Reprinted from the

Proceedings of the
Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Wild Open Source, Inc.*

Mandatory Access Control for Linux Clustered Servers

Miroslaw Zakrzewski
Open Systems Lab
Ericsson Research
8400 Decarie Blvd
Town of Mont Royal, Quebec
Canada H4P 2N2
Miroslaw.Zakrzewski@ericsson.ca

Abstract

In today's world, the use of computers and networks is growing and the vision of a single infrastructure for voice and data is becoming a reality. However, with different technologies and services using the same networking infrastructure, the realization of this vision requires higher levels of security to be implemented in computer systems. Current security solutions do not address all of the security challenges facing today's computer systems, including clustered platforms, in one comprehensive and coherent fashion.

This paper presents the previous work done in the area of access control and then focus on new mechanisms for clustered Linux servers as part of the research project at the Ericsson Open Systems Lab. In this paper, we address the design and implementation of a framework for the mandatory access control in the distributed security infrastructure (DSI). The ongoing work is mainly based on the Flask architecture and the Linux Security Module (LSM) framework with a focus on Linux clustered servers. The paper also addresses the effects of the cluster security on the performance of the distributed system, since enforcing security may introduce degradation in the performance, an increase in administration, and some annoyance for the user.

We are implementing cluster-aware access control mechanisms in the Linux kernel. We expect that our work will help position Linux as a secure operating system for clustered servers.

1 Introduction

The security of computing systems could be enforced on different levels of the computing environment such as hardware, operating system, application and network level. The primary subject from the security prospective is the operating system level, being a fundamental piece of the security of every computer system, and a critical point of failure for the entire system. Currently implemented security mechanisms of operating systems are based on user privileges and are inadequate to protect against the various kinds of attacks in today's complex environments. To address these problems, security in operating systems has long been a well-researched topic, which formulated various security models and policies.

Various research results have shown that mandatory security provided by the operating system is essential for the security of the whole system [3]; furthermore, they proved that mandatory access control mechanisms are very efficient in supporting complex relationships between different entities in the computing environment.

Several attempts were made to reach a very secure platform. For instance, The FLASK architecture [5,12] (on which SE Linux [17] is based) was created as an attempt to serve as a generic architecture for the mandatory access control. An important design goal was to provide flexible support for security policies. The FLASK architecture achieved the goal by separating the security policy from the enforcement mechanism and by having security checks transparent to the applications. Another attempt from SE Linux was to prototype the access control in the Linux kernel.

However, the existing solutions, including Flask and SE Linux, do not address the access control in distributed environments. One such environment is computer cluster. In our context, a cluster is defined as a collection of interconnected stand-alone computers working together to solve a problem as a single computer. The cluster can appear as a single system to users and applications. Since from the logical point of view we can see a cluster as a single entity, we should apply this definition to the cluster security as well and treat the subjects and resources as if they were located on one virtual machine.

Even though new security approaches, such as FLASK, address the problem of mandatory access control between subjects and resources belonging to the same processing node, they are still missing the mandatory, finer-grained security checks between the subjects and resources belonging to different nodes.

There exist many security solutions for clustered servers ranging from external solutions, such as firewalls, to internal solutions such as integrity checking software. However, there is no solution dedicated for clusters. The most commonly used security approach is to package several existing solutions. Nevertheless, the integration and management of these different packages is very complex, and often results in the absence of interoperability between different security mechanisms. Additional difficulties are also raised when integrating these many packages, such as the ease of system maintenance and upgrade, and the difficulty of keeping up with numerous security patches and upgrades.

Carrier class clusters have very tight restrictions on performance and response time. Therefore, much pressure is put on the system designer while designing security solutions. In fact, many security solutions cannot be used due to their high resource consumption.

In a distributed environment, subjects and resources can be located anywhere on the network so the relations between them are more complex.

In this paper, we present the preliminary results developing the Linux security module (LSM) that links all the nodes of the cluster in a transparent fashion; the Linux security module is also referred to as the distributed security module. The security module enforces the security checking on a node between subjects and resources belonging to the same or different nodes of the cluster. The distributed security module is a part of the distributed security infrastructure (DSI) and cannot be used without it. The DSI decides about the security policy and defines mechanisms that control the module. In the next section, a brief description of the distribution security infrastructure is introduced.

2 Distributed Security Infrastructure

2.1 DSI Characteristics

As part of a carrier class Linux cluster, DSI [6] must comply with carrier class requirements such as reliability, scalability, and high availability. Furthermore, DSI supports the following requirements: coherent framework, process level approach, pre-emptive security, dynamic security policy, transparent key management, and minimal impact on performance.

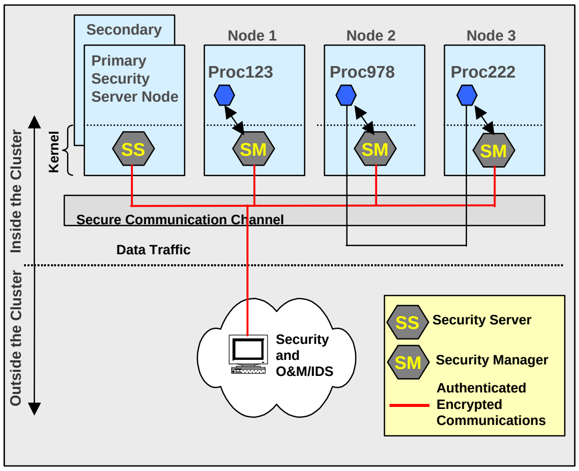

Figure 1: Distributed Architecture of DSI

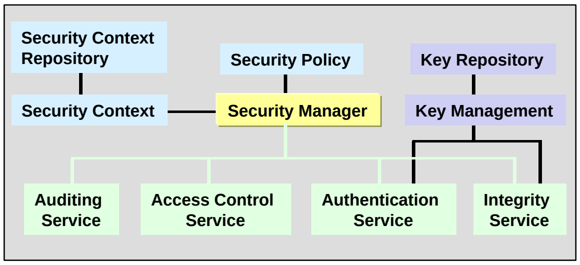

Figure 2: DSI Services

2.2 DSI Architecture

DSI has two types of components: the management components and service components. DSI management components define a thin layer of components that includes a security server, security managers, and a security communication channel (Figure1). The service components define a flexible layer, which can be modified or updated by adding, replacing, or removing services according to the needs.

The security server is the central point of management in DSI, the entry point for secure operation and management, and intrusion detection systems from outside the cluster. It is the central security authority for all the security components in the system. It is responsible for the distributed security policy. It also defines the dynamic security environment of the whole cluster by broadcasting changes in the distributed policy to all security managers.

Security managers enforce security at each node of the cluster. They are responsible for locally enforcing changes in the security environment. Security managers only exchange security information with the security server.

The secure communication channel provides encrypted and authenticated communications between the security agents. All communications between the security server and the outside of the cluster take place through the secure communication channel.

The DSI architecture at each node is based on a set of loosely coupled services (Figure 2).

Each service, upon its creation, sends a presence announcement to the local security manager, which registers these services and provides their access mechanisms to the internal modules.

There are two types of services: security services (access control, authentication, integration, auditing) and security service providers (for example secure key management) that run at user level and provide services to security managers.

3 Cluster Access Control

3.1 General Discussion

In general, the Access Control Service (ACS) can be seen as a layer (software, hardware) that enforces the security policy as a two-parameter function. It relies on the notions of subject (or access request initiator), resource (or target), environment, decision, and enforcement.

A subject could be a program or process and a resource can be a file or a communication resource. The same process can be a subject in one access control operation and a resource in another.

An access control could be interpreted as a matrix where one axis is the list of all possible subjects and the other is the list of all possible resources. The entries in the matrix define the permissions. Even for reasonable-sized systems the matrix gets complicated, very fast so there is a need to reduce its complexity. In order to do this, the term class is introduced. Class groups the subjects and resources, which have the same permission and create only one entry for them in the matrix.

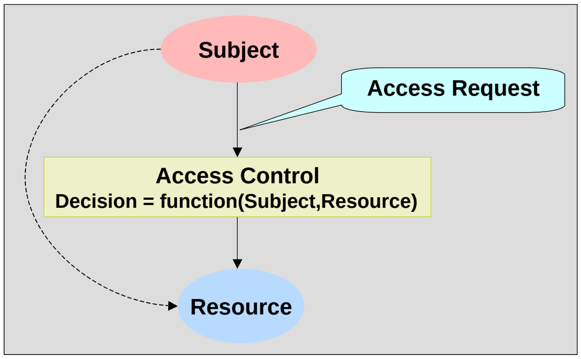

Figure 3: Access Control

When a Subject tries to access a Resource (Figure 3), the access request is intercepted by the access control layer and based on the subject's rights, the access either is granted or not. The access control of an operation system is usually added in the system call layer (Linux). This is ideal for the operating system because it makes the access control transparent for the applications, and more secure because it's located in one of the lowest software layers, in addition it's fast because it's embedded into the operation system.

The ACS assumes that the subjects have been properly authenticated. One of the important characteristics of the access control for clusters is that it allows verifying the access control privileges even when subjects and resources are located on different nodes in the cluster.

The ACS that runs on the cluster processors is comprised of two parts:

1. A kernel-space part: This part is responsible for implementing both the enforcement and the decision-making tasks of access control. These two responsibilities are separated. The kernel-space part maintains an internal representation of the information upon which it bases its decisions. This information (security pol-

icy) is supplied by the security server and stored in the local memory for fast access (hash table). On Linux, the kernel-space part is implemented as a Linux Security Module (LSM).

2. A user-space part: This part has many responsibilities. It takes the information from the Distributed Security Policy and from the Security Context Repository, combines them together, and feeds them to the kernel space part in an easily usable form. It also takes care of propagating back alarms from the kernel space part to the security manger, which will feed them to the Auditing and Logging Service and if necessary propagate to the security server through SCC.

Both parts, kernel-space and user-space, are started and monitored by the local Security Manager (SM) on each node. The SM also introduces them to other services and subsystems of the DSI infrastructure with which they need to interact.

The ACS aims to provide fine-grained access control (at the system call level). It respects the minimization principles of least privilege to limit the propagation and damage caused by eventual security breaches. As such, it provides defense in depth.

The ACS that is running on a processor must make as little assumptions as possible about other processors, including whether they have been compromised. For that reason, an ACS instance is always the one making access decisions about resources that are local to its processor. For the initial design of the ACS, only grant/deny decision will be considered. Other more involved decisions would involve rate limiting and total usage limiting. Actions other than access control decision, such as interposition and active reactions, are not implemented either.

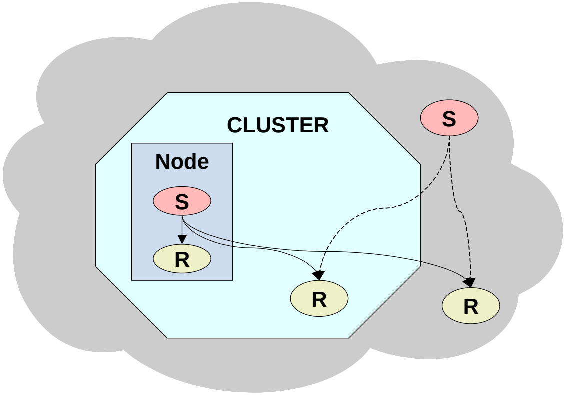

Figure 4: Cluster Access Control

3.2 Cluster Access Types

The distributed environments allow that the actors of ACS (subject and resource) can be located anywhere in the cyber space.

Based on their mutual location in the cluster (Figure 4), and to reduce the complexity when analyzing access control, we can distinguish the following types of the access control:

- Cluster Local Access: Both subject and resource are located on the same node in the cluster
- Cluster Remote Access: Both subject and resource are located on different nodes inside the same cluster
- Cluster Outside Access: Subject is located on a node inside cluster and resource is outside the cluster or Subject is located outside the cluster and resource on a node inside the cluster.

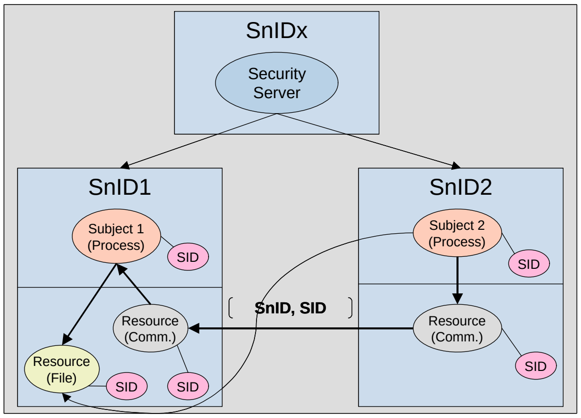

Figure 5: Distributed Access Control

- No Cluster Access: Both subject and resource are outside the cluster

The above classification allows us to reduce the complexity of the cluster access control by classifying the various access approaches. First, we analyze the Cluster Local Access and next we will move to the Cluster Remote Access. The Cluster Outside Access and No Cluster Access are out of the scope of this paper.

3.3 Distributed Access Control Architecture

Finding an efficient solution to the cluster mandatory access control is a complex task. There are many factors involved in defining the access rights because the subjects and resources can be located on different nodes in the cluster. To simplify the relationships, we can handle the access control in two levels:

1. Local when subject and resource are located on the same node, and

2. Remote when subject and resource are located on different nodes.

For local access control, the access rights are the functions of the security IDs of the subject (SSID) and the resource (TSID). This is based on the FLASK architecture:

```
Access = Function (SSID, TSID)
```

The FLASK architecture can serve as a solution for the single node processing. When the nodes are presented as a cluster, security solutions become more complicated. In this case, we extend the FLASK architecture to the cluster remote access model. One of the new parameters is the security node ID (SnID) (Figure 5), which defines the node in terms of the security. Access rights are no more just the function of the subject and target security ID's, but as well, the function of the security node ID.

```
Access = Function
(SSnID,SSID,TSnID,TSID)
```

An important part of the distributed system is the network, which spans the nodes of the cluster. To apply the access control functions in the cluster, there must be a way to pass the security parameters between the nodes in a transparent fashion. In our research, we try to find the appropriate architecture for this problem as well.

Our prototype is based on a cluster of Linux machines and the implementation of the mandatory access control will by exercised in the Linux kernel. By implementing the mandatory access control inside the kernel, we can achieve security transparency in the system.

Another functionality of the access control is to be able to generate alarms in case of intrusion detection. When the security module detects the intrusion, an alarm notification will be passed to the security manager and later to the security server. Based on the severity of the alarm, the security server will take an action.

An example of the action will be a change of the security node ID, loading a new security policy, or declaring a node compromised and disconnecting it from the cluster. In the most severe case, the security server may ask ACS to block all accesses except the security path of the security manager.

4 DSI Security Module

Our security enforcement software for Linux is built as a Linux module and works in the kernel space. We based our development on the Linux Security Module (LSM) infrastructure (security hooks) introduced in the Linux kernel.

LSM framework does not supply any additional security in the Linux kernel. It only provides the infrastructure to support the security development as Linux modules. The LSM kernel patch adds security fields to kernel data structures and inserts calls (called hooks) at special points in the kernel code to perform a module specific access control. LSM adds methods for registering and un-registering security modules, and a general security system call that allows the communication between user programs and the LSM for security aware applications. Each LSM hook is a function pointer in a global structure called `security_ops`. Because the hooks are embedded in the kernel and are called even before a security module is installed, this structure is initialized to a set of functions provided by a dummy security module. These functions are just placeholders for more useful security mechanisms that can be loaded as a Linux module. A `register_security` method is introduced to allow a security module to set its own security functions (to overlay the dummy functions). An `unregister_security` method is used to return to the dummy functions.

The LSM methods are organized into two categories:

- Hooks to handle the security fields
- Hooks to perform access control

We started the development with the kernel 2.4.17 [13] and the appropriate security patch (`lsm-full-2002_01_15-2.4.17.patch` [15]). The DSM module cannot act alone and rely on the services supplied by DSI. DSM only enforces the access control but the policy is decided by the DSI security server. The security server is responsible for giving the security policy to the security module. The security server (SS) is responsible to supply the security node ID to each node of the cluster as well. Sending the security node ID to the node of the cluster means that the node is part of the cluster from the security point of view and it can start the security operation. Before the security node ID is sent to the cluster node, all the security checks are disabled on this node.

DSM takes security decisions based on the security policy decided by the security server and the security identifiers (SID) assigned to each subject and resource. Security Identifiers are non-persistent and are meaningful only on the local node. The security server provides functionality for converting a SID to its corresponding security context. All the entities for which security is being enforced are divided into security classes. A security class is a distinct type of resource with a distinct set of legal operations, for example, a process, a file, etc.

When a security decision must be taken, the security IDs of a subject and a resource are extracted from their kernel representations and will be used for the security access decision. For efficiency, the security policy is represented in the kernel memory.

4.1 Labels

As already mentioned, all the subjects and resources must be labeled. Since the security module can be loaded run-time, we distinguish two modes of subjects labeling. Before the module is loaded there are no labels attached to any subject or resource in the system. At the module initialization time, all the running tasks are scanned and the labels are attached to them. When a new process is created after the security module is loaded, the security hooks are used to do the labeling.

Because Linux stores the process descriptor and the Kernel Mode process stack in a single 8KB memory area, we can use this fact and avoid allocating memory for labeling the subjects (Figure 6). The other labels are attached to the resources run-time, which implies that the module checks if the label is there. If the label is not attached, a new label will be created.

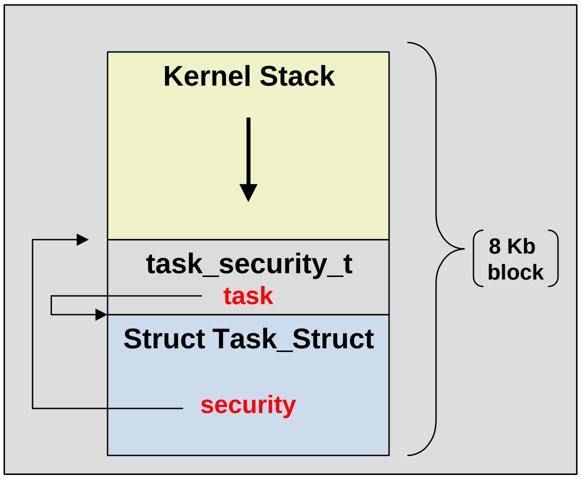

Figure 6: Task Label

4.2 Network Labels

Because the access in the cluster can be performed from a subject located on one node to the resource located on another, there is a need to control such accesses as well.

When a process on one node makes an access to a resource on another node, first the local access to the communications resources (socket, network interface) is checked. When the local access is granted then the message can be sent to the remote location. In order to identify the sending subject, the Security Node ID (security node identifier) and the Security ID of the subject (security subject identifier) are added to the message. For the purpose of this exercise, we use the IP protocol for the security information transfer. A new option is added after the IP header based on the hooks in the IP protocol stack. On the receiving side, these two information (Security Node ID and Security SID) are extracted (based on the hooks in the IP stack) and are used to build the network security ID (NSID).

```
NSID = Function (SnID, SID)
```

This function is specified by the security server in form of the conversion table. The receiving side looks up into the table by specifying SnID and SID and extract the Security Network ID. Now the security network ID can be used as a local label to all the access controls.

For instance, a client process tries to access a server's process. The client node does not know what is the security of the server node, so it can only perform access control checks based on the security attributes of the communications resources (sockets, network interface). The server node can perform access control checks based on the security attributes of the client process, the source node, and the server process. When a process attempts to accept a connection or receive a packet, if the policy prohibits the server process from receiving data from the client process, the connection or the packet is dropped and alarm is generated.

4.3 Implementation status

We are in the process of building the working prototype of the cluster security infrastructure. The kernel module has been implemented where the subjects (tasks) have been labeled. The local access to the communication objects (sockets) has been implemented and we are currently working on the remote access implementation.

In the current implementation, the security information is added to the IP message after the IP header as an option. There is no implementation of the interface to other parts of the distributed security architecture. The actions of the security server are simulated by the user mode programs (load policy, load security node ID). The alarms generated by the distributed security module are sent to the special user mode program as well. The current implementation is not optimized for the performance and it is built in order to check the overall logic of the cluster security.

5 Performance Challenges

Enforcing security does not come free; there is always a performance price to pay. At the same time, an over secured system is almost unusable; therefore the security introduced to the system must be properly balanced. This section discusses the impact of the security implementation on the overall performance of the system.

We performed testing for three different kernel configurations: the first testing was done with kernel 2.4.17; the second was done with the same kernel and our security module loaded plus the IP packet modification; the third was done with the same kernel and the security module but without IP packet modification. These tests were executed on a Pentium III 650 MHz Dell laptop with 265 MB RAM.

5.1 Test Types

We performed three types of testing: process creation with fork, UDP local access, and UDP remote access. The purpose of the testing was to get a preliminary performance evaluation of the security module, to answer the question of how much performance we lose when adding extra security features. The UDP tests were performed with and without IP packet modification in order to see how much performance was lost during IP packet modification. In the following subsection, we explain the testing procedure per testing type.

`Process Creation Testing`

This test measures the time a process can fork a child that immediately exits. The parent process loops 100,000 performing fork and wait calls. The test was performed 5 times and the average was calculated. Later the average time of the single loop (fork, wait) was calculated.

`UDP Local Access Testing`

The UDP Local Access test measures the time needed by a process to send a UDP message.

This test sends 500,000 UDP messages in a loop. The test was performed 5 times and the average was calculated. Later the average time of the single loop (send) was calculated. The sending process does not check if the message was sent outside the node; in addition, it does not wait for the confirmation. In this case, it is not important whether the server has DSM installed or not.

UDP Remote Access Testing

The UDP Remote Access test measures the time needed by a process to send a UDP messages and receive a UDP response from a server. The client process will send a new message after receiving the confirmation from the server. It is important, in this case, that the server runs the DSM software for the permission to be checked on the receiving side. In this test, the second server is a Pentium II 300 MHz desktop with 128 MB RAM. This test sends and receives 100,000 UDP messages in a loop. The test was performed 5 times and the average was calculated. Later the average time of the single loop (send, recv) was calculated.

5.2 Test Results and Interpretation

Based on the testing performed, we present the results in Table1 and 2. All numbers are in microseconds.

Process Creation Testing Results

The average fork test with kernel 2.4.17 and the DSM module was completed in 167 microseconds, compared to 169 microseconds with kernel 2.4.17 without the DSM module. As a result, we have a 1.2% increase as overhead. This is because the system had to perform a permission check on the fork operation and to spend

	Linux 2.4.17	Linux 2.4.17 with DSM	Overhead %
Fork	167	169	+1.20%
UDP Local Access (Send Message)	16.388	19.7	+20%
UDP Remote Access (Loopback)	133.44	173.88	+30%

Table 1: Performance Analysis with IP packet modification

	Linux 2.4.17	Linux 2.4.17 with DSM	Overhead %
UDP Local Access (Send Message)	16.388	17.084	+4.2%
UDP Remote Access (Loopback)	133.44	140.64	+5.4%

Table 2: Performance Analysis without IP packet modification

some extra time on labeling of the child process.

`UDP Local Access Testing Results`

In this case, the average overhead for the setting with DSM module against the setting without the DSM module is 20%. This overhead consists of performing permission check on the socket send message and `sk_buff` label attachment for each message sent plus the labeling of IP messages. When the IP packet modification is disabled (Table 2) the overhead drops to 4.2%. As we can see most of the overhead is related to IP packet modification. Only a small fraction of the overhead is caused by the security module.

`UDP Remote Access Testing Results`

In this case, the average overhead for the setting with DSM module against the setting without the DSM module is 30%. The overhead consists of the following:

- Performing a permission check on the send socket side,
- Attaching a label to `sk_buff`,
- Attaching the security information to the IP message,
- Retrieving the security information on the receive side,
- Attaching the network security ID to `sk_buff`,
- Performing the permission checking on `sk_buff`,
- Performing the security checking on the socket, and,
- Repeating all the above operations on the return message.

When the IP packet modification is disabled (Table 2), the overhead drops to 5.4%. As we can see most of the overhead is related to IP packet modification. Only a small fraction of the overhead is caused by the security module.

5.3 Discussion

One of the most frequently asked questions is how adding security mechanisms will affect the performance of the system. Based on the testing results (Table 1), the percentage overhead for some operations, such as the UDP remote access, is considerable. The simple test, like fork, has relatively small overhead because there is only one security check. Nevertheless, some more complicated tests, like loopback, have high overhead because the security is checked in many points on the way of the traveling message. As it is shown in Table 2, the most of the overhead is added by the IP packet modification.

These results must be regarded as an upper case of the performance because no single security operation has been optimized. Nevertheless, the results demonstrated the challenges facing the development of efficient distributed security.

We believe that after optimizing the implementation, we will decrease the percentage overhead significantly.

6 Conclusion

6.1 Lessons learned

One of our objectives was to prototype a distributed security module for Linux clusters.

During the process, we acquired a lot of competence in the area of Linux kernel internals, which allowed us to set up the task security structure without memory allocation.

It is always important to divide complex problems into smaller parts in order to simplify the solution. In our case, we approached the problem of distributed access control in the way that we tried to answer three important questions:

1. How to perform the local access control?
2. How to perform the remote access control?
3. How to transfer the security information from one node to another in a transparent way?

While building the first prototype, we managed to crash the kernel many times. We realized that the swapper task (task 0) is not on the `for_each_task` list and has to be handled separately.

One of the lessons was that the system could not be over secured because it becomes unusable. By loading a very strict policy, we were not able to interact with the operating system up to the point where we had to reboot the system.

6.2 Final remarks

We were able to achieve our first goal of building the framework of the mandatory access control for Linux cluster. The security checks can be performed on the subjects and resources located on the same (local access) and different nodes (remote access) of the cluster.

We tested the framework with buffer overflow attacks and it proved that the current solution could guard against these types of attacks.

We continue to work implementing the new functionality in DSM for Linux clusters. In addition, we are in the process on building a benchmarking environment (Security Evaluation Lab) that is capable of testing the performance and the resistance of the system against various possible attacks such as denial of service attacks.

The distributed security module (DSM) is an integral part of the distributed security infrastructure (DSI). It relies on the services of DSI and provides access control services to DSI. The development of DSM and DSI are ongoing at full speed.

In the short term, we plan to implement interfaces between some services of DSI and DSM. One of the examples could be the interface between the security manager on a node and the DSM. This interface will be used to load a new policy and to pass a new node security node ID downloaded by the security manage to a node. In addition, we plan to introduce the mechanisms to pass alarms from DSM to SM and later to SS.

7 Future Work

We are in the early stage of the prototyping and in the first stage of building the mandatory access control for Linux clusters. Our first goal is to prototype the framework of the distributed access control to check the logic of the distributed access.

Based on the limited functionality (socket level network access), we plan to exercise the server security as a function of the received connection (traffic) from the clients with different security ID's. When a server accesses resources on the local node the access control does not know whether the access is a local access or is performed on behalf of a remote client. In this case, there must be a change of the server access rights based on the clients connected to it.

In the current implementation, the security information sent on the network is not protected; they can be sniffed and used in possible attacks. Our next objective is to securely transmit this information without any performance degradation.

The security information is attached as options to the IP packet. Because the IP protocol is relatively high level, there is a need to implement this feature on lower levels of the network stack.

One of our next steps is to investigate the relationships when a subject or a resource is outside the cluster. Since we are at an early prototype phase, the performance optimization is not done yet. Therefore, improving the performance of the secure system is the next challenge.

Finally, we plan to test our security mechanisms built into the servers of the cluster through generating different types of attacks to verify how the new security mechanisms can improve the overall system security.

Acknowledgments

Ibrahim Haddad, Ericsson Research Canada, for commenting and reviewing this paper. David Gordon, Ericsson Research Canada, for contributing to the IP options implementation and buffer overflow exercise.

References

[1] A. Chitturi "Implementing Mandatory Network Security in a Policy-Flexible System," Masters Thesis, University of Utah, June 1998.

[2] P. Loscocco, S. Smallay "Integrating Flexible Support for Security Policies into Linux Operating System" Technical Report, NSA and NAI Labs, Oct 2000.

[3] P. Loscocco, S. Smallay, P.A. Muckelbauer, R.C. Taylor, S.J. Turner, J.F. Farrell "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments" In *Proceeding of the 21st National Information Systems Security Conference*, Oct 1998.

[4] P. Loscocco, S. Smallay "Meeting Critical Security Objectives with Security Enhanced Linux" Technical Report, NSA and NAI Labs, Oct 2000.

[5] R. Spencer, P. Loscocco, S. Smallay, M. Hibler, D. Andersen, J. Lepreau "The Flask Architecture: System Support for Diverse Security Policies," NSA, SCC, University of Utah.

[6] M. Dagenais, I. Haddad, C. Levert, M. Pourzandi, M Zakrzewski "A New Architecture for Security in Carrier Class Clusters," Apr. 2002.

[7] G. Nutt *Kernel Projects for Linux*, Addison Wesley Longman, 2001.

[8] A. Rubini, J. Corbet *Linux Device Drivers*,

O'Reilly, 2001, Second Edition.

[9] M. Beck, H. Boehme, M. Dziadzka, U. Kunitz, R. Magnus, D. Verworner *Linux Kernel Internals*, Addison Wesley Longman, 1998, Second Edition

[10] D.P. Bovet, M. Cesati *Understanding the Linux Kernel*, O'Reilly, 2001, First Edition.

[11] Buffer Overflow, `http://www.insecure.org/stf/smashstack.txt`

[12] Flask Architecture, `http://www.cs.utah.edu/flux/fluke/html/flask.html`

[13] Linux Kernel, `http://www.kernel.org`

[14] Linux Kernel Module Programming, `http://metalab.unc.edu/mdw/LDP/lkmpg/mpg.html`

[15] LSM Patches to Kernel, `http://lsm.immunix.org`

[16] Network Patch (selopt), `http://www.intercode.com.au/jmorris/selopt/old/`

[17] SELinux, `http://www.nsa.org/selinux`

Glossary

ACS Access Control Service

DSI Distributed Security Architecture

DSM Distributed Security Module

LSM Linux Security Module

NSID Network Security ID

SCC Secure Communication Channel

SM Security Manager

SnID Security Node ID

SS Security Server

SSID Source Security ID

SSnID Source Security Node ID

TSID Target Security ID

TSnID Target Security Node ID