of object and if the attribute is not, use default value or inherit the attribute of upper object in accordance with option on compile. The function codes is shown simply at below

int **rbac_get_attr**(enum rsbac_target_t target, union rbac_target_id_t tid, enum rbac_attribute_t attr,
union rbac_attribute_value_t * value, boolean inherit)

switch (target)
case T_FILE: case T_DIR:
hash_nr = fd_hash(tid.file.inode); /* inode % 30 */
device_p = lookup_device(tid.file.device);
fd_item_p = lookup_file_dir(tid.file.inode, device_p);
switch (attr)
case A_security_level:
if (fd_item_p != NULL) value->security_level = fd_item_p->aci.sec_level;
else value->security_level = default_fd_aci.sec_level;

```
if (value->security_level == SL_inherit && inherit)
 break;
case A_XXX:

case T_YYY:
default :
return(-RSBAC_EINVALIDTARGET);

return(err);
```

5.3.2 rbac_set_attr()

rbac_set_attr() modify the security attribute of object and if the attribute is not, generate a given attribute value. The function codes is shown simply at below

int **rbac_set_attr**(enum rsbac_target_t target, union rsbac_target_id_t tid,
enum rbac_attribute_t attr, union rsbac_attribute_value_t * value)

```
switch (target)
case T_FILE: case T_DIR:
hash_nr = fd_hash(tid.file.inode); /* inode % 30 */
device_p = lookup_device(tid.file.device);
fd_item_p = lookup_file_dir(tid.file.inode, device_p);
if (fd_item_p == NULL)
fd_aci.id = tid.file.inode;
fd_item_p = add_file_dir_item(fd_aci, device_p);
if (fd_tiem_p == NULL) return (-RBAC_ECOULDNOTADDITEM);

switch (attr)
case A_security_level:
fd_item_p->aci.secl_level = value->security_level;
break;
case A_XXX:

case T_YYY:
default :
return(-RBAC_EINVALIDTARGET);

return(err);
```

5.3.3 rbac_remove_target()

As delete object, rbac_remove_target() delete its security attribute. the following codes is simply summarized.

int **rbac_remove_target(**enum rbac_target_t target, union rbac_target_id_t tid)

```
switch (target)
```

```
case T_FILE: case T_DIR:
if (target == T_FILE) rbac_auth_remove_f_capset(tid.file);
rbac_acl_remove_acl(target, tid);
hash_nr = fd_hash(tid.file.inode);
device_p = lookup_device(tid.file.device);
fd_item_p = lookup_file_dir(tid.file.inode, device_p);
remove_file_dir_item(tid.file.inode, device_p);
device_p->head_table[head_nr].dirty = TRUE;
break;
case T_YYY:
default :
return(-RBAC_EINVALIDTARGET);
return(err);
```

6. Conclusion and Futher Works

Most of existing system is needed to subsystem to process important and private information, each countries is establishing reliable standards, researching secure OS, and developing trusted systems. After 1980, kernel-level secure OS has been published users and others. Recently, secure kernel approach is design and implementation secure kernel applying micro-kernel mechanism. In comparison to the traditionally used integrated kernel, the micro-kernel based operating system has slower speed. But, micro-kernel based OS and portability point of view. Each unit and memory of a processor must be initialized by using a booting information so that the multi-level OS can actively run the function of the system. By using RBAC policy providing the more strengthening access control functionality, the very secure OS can be constructed.

References

1. Tae Gyu Park, Yon Ho Im, “*Implementation secure OS based linux kernel*”, *Proc. of KOSTI 2000*, 2000
2. Dok Jong Son, “*Object-oriented RBAC model using desing pattern*”, M.S paper, Ajou Univ. Feb, 2000.
3. J. G. Ko et al. "*Design and Implementing for Secure OS based on Linux*", *Proc. of WISA2000*, pp.175-182, Nov. 2000.
4. Dae Jung Kim, Hyung Jung Kim,,,, “*Implementation RBAC system based MLS*”, *Proc. of CISC 2001*, pp.39-42, Dec. 2001.
5. Jae Kyoung Park, "Features implementation of BLP Security model on Linux”, M.S paper, Hongik Univ. Feb, 2000
6. Ulfar Erlingsson ,Athanasios Kyparlis, "Microkernels," http://os.korea.ac.kr/~yuko/research/microkernel/cornel.htm.
8. The University of Western Ontario and RAVI SANDHU and QAMAR MUNAWER George Mason University, “*Configuring Role-Based Access Control to Enforce Mandatory and*

Discretionary Access Control Policies", *ACM Transactions on Information and System Security*, Vol. 3, No. 2, pp.85-90, May. 2000.