Distributed Access Control for Carrier Class Clusters

M. Pouzandi, A. Apvrille, E. Gingras, A. Medenou, D. Gordon
Open Systems Lab, Ericsson Research Canada,
8400 Décarie Blvd, Town of Mount-Royal, (QC) Canada H4P 2N2.
{Makan.Pourzandi, Axelle.Apvrille}@Ericsson.ca
egingras@meg.qc.ca, medenou@ieee.org, gordd00@dmi.usherb.ca

Abstract

The telecommunication industry traditionally uses clusters to meet its carrier-class requirements of high availability and reliability. As security has also become a major issue, a Distributed Security Infrastructure (DSI) has been initiated for carrier-class Linux clusters. DSI is a security framework which focuses on providing distributed security services and simplifying security administration.

This paper presents one of those services: distributed access control service (DisAC). This service manages access rights throughout the whole cluster with process-level granularity. Rules are configured through a unique security policy, which is propagated to each node of the cluster. DisAC enhances this policy at node level but also inter-node access control with process-level granularity.

Keywords: Security, Carrier Class Clustered Servers, Distributed Infrastructure, Linux Security Module (LSM).

1. Introduction

With the recent expansion and opening of telecommunication networks, carrier-class clusters' characteristics have evolved from high availability, reliability and performance to include new features such as cluster-wide security.

Many security solutions exist, ranging from external solutions (such as firewalls) to internal solutions (such as integrity checking software). But, unfortunately, all of them are based on a single node approach. Hence, they lack a homogeneous view of the cluster. Most of the time, administrators end up installing, patching, integrating and managing several security solutions. The increased management difficulty soon leads to decreased security as interoperability issues increase with the updates of heterogeneous basis of software.

This research targets the carrier grade applications running soft real-time applications on clusters. The fact that the source code of Linux kernel operating system is available is a major advantage developing DSI on Linux based clusters. Consequently, an open source project, named *Distributed Security Infrastructure* (DSI)[1], was initiated, so as to propose an adequate security solution for carrier-grade clustered servers. DSI is a security framework which provides applications running on clustered systems with distributed mechanisms for access control, authentication, confidentiality and integrity of communications, and auditing services with a process-level granularity. Detailed architecture of DSI has already been presented in [8].

Distributed Access Control Service (DisAC) is a core service of DSI. DisAC extends the kernel-level *Mandatory Access Control* (MAC) features for a single computer into features for a distributed environment.

DSI is based on open and standard software such as Linux Security Modules (LSM) for kernel level security mechanisms [10], CORBA [7] for inter-node communications, SSL/TLS and IPSec for communication security. Therefore, DSI can be easily extended to be used for other types of distributed environments, for instance Grid computing.

The paper is organized as follows. In section 2, we explain the goals behind developing DSI. After briefly reminding DSI's architecture, section 3, we introduce in section 4 the distributed security policy, a pre-requisite to functionalities of all security services. In section 5, we present the distributed access control service. Benchmark results of our implementation are presented in section 6. Finally, section 7 concludes with ongoing work and future plans for DSI.

2. Goals

DSI basically focuses (1) on providing coherent distributed security services across different nodes with process-level granularity and (2) on simplifying cluster's security administration.

In DisAC, those requirements have been taken into account in the following way.

First, DisAC implements the MAC paradigm over the *entire* cluster with process-level granularity. This is discussed more precisely in §5.2. This is particularly useful for sharing large clusters between several functionality for practical or economical reasons. In this case, there is a need for compartmentalization of cluster into separated logical sub-clusters with restricted/controlled connections between them. For instance, this scenario is quite useful for carrier grade clustered servers that are shared among different operators: operators share the global infrastructure of the cluster providing different services to their clients, but they do not wish to share their binaries or data with other operators.

Second, in order to reduce the security management complexity, DisAC uses a centralized configuration point: the Distributed Security Policy (DSP). The security administrator sets up the security policy on the security server, upon its validation the DSP is propagated through the whole cluster. This security policy is then automatically enforced at each node. Therefore, DSI eliminates the need for configuring individually each node of the cluster eliminating a major source of security breaches for servers: misconfiguration (c.f. §4).

Furthermore, DisAC allows administrators to simplify access control rules by setting different categories of security contexts and grouping binaries (c.f. §5.3).

3. DSI architecture

DSI targets clusters and, in doing so, introduces original contributions to their security. Some of its parts, however, such as its Access Control Service and its use of security contexts and identifiers, owe much to existing propositions, such as Security Enhanced (SE) Linux [4, 9].

One of our main initial hypothesis is that we do not have access to source code of the application, therefore DSI is a system-level tool for administrators/integrators rather than a development tool for developers. DSI enforces security policy for applications based on their binary images independently from security mechanisms implemented (or not) at source code. This approach is particularly useful for integrating third-party software to trusted environments.

3.1 DSI main components

DSI is composed of one security server (SS) and multiple security managers (SMs) - one per node (see Figure 1). The SS is the central point of management of the cluster: it gathers all alarms and warnings sent by the SMs and propagates the security policy over the cluster. Each SM is responsible to enforce security on its own node. Administrative messages between SMs and SS are sent on secure encrypted and authenticated channels, using SSL/TLS over CORBA (i.e.; Secure Communication Channel in figure 1).

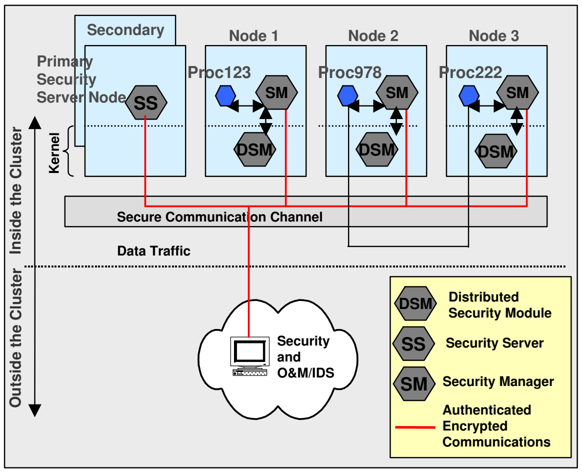

Figure 1. *Distributed Architecture of DSI*

Initially, the administrator assigns each node a *security node identifier* SnID. All processes also receive a *security context identifier* (ScID). ScIDs are global over the cluster and persistent (they do not change after rebooting the host). Actually, one should think of SsIDs more like security Group IDs (GIDs) than Process IDs (PIDs): ScIDs are meant to group together processes that have the same security context. So, contrary to PIDs, SsIDs do not uniquely identify processes but security contexts.

Hence, the distributed security policy (DSP) simply consists of a list of rules to be applied to a couple of (SnID, ScID). Through the DSP, security rules can be set for each (SnID,ScID) couple, thus enabling a fine-grained process-level security policy, valid over the whole cluster.

For security mechanisms to be effective, users should not be able to bypass them. Hence, the best place to enforce security is at kernel level. Therefore, when necessary, all security decisions are implemented at kernel level, in the so-called *DSI Security Module* (DSM). DSM is a set of kernel functions enforcing distributed security policy, and is implemented using LSM [10] as a Linux kernel module.

A more detailed presentation of DSI can be found in [8, 1].

3.2 A service based approach for DSI

DSI adopts a service based approach. The security functionality at SM is concentrated into several logical units: distributed security services (see Figure 2). DisAC enhances local and remote access control rules. The choice of a service based architecture has been motivated by various reasons:

- services can be implemented individually and separately for the rest of the system, as long as they conform to an API.

- administrator may choose to enable or disable a given security service according to specific security requirements or performance issues.

- services may be updated with newer versions that conform to the same API. In the telecommunication industry, the update procedure is particularly interesting as clusters should not be rebooted.

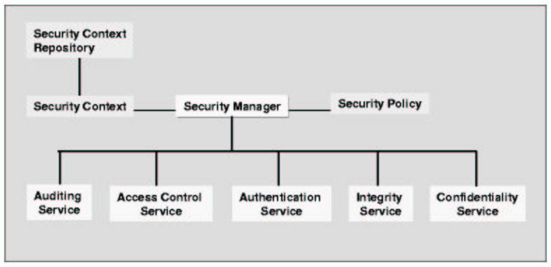

Figure 2. *DSI services.*

In DSI, all services are configured through a key element: the *distributed security policy*. The next section details the syntax for security policy and security service interactions.

4. The Distributed Security Policy (DSP)

The goal of the DSP is to define a unique, homogeneous and cluster-wide security policy to be enforced over all nodes of a cluster. It contains customization for all security services running on DSI (c.f. §3.2). As this paper focuses on DisAC, this section will concentrate only on configuration of access control rules in the DSP.

Basically, an access control rule consists in various permissions to be applied to entities (i.e.; processes, sockets,. . .) sharing the same security context and security node identifier (i.e.; a ScID/SnID couple). Permissions are organized in different *classes*. For instance, there are permissions relative to sockets (create, bind, send, receive...), others relative to process transitions etc. All kinds of permissions have not been implemented yet. Actually, we have mainly focused on network communication and process creation so far, as we believed those were the most important to our targeted environment[1].

This means that, currently, the DSP is capable of :

- assigning permissions to a given ScID/SnID couple. Wildcards may be used instead of ScID or SnID, to specify that the permission should be applied to all entities on a given node, or all entities of a given security context.

- allowing or denying a given ScID/SnID permission to spawn new processes. The DSP enables control over `fork()` or `execve()` system calls (c.f. §5.3).

- controlling specifically permissions of sockets on the cluster. ScIDs may be assigned to sockets of a given node, using a given protocol and port[2], or to processes using such sockets. Then, it is possible to set permissions between source and target sockets/processes. For instance, the following DSP sample rule allows processes with ScID=2 on node 1 to create and/or query options of sockets with ScID=2 on node 2.

```
<SOCKET_class_rule>
  <sSnID> 1 </sSnID>
  <sScID> 2 </sScID>
  <tSnID> 2 </tSnID>
  <tScID> 2 </tScID>
  <allow> CREATE
          GET_OPTIONS </allow>
</SOCKET_class_rule>
```

- more generally, controlling network permissions on the cluster, such as allowing or denying a given ScID,SnID to receive network information from a given target ScID, SnID.

In DSI, the DSP is implemented as an XML document. XML has been chosen for multiple reasons. First, XML documents are easy to read, and can consequently be easily edited by cluster administrators. Second, XML comes with a wide variety of open source tools to parse, display or even secure XML documents [2, 3]. For instance, DSI uses the Xerces parser [12] to validate the DSP against an XML Schema [13] : any syntax error in the DSP the administrator writes is immediately spotted. Third, XML's flexibility is essential in our case, so as to be able to support modification and addition of rule types that might occur through development of DSI.

5. The DIstributed Security Access Control service (DisAC)

5.1 Access control at OS kernel level

For security not to be bypassed, the DisAC service is enforced at OS kernel level (with the hypothesis that the kernel is not compromised). The Linux Security Module [10]

[1]Most carrier-grade clustered servers have many disk-less hosts and only few processors with disks.

[2]Currently only TCP and UDP are supported.

project provides a lightweight, general purpose framework for access control by implementing a set of hooks at kernel level. DSI's kernel load module is named *Distributed Security Module* (DSM), and implements different security hooks provided by LSM [8].

To each kernel entity structure (e.g., process, socket...) an ScID is added, and on each node, the DSP is loaded as a set of rules into the DSM. Then, DSM checks the access request from a process against its privileges, which are defined by its security context (ScID, c.f. §5.2) and makes a decision to grant or not the permission.

5.2 Cluster-wide access control for DisAC

Local access is based on ScID of source process (SScID) and target object (TScID) for example a TCP or UDP socket. Therefore, DisAC provides the system with a process-level granularity for access control decisions.

Furthermore, DisAC extends the local access control to a *distributed* access control for the whole cluster, using both source/target security node and security context identifiers as security information. So, access may be defined by:

```
Access = Function(SSnID, SScID, TSnID,
                  TScID)
```

At implementation level, when a source tries to access a local resource, the DSM checks locally using source and target ScIDs.

Figure 3 illustrates the case where a remote resource is being accessed. First, a local check is performed to verify that local entity (here, process 12, ScID=4) has permission to send information on local TCP socket (ScID=30, port number is fixed by the operating system). This is rule number 1 (see table 1). If permission is granted, an IP packet is sent to the remote node, containing SScID and SSnID of the source entity (here process 12, ScID=4) in its IP Options based on FIPS definition of standard security labels for information transfer [6]. When the IP packet is received on the remote node, the DSM module retrieves the IP Options and uses SScID and SSnID to check that source entity (here process 12, ScID=4) has permission to send information to the defined target entity (here TCP port 8000, ScID=31). This is rule number 2. Furthermore, a check is performed to verify that source entity has the permission to send information to the parent process (here process 14, ScID=5) which created the socket (rule 3). Finally, of course, DSM verifies that the process 14 has permission to receive information from TCP socket port 8000 (rule 4).

One has to remark that all these are presented at DSM through the permissions associated with different security contexts (SScID,...) of the entities involved (process 12, process 14, and TCP port 8000) on different nodes of the cluster.

Rule	SScID	SSnID	TScID	TSnID	Perm.
1	4	1	30	1	Send
2	4	1	31	2	Send
3	4	1	5	2	Send
4	5	2	30	2	Receive

Table 1. *Rules checked for secure remote access control.*

If there does not exist an exact match between these IDs and the existing DSP loaded in the DSM, a default behavior is used. This default behavior is defined in the DSP definition: permissive (accept the IP packet) or restrictive (drop the IP Packet).

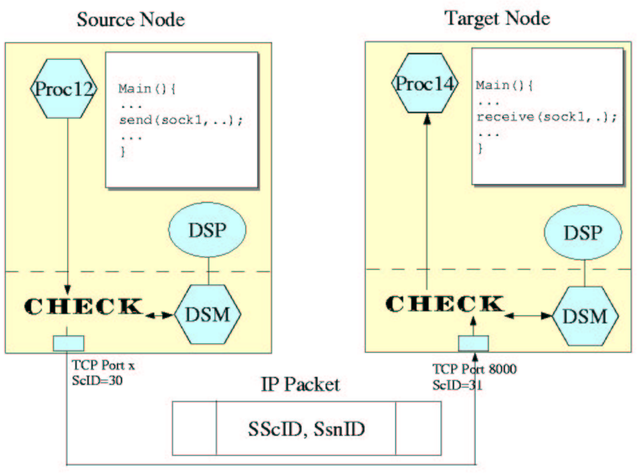

Figure 3. *Secure remote access control.*

5.3 Categorizing binaries for an easier management

In DSI, ScID are stored in ELF header of the binaries. To avoid tampering with binaries by intruders to modify the ScID, DSI plans to support digital signatures for binaries.

All processes running in DSI have an ScID, and the DSP rules explicitly under which ScID a new process may be created. This is essential for instance to detect replicating viruses. Figure 4 illustrates this procedure.

There are two different ways to create a new process: fork the process, or spawn a new process:

- If a process is forked[3], then the DisAC service checks with the DSP whether the process has such authorization or not. If authorization is granted, the forked pro-

[3]This means that the parent's binary is also used for this new process.

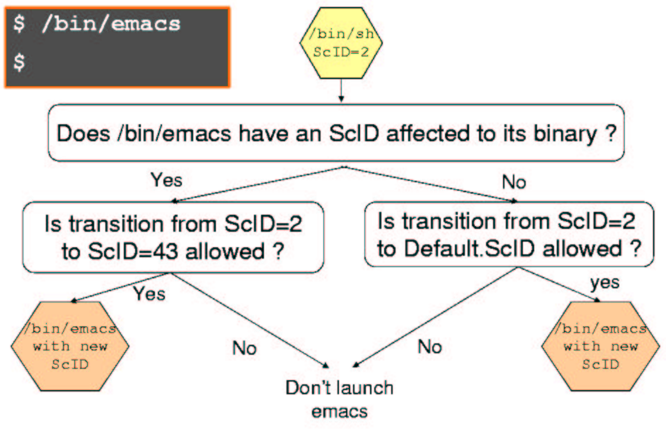
$ /bin/emacs
$
/bin/sh
ScID=2
Does /bin/emacs have an ScID affected to its binary ?
Yes
No
Is transition from ScID=2 to ScID=43 allowed ?
Is transition from ScID=2 to Default.ScID allowed ?
Yes
yes
No
No
/bin/emacs with new ScID
/bin/emacs with new ScID
Don't launch emacs

Figure 4. *Process transition: the shell `/bin/sh` tries to spawn `/bin/emacs`. The ScID 43 has been stored in the ELF header of the emacs binary.*

cess inherits everything from its father including its ScID. If the father hasn't any ScID, then the process is assigned a default (generally restrictive) ScID.

- If a new process is to be spawned (i.e.; `execve()`), then the DisAC service checks whether a binary of a given ScID, launched by a parent with a given parent ScID, is allowed to *transit* to a given *new* ScID. If such a transition is granted, the new process is launched under the new ScID. If either the parent, or the binary do not have a specific ScID, then a default ScID is used.

In §3.1, we have reminded that *security context* identifiers identify a given *security context*. Hence, if the same security context applies for different binaries, they may share the same ScIDs. This enables *compartmentalizing* of binaries simplifying security administration of the cluster.

Grouping the binaries according to their security context, the administrator will assign the same ScID to all binaries belonging to the same group. Simplifying the burden of managing security contexts for each binary to managing those contexts for several groups.

Therefore according to security needs, an administrator could choose to classify his processes from the simplest to the most complex. For example, he/she can choose to divide binaries into only two groups: trusted vs. untrusted or use very fine grained control selecting a distinct security context for each binary.

6. Benchmark results

We ran LMBench 3.0 [5] on a Linux 2.4.17 kernel running on a Intel Pentium IV 2.4 GHz to test the impact of DSI security mechanisms on the system. Tests have been performed ten times on two different configurations:

- Base: a "basic" 2.4.17 kernel with the LSM [10] patch, without any security check performed. This configuration is used as reference for comparisons,
- DSM: the same patched kernel, with the DSM module loaded, implementing different security mechanisms defined above.

Average total overhead due to DSM has then been calculated (see Table 2).

A detailed explanation of tests can be found at [5]. The `stat` test measures the time to invoke the `stat` system call on a temporary file. The open/close test measures how long it takes to open a file for reading and immediately close it.

Concerning the fork, exec and sh proc tests, they respectively measure how long it takes to fork a new process, launch a new process using `execve`, and execute a shell that spawns a new process.

In some cases (stat, sh proc), DSM improves the base. Of course, this is impossible, and we believe it only means that overhead is not significant and that benchmarks should be run on more than 10 samples.

Test type	Base	DSM	Overhead
Stat	1.92	1.94	1.0%
Open / Close	2.68	2.68	0%
Fork	92.81	93.58	0.82%
Exec	322.56	328.33	1.78%
Sh proc	2140.75	2150	0.43%
UDP	9.68	10.61	9.6%
RPC/UDP	17.66	18.7	5.9%
TCP	11.08	12.68	14.4%
RPC/TCP	23.42	24.3	3.75%

Table 2. *Comparison of performances between a LSM patched kernel without any security mechanisms implemented and a kernel supporting DSI distributed security services. Time units are microseconds.*

UDP and TCP latency tests are performed by having client and server loop on exchanging a message of 4 bytes. The RPC tests are similar, but using Sun's RPC layer over TCP or UDP.

For TCP and UDP tests, DSM's overhead ranges from 9 to 15% as security checks have to be done before processes are allowed to communicate and at the reception of each IP packet. We are currently working to reduce this overhead, and preliminary efforts show so far that if DSI security mechanisms are implemented at driver level, the overhead can be reduced till less than 4%.

For RPC tests, as the overhead due to RPC connections increases in the total time of communication, the overhead due to security checks decreases in percentage.

Furthermore, we performed performances with NetIO testing tool [11]. This tool measures the performances for already established TCP connections. Results presented in table 3 shows clearly that the overhead of DisAC for already established connections varies between the worst case 3% for short messages to an average of 1%.

Message size	Base	DSM	Overhead
1 KBytes	11497	11132	3%
2 K Bytes	11440	11281	1%
4 K Bytes	11330	11328	0%
8 K Bytes	11494	11290	2%
16 K Bytes	11438	11275	1%
32 K Bytes	11449	11331	1%

Table 3. *Comparison of performances between a LSM patched kernel without any security mechanisms implemented and a kernel supporting DSI distributed security services. Bandwidth between 2 machiens is presented according to the message size in KBytes/sec.*

Those results are quite encouraging. Moreover, one should note that all communication related tests have been performed locally on a single node; whereas DSI targets communications over a network where the overhead due to the network latency will reduce DSI's impact on performances. Work is currently under progress to optimize DSI code and measure DSI's performance in a real clustered environment.

7. Conclusion

Existing security solutions being all single-node based, they lack a homogeneous view of distributed environments such as carrier-class clusters. DSI aims at solving this lack by proposing a security framework dedicated to such systems. Our goal is to extend the Mandatory access control features for a single node to features for a distributed system.

In this paper, we have presented the Distributed Access Control service (DisAC), one of core services of DSI. We showed how DisAC implements inter-node access control mechanisms for a cluster making possible a compartmentalization of the cluster into sub-clusters.

Furthermore, to avoid the numerous vulnerabilities due to misconfiguration, DSI has focused on simplifying administrator's tasks. Two methods are detailed in this paper: first, the automatic enforcement of security rules at all nodes of the cluster upon the validation/modification of Distributed Security Policy configuration file, and second, reducing size of the security policy by assigning security contexts to groups of binaries instead of individual applications.

Benchmark results of a first prototype show the feasibility of our approach. They show minimal impact on local operations. DSI security mechanisms induce some overhead for communications as there is a price to pay for extending the fine grained security controls to the whole cluster.

We believe that our approach is general enough to be used on other kinds of distributed systems when there is a need for monitoring the access control permissions in a distributed system (e.g.; cluster, processors part of a grid,...) divided into different distributed virtual sub-systems running different applications.

We plan to focus on completing implementation of remaining LSM hooks and of course, optimizing code.

References

[1] *The Distributed Security Infrastructure Project*, `http://sourceforge.net/projects/disec`.

[2] Eastlake D., Reagle J., Solo D., *XML-Signature Syntax & Processing*, Network Working Group, RFC 3275, March 2002.

[3] Imamura T., Dillaway B., Simon E., *XML Encryption Syntax & Processing*, W3C Recommendation, December 2002.

[4] Loscocco P., Smalley S. *Integrating Flexible Support for Security Policies in the Linux Operating System*, in the Proceedings of the FREENIX track of the 2001 USENIX Annual Technical Conference, 2001, `http://www.nsa.gov/selinux`.

[5] Mc Voy L., Staelin C. *LmBench: portable tools for performance analysis*, in Proceedings of the 1996 USENIX Annual Technical Conference, `http://www.bitmover.com/lmbench`.

[6] NIST, *Standard Security Label for Information Transfer*, FIPS 188, Computer Security category, Security Labels subcategory, `http://csrc.nist.gov/publications/fips/fips188.html`

[7] *omniORB*, `http://omniorb.sourceforge.net`.

[8] Pourzandi M., Haddad I., Levert C., Zakrzewski M., Dagenais M., *A Distributed Security Infrastructure for Carrier Class Linux*, in Proceedings of the Fourth Annual Ottawa Linux Symposium , 2002.

[9] Spencer R., Smalley S., Loscocco P., Hibler M., Andersen D., Lepreau J., *The Flask Security Architecture: System Support for Diverse Security Policies*, in the Proceedings of the 1999 USENIX Security Symposium.

[10] Wright C., Cowan C., Smalley S., Morris J., Kroah-Hartmann G., *Linux Security Modules: General Security Support for the Linux Kernel*, in the Proceedings of the 2002 USENIX Security Symposium, `http://lsm.immunix.org`.

[11] *Netio, Ed Avis*, `http://ftp.leo.org/pub/comp/os/os2/leo/systools/netio116.zip`

[12] *The Xerces C++ Parser* `http://xml.apache.org/xerces-c`.

[13] *XML Schema*, W3C Recommendation, May 2001, `http://www.w3.org/XML/Schema`.