# An Architecture for Systematic Administration of SELinux Policies in Distributed Environments

Pedro Chavez Lugo, Juan J. Flores, and Juan Manuel Garcia Garcia

*Abstract*—**An operating system designed under the criteria of the class A1, consists of a collection of security strengthening mechanisms for the kernel. SELinux is an example of this type of operating system that supports several types of security policies applied to access control. In this paper we address the problem of inconsistency in SELinux policies, which can be present in distributed environments. To solve this problem, we propose an architecture that integrates a policy server for enabling a simple and secure administration. The policy server collects, integrates, and updates all policies that are applied in the distributed environment. We aim to achieve authenticity, integrity and confidentiality in the policy update process through the Kerberos V protocol. A redundant policy server is used to obtain availability on policies.**

*Index Terms*—**Access, control, distributed, administration, SELinux, policies, Kerberos.**

## I. Introduction

NOwadays an operating system must integrate all the security mechanisms that enable it to identify users, control access to system resources, and record events (legitimate and intrusive). An operating system should provide functionality for managing hardware, serve as a base for application programs, and act as an intermediary between the end user and hardware, in addition to providing security and protection [1]. The Orange Book [2] classifies systems into D, C, B, and A divisions of enhanced security protection. Division A uses of formal security verification methods to assure that the mandatory and discretionary security controls; class A1 is a verified design [3]. A class A1 system bring along an increase in complexity in the use and administration. In this paper, we address the problem of inconsistency in SELinux policies, which can be present in distributed environments.

### *A. Security Mechanisms*

To identify users, limit the access to objects, and log the actions performed by subjects, an operating system must contains some of the following non bypassable mechanisms:

- Authentication.
- Auditing.
- Access Control.

Those mechanisms are described in the following subsections. The first question is how to determine if a computer system is secure. To answer this question some organizations formed hackers teams trying to obtain unauthorized access to system resources. But the best point to the attackers is to know vulnerabilities that are not known by developers and administrators.

*1) Authentication Mechanism:* The authentication mechanism determines if the user really is who he/she claims to be. Authentication can be based on one or more of the following factors:

- Something you know (a number or password).
- Something you possess (a key or smart card).
- Something you are (Biometrics).

A “something you know” factor does limit the number of incorrect online or offline login attemps. Users can reproduce their own features accurately and repeatedly by a biometric factor. The “something you possess” factor is a poor authentication mechanism, and it is neccessary to employ the “something you know” or “something you are” factor. A strong authentication mechanism combines two or more factors but it comes with an increase in cost. Some works about biometry, password managers, and smartcards can be found in [4]–[7].

*2) Auditing Mechanism:* The designers, builders, and administrators sometimes need to analyze the audit records to solve security problems. A trusted OS needs the ability to record (log) the system’s and the users’ activity. The identity, action, and time are the minimum aspects to log in order to answer questions. It’s necessary to log the activities to:

- Perform chronological reconstructions of events.
- Detect unauthorized events recognition/spoofing.
- Provide problem identification.

It is necessary to limit the space for the audit records in a storage medium. More detailed information about audit records can be found in [8].

*3) Access Control Mechanism:* An operating system must contain a lot of subjects and objects and each subject can access some or all objects (see Fig. 1). Access control limits the interaction between subjects and objects. Authorization is part of access control, and its function is to grant or deny the access to an object by a subject action (see Fig. 2).

Morrie Gasser [9] cites three tasks for the access control mechanism:

- *Authorization* determines which subjects are entitled to have access to which objects.
- *Determining the access rights* (a combination of access modes such as read, write, execute, delete, etc).
- *Enforcing the access rights.*

In Access Control subjects and objects have security attributes, and access is determined by a policy. A policy is a set of rules that guide the access control engine based on

Div. de Est. de Postgrado, F. de Ing. Electrica, Universidad Michoacana, Morelia, Michoacan, Email: juanf@umich.mx, pedro@lsc.fie.umich.mx
Dep. de Sistemas Computacionales, Instituto Tecnologico de Morelia, Morelia, Michoacan, Email: jmgarcia@itmorelia.edu.mx

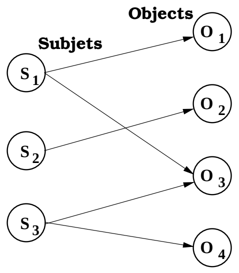

Fig. 1. Subjects accessing objects

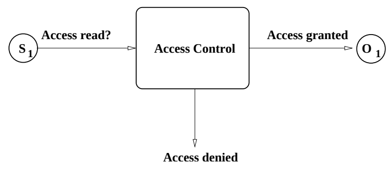

Fig. 2. Access control - Authorization

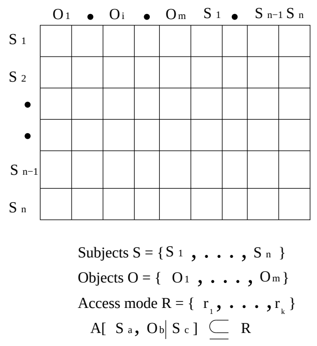
O1 • Oi • Om S1 • Sn−1 Sn
S1
S2
•
•
Sn−1
Sn
Subjects S = {S1 , . . . , Sn }
Objects O = { O1 , . . . , Om }
Access mode R = { r1 , . . . , rk }
A[ Sa , Ob | Sc ] ⊆ R

Fig. 3. Access matrix

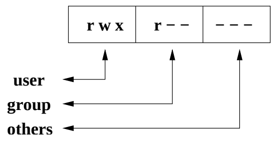

Fig. 4. Object mode permission bits

one or more access control models. A *constraint* is a mean to disallow granted permissions. Some operating systems like SELinux use some access control models, and the system administrators need to know the right configuration steps and each model used [10]. In following sections we define the two diferent access control types and the most popular access control models.

## II. Access Control Types

An access control type describes the conceptual definition for access control. The Trusted Computer System Evaluation Criteria document [11], cites two different access control types:

- *Discretionary Access Control (DAC).*
- *Mandatory or Non-discretionary Access Control (MAC).*

### A. Discretionary Access Control

DAC is a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. This kind of control is discretionary in the sense that a subject with a certain access permission is capable of passing on that permission (perhaps indirectly) to any other subject (unless restrained by mandatory access control).

The rules of Discretionary Access Control allow users to change the security attributes of their objects.

### B. Mandatory Access Control

MAC is a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity.

The rules in MAC disallow users to change security attributes to their objects. We believe it is necessary a new MAC definition, including a policy that specifies that rules are controlled by the organization and not by users.

## III. Access Control Models

An access control model describes the ideal and concrete definition for access control. A formal model is an important component that provides a base to design and build trusted systems. It is necessary to enforce the chosen model in the kernel and not to use the user space to solve all security problems. The formal model helps to demonstrate how secure a given implementation is. The survey [12] cites some formal models (developed from 1970 to 1980) to prove an O.S. really provides the security its claims.

### A. Access Matrix Model

A first matrix access control approach was proposed by Lampson B. W. [13]; his model has four main sets: Subjects (S), Objects (O), read-write-execute combination represented by access mode (R), and a matrix to represent how and which objects or subjects can be accessed by a subject.

Figure 3 shows an access matrix and notations for subjects, objects, and access modes. An object $O_b$ or subject $S_c$ can be accessed in mode $r_j$ by subject $S_a$ if A[$S_a$,$O_b$ | $S_c$].

Nowadays, a variant of the matrix model variant is called Access Control Lists (ACLs), and used in the Unix and Linux OS since the 70's and 80's, respectively [14]. In this model every object has three sets of permissions to define access for the owner, owning group, and others. Each set defines read (r), write (w), and execute (x) permissions, wich are represented by only nine bits (see Fig. 4).

ACLs, has survived because it is a simple model, and provides easy customization, administration, and usage. A disadvantage of ACLs is the coarse access granularity; i.e. user, group, and others. ACLs is a DAC model, where the users/subjects can change their objects security attributes. For some subject operations it is necessary the admin identit. This final point can generate a threat like buffer overflow to gain permission administrator privileges [15].

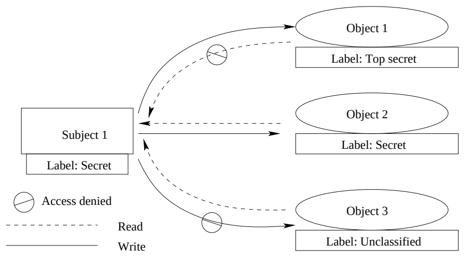

Fig. 5. Data flow in MLS

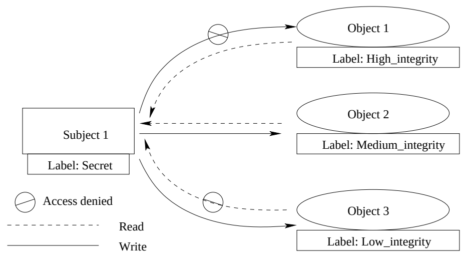

Fig. 6. Data flow in Biba Integrity

## B. Multilevel Security

D. Elliott Bell y Leonard J. Lapadula proposed the Multi-level Security (MLS) model [16], based on a military information structure. Subjects are classified in clearance levels and objects in sensibility levels. Top secret, Secret, and Unclassified are name labels for sensibility levels. The need to know principle is used to limit subject access needed for a specific work and close the unnecessary access to other objects with the same sensibility label. Two properties define the interaction between subjects and objects, where L defines a clearance level or a sensibility level:

- *Simple security property*: A subject *s* can read object *o* if and only if L(o) $\leq$ L(s) and has permission to read *o*.
- *Star property*: A subject *s* can write object *o* if and only if L(s) $\leq$ L(o) and has permission to write.

Fig. 5 shows the data flow in MLS. A Subject with “Secret” label has write access to objects with “Top Secret” label, and only read access to objects with same or less label. A MLS model is characterized by the phrase “no write down, no read up”. The MLS is a Mandatory Access Control type where the users can change security attributes of their objects only if the policy grants it. The main MLS goal was confidentiality enforcement to prevent unauthorized disclosure of sensitive information.

## C. Biba Integrity Model

Clark and Wilson [17] compare military and commercial systems and conclude that control of confidential information is important in both environments, but a goal of commercial systems is the separation of duties to ensure information integrity to prevent frauds and errors. The Biba integrity model [18] is similar to the MLS model, but its goal was integrity enforcement to prevent unauthorized modification of sensitive information. Fig. 6 shows the data flow in the Biba Integrity model. A Subject with “Medium_integrity” label has read access to objects with “High_integrity” label, and only write access to objects with same or less label. The Biba integrity model is characterized by the phrase “no write up, no read down”. Chinese Wall is an other MAC example proposed to be used in commercial environments [19] for the integrity enforcement.

## D. Type Enforcement

Type Enforcement (TE) Technology is a Secure Computing trademark; in this scheme a process is confined to access only

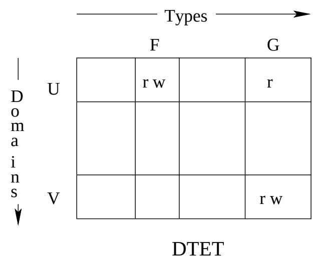

Fig. 7. DTE table

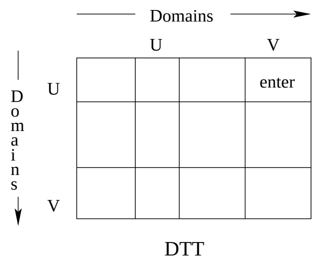

Fig. 8. DT table

to the necessary resources to do its work (least privilege). Domain and Type Enforcement [20], is an improved TE, where subjects are classified in domains and objects in types. A process can into other domain by a transition. The relations between subjects and objects are controlled by a Domain and Type Enforcement Table (DTET), and a Domain Transition Table (DTT). DTET specifies what types and access modes can be accessed by a domain (see Fig. 7). DTT specifies the transitions between domains,

(see Fig. 8). In 1995 DTE was proposed to be implemented in the UNIX OS [21], and was customized in 1996 [22] to confine some processes like httpd that require the admin identity. In Linux, DTE was customized in 1997 by Serge E. Hallyn [23].

## E. Role Based Access Control

In Role Based Access Control (RBAC), a role defines tasks and responsibilities of organization members [24]. The roles doctor, nurse, and administrator, are role examples in a hospital. Recently, the National Institute of Standards and Technology proposed a standard for RBAC [25], based on the RBAC reference model and the RBAC functional specification. The RBAC reference model provides a rigorous definition of

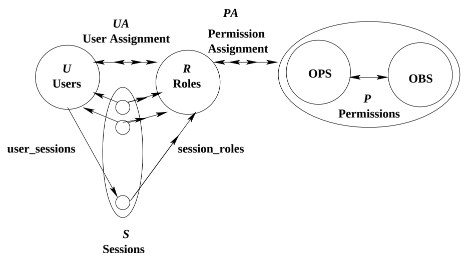

Fig. 9. Core RBAC

RBAC sets and relations and the RBAC functional specification defines requirements over administrative operations for the creation and maintenance of RBAC sets and relations.

The RBAC reference model defines a set of Users-U, Roles-R, Sessions-S, and Permissions-P (see Fig. 9). A user is a human, a role is an occupation within an organization semantics. A permission is an access mode for an object or objects. The users can get one or more roles per session.

## IV. SELINUX

An operating system designed under the criteria of class A1, consists of a collection of security strengthening mechanisms for the kernel. SELinux is a Linux kernel module, designed by the National Security Agency (NSA), that produces an operating system class A1. SELinux supports several access control models.

SELinux kernel module is based on the Flux kernel advanced security (Flask), which is a flexible architecture that supports several mandatory policies. The Flask architecture was designed for microkernel environments but their use has spread to the Linux monolithic kernel to produce SELinux [26]. SELinux enhances the access control mechanism through integration of the *Discretionary Access Control* (DAC), and the *Mandatory Access Control* (MAC). The Access Control List (ACL) model used by the discretionary access control, and the Multilevel Security, Type Enforcement (TE), Role-Based Access Control (RBAC), and User Identity (UI) models are used by the mandatory access control. Fig. 10, shows the components of the SELinux module. An administrator, using an interface for policy management, sends a policy to the SELinux File System, which in turn sends the policy to the Security Server. The Security Server determines whether access to a resource is allowed or denied. The purpose of the access vector cache is to store the rules of the policy that are frequently evaluated to obtain a response as quickly as possible, avoiding using the security server repeatedly.

### A. SELinux OPERATION

At a high level of abstraction users interact with the hardware through the operating system. At the operating system level a user sesion is associated with a *process*, and at the access control level that process is associated with the *subject* term. At the operating system level a *resource* is associated to a file, directory, socket, etc. At the access control level that resources are associated with the *objects* term. Objects are

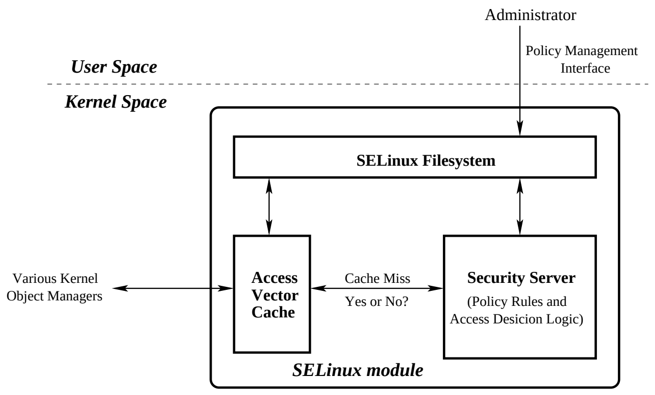

Fig. 10. SELinux module architecture

grouped into classes, and each class specifies some actions (read, write, etc.) that can be performed on such objects. In SELinux each subject and object have a security context associated to them. A security context is formed by a set of attributes based on the mandatory access control models. The *identity* attribute is taken from the UI model, the *domain* and *type* attributes are taken from the TE model. The *role* attribute is taken from the RBAC model and the *level* attribute is taken from MLS model. An example of security context is *user_u:system_r:system_t:s0*, where the values *user_u*, *system_r*, *system_t*, *s0* correspond to identity, role, type, and level attributes respectively, where these concepts are defined as follows:

*Identity.* Each user (subject) has an identity normally represented by the user name.

*Role.* Within an organization a user can perform various activities, which are based on the tasks entrusted to it. The role attribute is used to classify the functions assigned to a user. Depending on the activities developed by a user, he/she can take one or more roles.

*Domain* and *Type.* The processes or subjects are grouped into domains, and system resources (objects) are grouped into types. A domain can have access to a type for a restricted set of actions. Transitions can be done between domains.

*Level.* The level attribute is used to define access control as follows:

- A subject *s* can have access to an object *o* for a set of actions, if and only if the subject's level *s* is greater than or equal to the object's level *o*.

Fig. 11, shows the relationships that exist between the different security attributes. An *identity* attribute can access one or more *role* attributes, a role attribute can access one or more *domain* attributes, and a domain attribute combined with a *level* attribute can have access to one or more *type* attributes with the same level value. Other important aspects are the dominance between roles and the transition between domains [21].

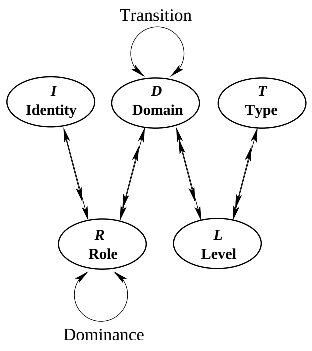

Fig. 11. Interactions between attributes

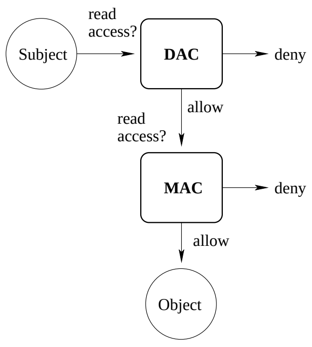

Fig. 12. Interaction between DAC and MAC

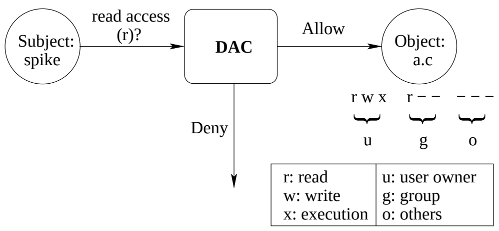

Fig. 13. Discretionary access control by ACL

## B. ACCESS CONTROL ON SELinux

When a subject tries to access an object to perform some action, the discretionary access control is the first requested instance. If the discretionary access control allows access to the object, then the mandatory access control is requested to allow or deny access to the object. If discretionary access control denies access to the object, then the mandatory access control is not invoked (see Fig. 12).

Fig. 13, shows the interaction between a subject and an object through discretionary access control. In SELinux as well as traditional Linux, ACL is the model used for discretionary access control. This model is based on the identity of the subject and uses the set of permissions for the owner, group, and others, respectively. Nowadays, ACLs are still employed by their easy deployment, configuration, and use. A disadvantage of ACLs is the lack of a fine granularity for access.

Fig. 14, shows the interaction between subject and object by the mandatory access control. This access control type analyzes the subject and object security contexts. The subject

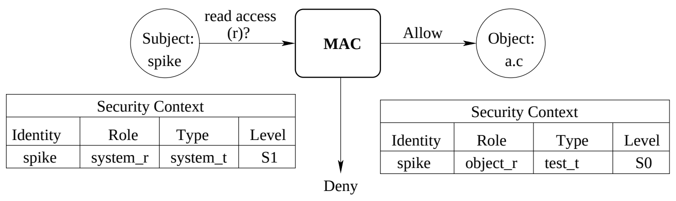

Fig. 14. Mandatory access control on SELinux

will have read access on the object *a.c* if and only if there is a rule that allows the *system_t* domain attribute to access the objects with the *test_t* type attribute, and where the level attribute of the object is lower than the subject. Another access possibility exists if the *system_r* attribute role dominates a role that has enabled access to a domain attribute which can access to the *test_t* type attribute, and the level criteria is satisfied. Another possibility to access an object may be given if the *system_t* domain attribute has permition to transition to a domain allowed to access the *test_t* type attributes, and the level criteria is satisfied. The object_r role attribute is the role for system objects.

## C. SELinux POLICY

Nowadays, SELinux policies are administered locally and the policy rules are grouped in several directories, which contain a set of files. Fig. 15 shows the set of directories used to store local SELinux security policies. One directory is associated with a class, object, and permission structures defined by the Flask architecture. Similarly, two directories are associated with the rules, based on TE and RBAC models. The directory associated with RBAC rules is combined with the directory for the user statements. The constraints directory contains the set of rules that deny others. For example if there is a rule that allows access, may be a constraint invalidating such rule. The security context directory specifies to each element of files and directories.

SELinux confines processes in sandboxes (i.e. a least privilege environment) to limit them to perform their specific functions. For example, the apache process can be confined into a domain to only have access to their resources like configuration files and web pages. Then access control only allows the access to resources needed by the apache domain and denies the access to other resources. The following rules confine the apache process for the resources to it:

- Assignment of a security context to the web directory /var/www.
  - system_u:object_r:web_resource_type:s0
- Domain and type definition rules.
  - type apache_t, domain;
  - role apache_r types apache_t;
  - type apache_exec_t, file_type, exec_type;
  - allow apache_t web_resource_type : file {write read};

The rules given above are part of the set of rules needed to confine the apache process, this rules define a *apache_r* role, a *apache_t* domain, and a *apache_exec_t* type. The apache_t domain can have access to read and write files

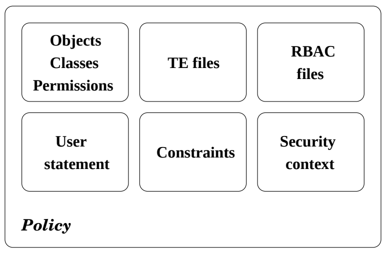

Fig. 15. SELinux local policy structure

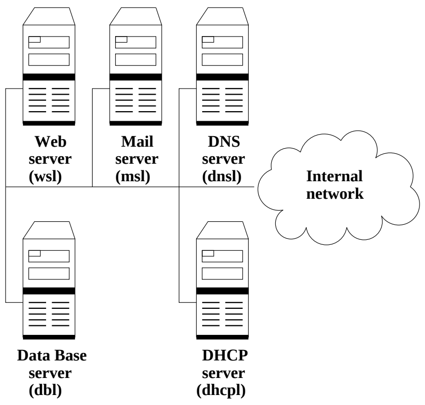

Fig. 16. SELinux distributed environment

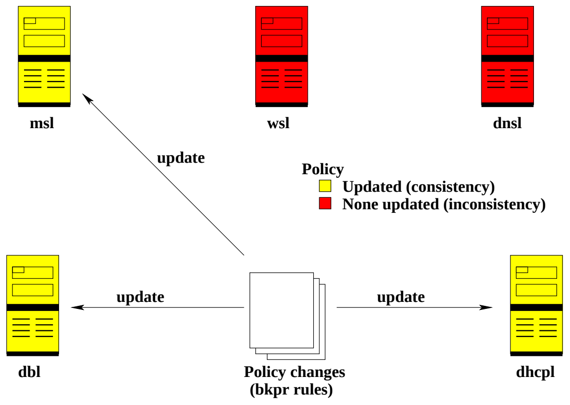

Fig. 17. Manual SELinux policies update

TABLE I
ADMINISTRATIVE ROLES

| Role | Description |
|---|---|
| bkpr | Backup role |
| dbr | Data base server admin |
| dhcpr | DHCP server admin |
| dnsr | DNS server admin |
| msr | Mail server admin |
| systemr | Operating system admin |
| wsr | Web server admin |

TABLE II
ROLES AND HOSTS

| | Hosts | | | | |
|---|---|---|---|---|---|
| Roles | dbl | dhcpl | dnsl | msl | wsl |
| bkpr | x | x | x | x | x |
| dbr | x | | | | |
| dhcpr | | x | | | |
| dnsr | | | x | | |
| msr | | | | x | |
| systemr | x | x | x | x | x |
| wsr | | | | | x |

TABLE III
ROLES AND USERS FOR DBL HOST

| | Users | | |
|---|---|---|---|
| Roles | spike | john | bob |
| bkpr | x | | x |
| dbr | x | | |
| dhcpr | | | |
| dnsr | | | |
| msr | | | |
| systemr | x | x | |
| wsr | | | |

of web_resource_type type. For the design and analysis of security policies, administrators must have experience with access control types and models. Another piece of knowledge administrators need to have for the design of SELinux policies is the macro preprocessor m4. For a good policy management, an administrator needs to know the correct configuration procedures.

In SELinux distributed environments each host has a policy associated to it. Fig. 16 shows an example of a distributed environment of SELinux hosts, where each host is responsible for providing a network service. The administrative roles that can be exercised in such environment are cited in Table I; Table II cites the roles that can be used in each host. Each host is associated with a set of users that can play one or more roles. Table III shows the users and roles for host *dbl*.

If a set of rules for some policies change, and those changes are valid in a number of hosts, the policy administrator must update the policy in all hosts where those changes are valid (See Fig. 17). That is certainly not a desirable situation because the administrator may have not updated all hosts. The consistency, authenticity, integrity, availability, and confidentiality aspects should be present in the policy update process. For these reasons, it is necessary to develop tools that aid the systematization of the policy management process. In the next section we present an architecture that solves the problems addressed in this section.

## V. PROPOSED ARCHITECTURE

To solve the problems generated by using SELinux in distributed environments, we propose an architecture that systematizes the policy management process. We eliminate policy inconsistencies by using a host that operates as a policy server. We achieve authenticity, integrity, and confidentiality in the policies update process by means of using Kerberos system. Fig. 18 shows the integration of the policy server and Kerberos system.

In case a policy changes, the policy server must update it in all systems associated with the changes. The policies must

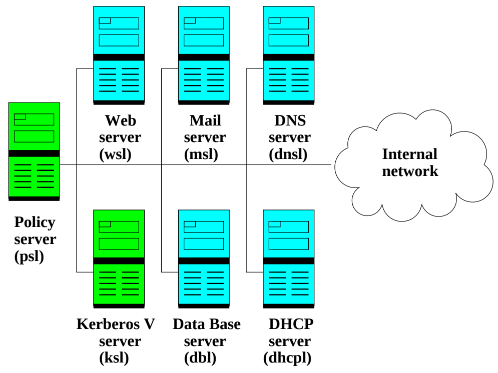

Fig. 18. Proposed architecture for policy management

be available in the update process, a redundant policy sever is proposed to provide this characteristic. Other work about secured distribution of contents on distributed environments are cited in [27], it uses the Public Key Infrastructure, and SEED encryption algorithm to protect the copyrights of digital contents.

## A. POLICY SERVER

The policy server centralizes the SELinux policies to avoid inconsistencies in the distributed environment when any rules change. This server reports to each host when the sets of policies it stores change. At that point, a policy update process takes place. Fig. 19 shows that a centralized policy segmented and distributed to the different hosts in the distributed environment.

### *Location Rules*

The term location is used to designate a host integrated in a SELinux distributed environment. To integrate the location concept in SELinux policies we use two rule extensions to express relationships between users, roles, and locations.

### *Rules involving locations and roles*

Syntax: location $l$ roles $r_1, r_2, ..., r_n$

Semantics: *location ws_l roles* { *system_r, ws_r, bkp_r*}*;*. The rules for locations and roles are expressed in the location statements module, these rules specify the set of roles that can be exercised in the locations.

### *Rules for users, locations and roles*

Syntax: user $u$ location $l$ roles $r_x$

Semantics: *user bob location ws_l roles* {*bkp_r*}*;*. The rules for users, locations and roles are expressed in the user statements directory. These rules specify the roles that a user may play in a given location.

### *Algorithms to produce local policies*

The introduction of location policies require a modification in all SELinux policy mechanisms. Instead of modifying those mechanisms, which involve modifying the compiler, loader, and interpreter of access control policies, we allow administrators to introduce policies in the augmented syntax. Policies using the augmented syntax are stored in the policy

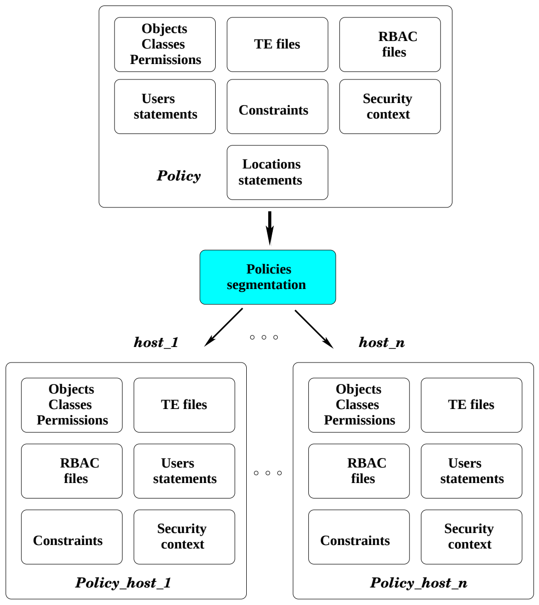

Fig. 19. Policy structure for locations

```
Input: Set of files for Locations statements directory
Output: Set of directories and files for host policies
foreach rule (location $l$ roles $r_1, r_2, ..., r_n$) do
    Create $l$ directory with a local policy structure;
    foreach  i = 1 to n do
        $R_l = R_l \leftarrow R_l \cup r_i$;
    end
end
```

**Algorithm 1**: Location algorithm

```
Input: Set of files for Locations directory
Output: user - role rules
foreach rule (user $u$ location $l$ roles $r_1, ..., r_m$) do
    if $r_1, ..., r_m \in R_l$ then
        Append (Roles (user $u$) $\{r_1, ..., r_m\}$) in users;
    end
end
```

**Algorithm 2**: User algorithm

server. When a host requires a policy, the server processes that rule, and send it to a host in the syntax required by SELinux. This mechanism allows hosts to ignore the location information managed by the policy server. If a policy does not allow a user to play a given role in a given host, a denial signal is sent, and no policy is issued to the host. On the other hand, if access is granted, the complete rule (in SELinux syntax) is sent to the location.

The User and Location algorithms are used for producing local policies. For each rule for locations and roles, the locations algorithm generates a directory $l$, containing domains, file_contexts, objects class and permissions, RBAC and TE directories, and the file for users statements. Similarly, the algorithm generates the set $R_l$, which defines the roles that can be played on location $l$. For each rule for users, roles and locations, the User algorithm adds a new rule user $u$ roles $r_1, ..., r_m$ in the user.local file contained in the $l$ directory.

### *Policy assignment/update process*

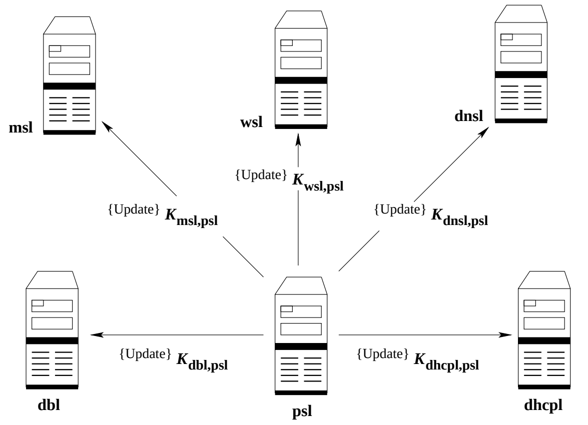

Fig. 20. Update policy process

When a host joins a distributed environment, it requires to identify itself in that environment. In the example shown in Fig. 18, the server responsible for providing the DHCP service assigns the following parameters needed for distributed operation:

- Network: ip, netmask, dns, dhcp, ntp.
- Kerberos server: Realm Name, Master, and Slave.
- Policy server: Master and Slave.

Once the client has obtained such parameters, it proceeds to interact with the Kerberos system to authenticate itself. For the service phase, the client sends a session key $K_{C,S}$ to the policy server, who will provide integrity, and confidentiality in the policy update process. In message 1 of the check consistency protocol protocol a client sends to the policy server a md5 value of their actual policy structure. The policy server compares the md5 value received and a md5 value obtained for the corresponding structure of the policy assignment to the client. If the two md5 values are equal, then the policy has not been changed; if they are different the client policy must be updated.

```
C-Client, and S-Server
1. C → S : $\{md5(policy)\}_{K_{C,S}}$
 - If md5 values are equal then
2. S → C : $\{Ok\}_{K_{C,S}}$
 - else
3. S → C : $\{Update\}_{K_{C,S}}$
4. S → C : $\{Newpolicy\}_{K_{C,S}}$
5. C → S : $\{Ok\}_{K_{C,S}}$
```

Protocol 1: Check consistency protocol.

*Update policy process*

When one or more policies change, the policy server is responsible for sending a message 3 to notify that policies should be updated. Such notification is shown in Fig. 20. In this case is necessary analyze what to do with the active user sessions, considering the possibility of new rules that affected their activities.

*Security architecture implementation*

The policy server is fully implemented, delivering segmented policies to the hosts. Policies segmentation and Kerberos Implementation are ready. The clients on each host have also been implemented. Hosts deliver requests to policy server, receive the policies, and integrate them to the SELinux kernel module.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper we address the policies inconsistency problem that SELinux presents in distributed environments. We propose an architecture for a distributed environment comprises a centralized a policy server to integrate and update policies on each host. On the policy update process the Kerberos V protocol provides authenticity, integrity, and confidentiality. The security policy server provides consistency on policies; availability is obtained with a redundant policy server.

An interesting question is what to do when a policy is in use and it is necessary to update it. One solution is to block the system to give way to the update policy process, but the work of the users would be affected. Another solution to this problem is to inform users of the need of policy update, and give them a prudent time to log out and ensure that their activities are not affected.

Users can play their assigned roles through local or remote sessions. Access control does not distinguish between local or remote sessions. Our implementation does not take into account the idea of connecting "to" a host "from" a different location "playing" a given role. The solution to this problem is under development. Another direction of work is to develop a policy server with heterogeneous operating systems. Another interesting idea is to integrate into the access control security policies a quantitative factor and time mark to limit the misuse of resources.

At the moment of writing this paper, authors do not know any work addressing the problem of inconsistency in security policies in distributed environments.

## REFERENCES

[1] Silberschatz, Galvin, and Gagne. *Operating Systems Concepts*. John Wiley and Sons, seventh edition, 2005.
[2] Departament of Defense. Department of defense trusted computer system evaluation criteria. Technical report, December 1985. DoD 5200.28-STD.
[3] Peter Loscoco, Stephen Smalley, Patrick A. Muckelbauer, and Ruth C. Taylor. The inetability of failure: The flawed assumption of the security in moderm computer enviroments. 2000. NSA.
[4] Daniel Lopresti Lucas Ballard, Fabian Monrose. Biometric authentication revisited: Understanding the impact of wolves in sheep's clothing. 2006. 15th USENIX Security Symposium.
[5] P.C. van Oorschot Sonia Chiasson and Robert Biddle. A usability study and critique of two password managers. 2006. 15th USENIX Security Symposium.
[6] Saar Drimer and Steven J. Murdoch. Keep your enemies close: Distance bounding against smartcard relay attacks. 2007. 16th USENIX Security Symposium.
[7] MDLIN TEFAN VLAD VALENTIN SGRCIU. Smart card technology used in secured personal identification systems. Bucharest, Romania, 2006. Proceedings of the 5th WSEAS Int. Conf. on DATA NETWORKS, COMMUNICATIONS & COMPUTERS.
[8] Marianne Swanson and Barbara Guttman. Generally accepted principles and practices for securing information technology systems. Technical report, National Institute of Standards and Technology, September 1996.
[9] Morrie Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold, 1998.
[10] Red Hat, Inc., PO Box 13588 Research Triangle Park NC 27709 USA. *Red Hat Enterprise Linux 4 SELinux Guide*, 2005. http://www.redhat.com.

[11] DOD. Departament of defense trsuted computer system evaluation criteria. Technical report, Departament of Defense, 1985.
[12] Carl E. Landwehr. Formals models for computer security. *Computer Surverys Vol. 13 ACM*, 1981.
[13] B. W. Lampson. Protection. *Proc. 5th Princeton Symp. Information Sciences and Systems*, 1971.
[14] Andreas Grunbacher. Posix access control lists on linux. Technical report, 2003. Suse Administration Guide.
[15] Peter G. Smith. *Linux Network Security*. Charles River Media, 2005.
[16] D. Elliot Bell and Leonard LaPadula. Secure computer systems: Mathematical foundations. Technical report, 1973. MITRE Technical Report 2547, Volume I.
[17] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. April 1987. IEEE Symposium on Computer Security and Privacy.
[18] Biba K. J. Integrity considerations for secure computer systems. Technical report, 1977. MITRE Technical Report 3153.
[19] David F.C. Bwever and Michael J. Nash. China wall security policy. May 1989. IEEE Symposium on Research in Security and Privacy.
[20] W. E. Boebert and R.Y. Kain. A practical alternative to hierarchical integrity policies. *Proc. of the 8th National Computer Security Conference, Gaithersburg, MD*, 1985.
[21] Lee Badger, Daniel F. Sterne, David L. Sherman, Kenneth M. Walker, and Sheila A. Haghighat. A domain and type enforcement unix prototype. June 1995. Unix Security Symposium Proceedings.
[22] Kenneth M. Walker, Daniel F. Sterne, M. Lee Badger, Michael J. Petkac, David L. Sherman, and Karen A. Oostendorp. Confining root programs with type enforcement (dte). 1996. Sixth USENIX UNIX Security Symposium.
[23] Serge E. Hallyn. Domain ant type enforcement for linux. 1997.
[24] David Ferraiolo and Richard Khun. Role-based access control. 1992. Proceedings 15th National Computer Security Conference.
[25] Ravi Sandhu, David Ferriaiolo, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramoulli. Proposed nist standart for role-access control. 2001. ACM Transactions on Information and System Security Vol. 4 No. 3.
[26] Peter Loscoco and Stephen Smalley. Integrating flexible support for security policies into linux operating system. 2001. NSA.
[27] UNG-MO KIM SEUNG-BAE YUN, HYUNK-JIN KO. The design and the implementation of web service security system for the secured distribution of digital contents. Madrid, Spain, 2006. Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases.

**Juan J. Flores** Dr. Flores got a B.Sc. degree in Electrical Engineering from the Universidad Michoacana in 1984. In 1986 he got a M.Sc. degree in computer science from Centro de Investigacion y Estudios Avanzados, of the Instituto Politecnico Nacional. In 1997 got a Ph.D. degree in Computer Science from the University of Oregon, USA. He is a full time professor at the Universidad Michoacana since 1986. His research work deals with applications of Artificial Intelligence to Electrical Engineering, Computer Security, and Financial Analysis. He is a member of the Sistema Nacional de Investigadores. He is a member of the Mexican Academy of Sciences, the Association for Computing Machinery, ACM, and of the group Computational Finance & Economics Network. He was an invited Professor-Researcher at the University of Oregon in 2005/2006.

**Pedro Chavez Lugo** degree in Electrical Engineering from the Universidad Michoacana in 1998. In 2005 he received MSc degree in Electrical Engineering Computer Systems option from the Universidad Michoacana. He is Professor at the Informatica Administrativa from Facultad de Contaduria y Ciencias Administrativas, Universidad Michoacana. His research work includes intrusion detection, information security, programming languages and access control, operating systems.

**Juan Manuel Garcia Garcia** received a MSc degree in Computer Science from the Universidad Nacional Autonoma de Mexico (UNAM) in 1994 and in 2003, a PhD degree in Computer Science from the Centro de Investigacion en Computacion (CIC) of the Instituto Politécnico Nacional (IPN), Mexico. Since 1991 he is a Full Time Professor at the Computer Systems Department of the Instituto Tecnológico de Morelia. His research interests includes information security, cryptography and information theory.