Distributed Security Infrastructure (DSI)

Copyright © http://sourceforge.net/projects/disec
REVIEW, Version 0.3

September 17, 2003

This document covers general information, technical details and basic scenarios to install and use Distributed Security Infrastructure. It also discusses the internals of the Distributed Security Infrastructure, the Distributed Security Policy, and the Distributed Confidentiality and Integrity mechanisms.

Legal Notice

All source code in this document is placed under the GNU General Public License.

Preface

The telecommunication industry's interest in clustering comes from the fact that clusters address carrier-class characteristics such as guaranteed service availability, reliability and scaled performance, using cost-effective hardware and software. These carrier-class characteristics have evolved with time to include requirements for advanced levels of security. However, only few efforts currently exist to build a coherent secure framework over clustered systems. Most of the time, clusters' administrators are left with no other choice than to integrate several partial solutions. Consequently, they often end up struggling with solutions' management, interoperability and numerous patches.

To solve this problem, Ericsson Research has decided to launch an Open Source project, named *Distributed Security Infrastructure* (DSI). DSI focuses on distributed security frameworks for real-time distributed software applications that run on large-scale carrier-class Linux clusters. This work is hosted as an Open Source project at SourceForge (`http://sourceforge.net/projects/disec`).

In this documentation, we present the rationale behind developing a new architecture named the Distributed Security Infrastructure (DSI). We describe the main elements of this architecture and discuss our preliminary results.

DSI supports different security mechanisms to address the needs of telecom application servers running on Linux clusters. DSI provides distributed mechanisms for access control services, authentication services, integrity of communications and auditing services.

Conventions used in this Book

The following is a list of the typographical conventions used in this book.

`type font` refers to source code or command line input.

Throughout the text there are references to pieces of code (for example the boxed margin note adjacent to this text). These are given in case you wish to look at the source code itself.

See `foo()` in `foo/bar.c`

Developers

- Makan Pourzandi (Ericsson Research Canada)
- Axelle Apvrille (Ericsson Research Canada)

- David Gordon (Sherbrooke University)
- Alain Patrick Medenou (Polytechnique de Montreal)
- Philippe Conan (Polytechnique de Montreal)

Past contributors

- Prof. Michel Dagenais (Polytechnique de Montreal)
- Eric Gingras (UQAM)
- Gabriel Ioan Ivascu (Polytechnique de Montreal)
- Charles Levert (Ericsson Research Canada)
- Jean-Guillaume Paradis (Sherbrooke University)
- Dominic Pellerin (Sherbrooke University)
- Vincent Roy (Sherbrooke University)
- Sam Yan (Sherbrooke University)
- Miroslaw Zakrzewski (Ericsson Research Canada)

Contents

Preface iii

1 Introduction 1

1.1 The Need for a New Approach . . . 1
1.2 Requirements for DSI . . . 2
1.2.1 Generic carrier-class requirements . . . 2
1.2.2 Attack resistance requirements . . . 3
1.2.3 Other requirements . . . 3

2 Architecture 5

2.1 A distributed architecture . . . 5
2.1.1 Overview of components . . . 5
2.1.2 Security Server (SS) . . . 6
2.1.3 Security Manager (SM) . . . 7
2.1.4 Secure Communication Channel (SCC) . . . 8
2.2 A service based approach . . . 9

3 Installation 11

3.1 Requirements . . . 11
3.2 Linux Security Module kernel patch . . . 11
3.2.1 Applying the LSM patch and recompiling the kernel . . . 11
3.3 OpenSSL . . . 13
3.3.1 Installing OpenSSL . . . 13
3.3.2 Creating certificates for DSI . . . 13
3.4 CORBA . . . 13
3.4.1 Compiling OmniORB and OmniEvents . . . 14
3.4.2 OmniORB config file . . . 15
3.4.3 OmniORB troubleshooting . . . 15
3.5 XML . . . 15
3.6 DSI . . . 16
3.6.1 Installing DSI . . . 16
3.6.2 Advanced configuration of DSI . . . 16

3.6.3 Sending commands directly to DSM module . . . 17
3.6.4 Troubleshooting DSI Installation . . . 17
3.6.5 Troubleshooting DSM . . . 18

4 Setting up a sample scenario for DSI **21**
4.1 A lightweight scenario setup . . . 21
4.2 A more realistic scenario: DisCI over UDP . . . 23
4.2.1 IP addresses . . . 24
4.2.2 CORBA setup . . . 24
4.2.3 DSP setup . . . 25
4.2.4 Load DSM . . . 26
4.2.5 Start the security servers and managers . . . 26
4.2.6 Start your UDP applications . . . 27
4.2.7 Changing the DSP through the cluster . . . 28
4.3 Other scenarios . . . 28

5 Distributed Security Module **29**
5.1 Kernel Socket Functions . . . 29
5.2 Socket Permissions and Alarms . . . 29
5.3 Allocation of Security Structure in Kernel Memory . . . 29
5.4 Speeding up policy rule matches . . . 30
5.5 Restrictive/Permissive rule enforcement in DSM . . . 30
5.5.1 Global recommendation . . . 31
5.6 Structure locking in DSM . . . 32
5.7 New System Call: sys_security . . . 32
5.8 Format of rules in DSM . . . 33

6 Distributed Access Control service (DisAC) **35**
6.1 Introduction to Mandatory Access Control vs Discretionary Access Control . . . 35
6.2 Access control for processes using ScIDs . . . 35
6.2.1 Cluster-wide access control for DisAC . . . 36
6.2.2 Categorizing binaries for an easier management . . . 36
6.2.3 Access control at kernel level . . . 37

7 Policy configuration file **39**
7.1 Distributed Policy File . . . 39
7.1.1 Security rules . . . 39
7.1.2 DSP structure . . . 43
7.2 Parsing of the DSP . . . 44
7.2.1 Parser specification . . . 44

7.2.2 Parsing the security rules . . . 45
7.3 Updating the policy . . . 46
7.3.1 Modifying the DSP . . . 46
7.3.2 Loading the DSP . . . 48

8 Secure Communication Channels 49
8.1 Events . . . 49
8.1.1 General format of events: XML . . . 49
8.1.2 XML Namespace . . . 49
8.1.3 Heart beat event . . . 50
8.1.4 Update policy event . . . 50
8.1.5 The DSM Rule event . . . 50
8.1.6 Alarm and Warning events . . . 51
8.2 Parsing dispatched XML events . . . 52
8.3 Integration of SSL in secure channels . . . 52

9 Distributed Confidentiality and Integrity service (DisCI) 53
9.1 Introduction . . . 53
9.2 The rationale . . . 53
9.3 Advantages/disadvantages of using DisCI . . . 53
9.4 How to use DisCI? . . . 54
9.5 Destination IP address modification . . . 54
9.5.1 Modification during connection establishment . . . 54
9.5.2 Modification without connection establishment . . . 54
9.6 Source IP address modification . . . 55
9.6.1 Modification with connection establishment . . . 55
9.6.2 Modification without connection establishment . . . 55
9.6.3 UDP IP transition during an active connection . . . 55
9.6.4 TCP IP transition difficulties . . . 56
9.7 IP Options Modification . . . 57
9.8 Digital Signatures . . . 58
9.9 Open Questions . . . 58
9.9.1 User Mode Linux . . . 58
9.9.2 Freeswan1.96 . . . 59
9.10 DisCI Conclusion . . . 59

10 Integrity service 61
10.1 Introduction . . . 61
10.2 Levels of digital signature verification . . . 61

11 Tools **63**

11.1 DciInit . . . 63
11.2 UpdatePolicy . . . 63
11.3 dsiUpdatePolicy . . . 63
11.4 ChangeProcSID . . . 64
11.5 SetSID . . . 64
11.6 SS_Console . . . 64
11.7 SetNodeID . . . 64
11.8 ls_dsi . . . 64
11.9 ps_dsi . . . 65
11.10PrintPolicy . . . 66

12 Testing DSI **67**

12.1 Client Server Test Programs . . . 67
12.2 DSM filesystem testing . . . 67
12.3 DSM unit testing . . . 68
12.4 DSM automatic scenario testing . . . 68
12.5 DisCI functionality tests . . . 69
12.5.1 Context . . . 69
12.5.2 Steps to Verify DisCI Functionality . . . 70
12.5.3 Integration Tests . . . 72

13 Debugging DSM **73**

13.1 dsi_debug.h . . . 73
13.2 Buffering and printk . . . 73

14 Benchmarking DSI **75**

14.1 LMBench results . . . 75
14.2 DisCI Benchmarks . . . 76
14.2.1 Dgram . . . 76
14.2.2 Dgramresp . . . 77

References **79**

Glossary **81**

Bibliography **82**

Chapter 1

Introduction

In this chapter, we discuss different characteristics of DSI and the necessity of using a new approach to security.

1.1 The Need for a New Approach

Numerous domains make use of services with high availability, reliability and scalability. For instance, on a practical point of view for telecoms, having emergency lines (9-1-1) out of service - even temporarily - would probably seem absolutely unacceptable to everyone.

To ensure high availability for such applications, the telecommunication industry uses *clusters*, i.e. a group of loosely coupled machines. However, carrier-class characteristics have evolved and now require additional requirements such as advanced levels of security.

To do so, many security solutions exist ranging from external solutions, such as firewalls to internal solutions such as integrity checking software.

Unfortunately, there are no dedicated solutions to clusters: all of them are based on a single node approach, and consequently lack a homogeneous view of the cluster. For instance, on clusters, this raises the following problems:

- each node has to set its permissions individually, but coherently to other machines. If a single node offers wider rights, or is not up-to-date, this may result in a security breach for the whole cluster.
- telecom clusters usually running the same application for a long period without interruption, under a same username, there is virtually no authentication of tasks. For instance, at Figure 1.1, process a wishes to access resources on node 2. It therefore makes a remote access to node 2's security manager (process b). If process b is allowed to access the resource, then access is granted: access control is granted according to process b but *not* according to process a. This is not coherent.

Most of the time, to cope with this homogeneity issue, administrators have no other choice than to package, integrate, patch and manage together several existing security solutions. This leads to an increased difficulty of management, and very often to a decrease in security as interoperability issues are raised and errors occur.

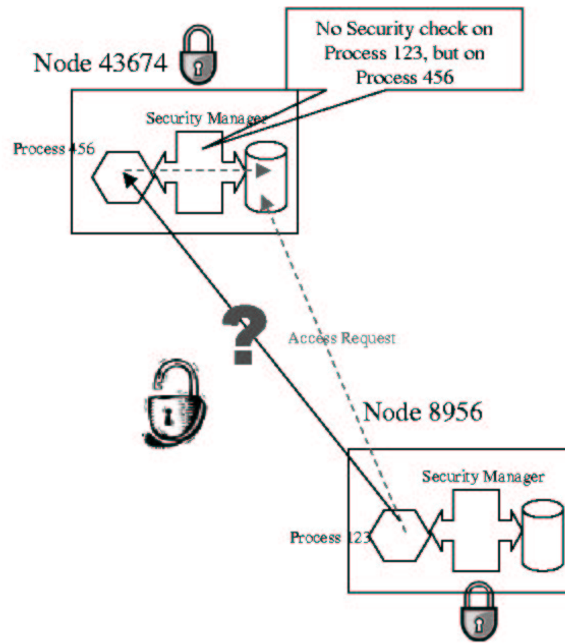

Figure 1.1: Current access control approach on clusters

Moreover, carrier-class clusters have tight restrictions on performance and response time, and in fact, many security solutions cannot be used due to their high resource consumption.

DSI tries to solve those problems by proposing a homogeneous security infrastructure for large-scale clusters running real-time distributed carrier-grade applications. As Linux provides a free reliable basis for telecom servers many on going projects have started using it [12, 16], and DSI too, has chosen to focus first on Linux.

1.2 Requirements for DSI

1.2.1 Generic carrier-class requirements

As part of a carrier-class cluster, DSI should comply with carrier-class requirements such as:

- a five nines availability (99.999%): clusters should operate non-stop, regardless of hardware or software errors. Operators should be able to perform upgrades and maintenance tasks without disturbing running applications.
- reliability
- flexibility is a major requirement for all security solutions adopted by Linux; security has to do with both technological and legal issues regarding privacy. For example, the approach to privacy issues is different in Europe and US, not to mention elsewhere in the world. Therefore, the solution should be flexible enough to allow its adaptation to different technologies (algorithms, key sizes, protocols . . .) but also legal environment.

- quality of protection: it should be possible to provide different clients simultaneously with different quality of protection. For example, some clients want to pay for a good quality of protection according to the type of transactions they handle through their terminals. It is to anticipate that the majority of clients do not want to pay an extra amount of money for the security. The system must be able to provide simultaneously different clients with different quality of protection.

1.2.2 Attack resistance requirements

- access to resources (sockets, inodes ...) should be granted according to each process. This is a fine-grained approach based on each individual process.
- authorization to spawn new processes should be controlled, and of course, new process should not be able to override security permissions that have been assigned to him.
- processes (from the same or different nodes) should be able to communicate on secure channels throughout the cluster, according to their needs (confidentiality, integrity ...).
- if malicious code were to run on a node of the cluster, 1/ it should be detected as fast as possible before it consumes too many resources and 2/ DSI administrator should be able to enforce immediately a new security policy cutting the malicious code off all its resources.

1.2.3 Other requirements

- Maximum performance: carrier grade industry is tied by very tight demands on response time and high availability. Any security solution that cannot satisfy those demands or in a way forbids the system to answer in a timely manner is not de facto an acceptable solution to the carrier grade industry. This is very different from some fields such as the military where security is a top priority and the client accepts easily long response times. Knowing that, one big leader in security field is military and alike businesses; many existing security solutions cannot be used as it is by carrier grade industry. The introduction of security features must not impose high performance penalties. Performance can be expected to degrade slightly during the first establishment of a security context; however, the impact on subsequent accesses must be negligible. As a reasonable range, the performance degradation should not exceed 10% at connection setup time, and should not exceed 5% during the data exchange phase.
- Dynamic security policy: it should be possible to support runtime changes in the distributed security policy. As carrier class servers nodes must provide continuous and long-term availability, it is thus impossible to interrupt services to enforce a new security policy. Pre-emptive security (i.e. the capability to reflect changes in security contexts immediately) should help achieve that.
- Coherent framework: security policy coherently enforced on all nodes of the cluster. This means that all security services should fit together to prevent any

exploitable security gap, and that they should all have a global understanding of the whole cluster and not only their node as an independant machine.

- Ease of configuration and use: when security is well understood and easy to configure, administrators get a better chance to efficiently secure their system. For instance, cluster's security should be managable globally, and should not require node per node administration.

Chapter 2

Architecture

In this chapter, we explain how DSI works, the role of each component and its interaction with other components.

2.1 A distributed architecture

2.1.1 Overview of components

DSI proposes a distributed security model meant to fit distributed environments such as clusters. Security components are distributed onto all nodes of the cluster. Basically, the architecture is made of:

- a security server (SS), which is the central point of management in DSI. It is the entry point for secure management and information such as alarms or messages coming from 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 cluster's security policy (also called the *distributed security policy*), and broadcasts any changes to all cluster nodes. To avoid a single point of failure, best use would be to have two security servers (a primary and a secondary) running the SS on equally hardened nodes, without any other services.

- multiple security managers (SM) enforcing security in each node of the cluster. Each security managers owns a local copy of the cluster's security policy and is responsible for enforcing this policy (and its changes) in the security environment of the node.

- and a secure communication channel (SCC), which provides an encrypted and authenticated communication 1) between security server and managers, and 2) between the security server and the "outside" of the cluster. To avoid rogue SMs introduced into the system by hackers, security managers only exchange security information with the security server.

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 persistant (they do not change after rebooting the host). Actually, one

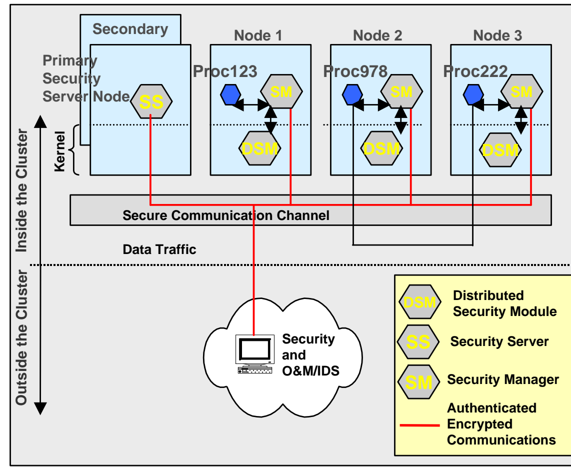

Figure 2.1: *Distributed Architecture of DSI*

should think of SIDs more like security GIDs than PIDs: SIDs are meant to group together processes that have the same security context. So, contrary to PIDs, SIDs do not uniquely identify processes but security contexts. Hence, the distributed security policy simply consists of a list of rules to be applied to a couple of (SnID, ScID).

It basically states whether such or such group of processes have such or such permissions.

Of course, 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 hooks enforcing distributed security policy, and is implemented using LSM [10] as a Linux kernel module.

For instance (see Figure 2.2) when process P tries to bind a socket on port 8800, he first calls `bind()` at user level. In turn, `bind()` uses system call `socket_bind()` at kernel level. Finally, `socket_bind()` calls specific DSM code that checks, according to the distributed security policy, whether process P is allowed to bind a socket on port 8800.

2.1.2 Security Server (SS)

The security server is the reference for all security managers. It is the entry point for DSI administrators. Its primary tasks include:

- triggering alarms and warnings inside and outside the cluster,
- initiating the SCC to propagate security related information such as distributed security policy updates, node security status, alarms and warnings. That information is sent using an event driven approach.

For instance, the security server has the authority to declare a node is compromised, and push to all other nodes the new security policy that enforces this update.

The security server stores:

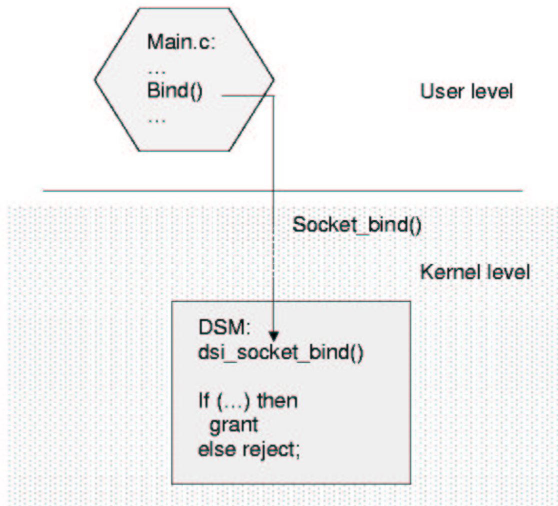

Figure 2.2: *DSI at user and kernel levels.*

- the current distributed security policy (DSP),
- a private key, a public key and public key certificate to be used on the SCC. Those keys and certificates should be generated using any suitable products[1], and administrator should ensure the private key is kept safely on the host.

2.1.3 Security Manager (SM)

The security manager is the component that enforces security on the node.

Before loading a security manager, the administrator should:

1. generate and store (securely) node'd private and public keys
2. get a public key certificate from a certificate authority. If the security server acts as a certificate authority, then certificates may be retrieved from the security server. However this is not required at all, and security managers may use any certificate authority as long as the security server trusts this authority.
3. assign a security node identifier (SnID) which identifies it uniquely for communication with the security server. Currently, each node's SnID is set manually by the cluster's administrator using the `SetNodeID` tool[2].

Then, the security manager connects to the SCC using its private and public keys. The fact that it is able to connect onto the SCC virtually makes the node join the cluster the security server manages. Of course, to make sure malicious security managers are not able to join the cluster, security server should require authentication of managers on the SCC (and same, the SMs should require authentication of the SS).

Once connected, the security manager:

[1]Note that there is no need for a certification authority on the security server. Security server's certificate can be issued by a larger certification authority such as VeriSign, Thawte, Baltimore, CertiSign . . .

[2]This should probably change in the future

- receives updates to the distributed security policy whenever there's a change[3], and is responsible to make the appropriate modifications to its own local copy of the policy.

- publish any change to security contexts of its local entities involved with remote entities.

- subscribe to changes in security contexts of remote, related entities.

2.1.4 Secure Communication Channel (SCC)

Please refer to chapter 8 for more details.

The different channels

More precisely, there are multiple different secure communication channels (see see Figure 2.3), which globally compose what is referred to as the *SCC*:

1. an alarm channel, which is used to broke alarms. SS sends alarms to SMs, and SMs send alarms to SS.

2. a warning channel (working the same way).

3. a management channel, also called the *"Service"* channel, which is used to broke management information[4].

4. a DSP channel, reserved to broke updates of the security policy and different security policy rules.

5. an Outside channel (named "SecureOM") to be used for secure communications with external clusters. Currently, no information is broked by this channel, however in the future, this could include communication with an external IDS system for instance.

Security channels are created (and deleted) by the Security Server (§2.1.2), whereas Security Managers (§2.1.3) subscribe to each one of these channels upon their creation. [5]

Channels' features

- SCC are based an event driven logic: each channel handles one specific kind of event (UpdatePolicy, Alarm, Warning...). The benefits of this approach are that (1) it does not present a single point of failure, and (2) it gives the possibility of event filtering, therefore less bandwidth is used and less time is needed for treating irrelevant information before discarding it. Events are handled by an implementation of OMG Event Services[15] (see figure 2.4).

[3]SM should also be able to "pull" the current DSP from the SS if necessary. For instance, if the SM loses the DSP, he should not have to wait for next update to get one. However, this "pull" mechanism is not implemented yet.

[4]Currently, this channel is solely used for the SS to send periodically "heart beats" to the SMs.

[5]For the time being, there is no while loop in order to avoid synchronization problems. Therefore, the security server must be created before the security managers.

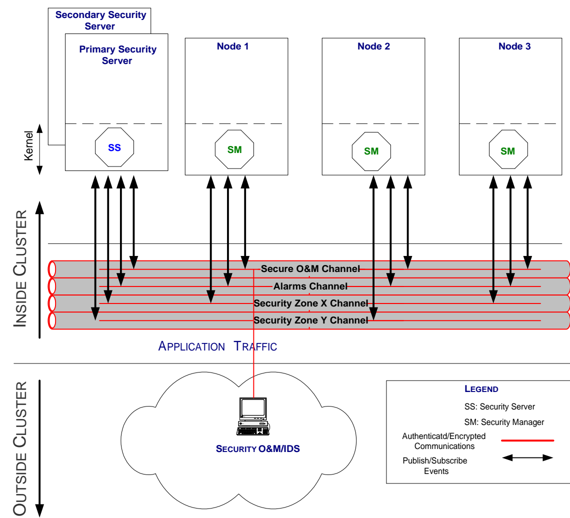

Figure 2.3: *The various secure communication channels.*

- SCC support priority queuing: if an alarm is very important and should be pushed as fast as possible to the SS (corresponding for instance to an intrusion), then the alarm event should be sent with the higher alarm priority.
- All communication channels provide authenticated (on both sides) and encrypted communications among security components. To achieve this, channels are built on top of SSL/TLS (see figure 2.4).

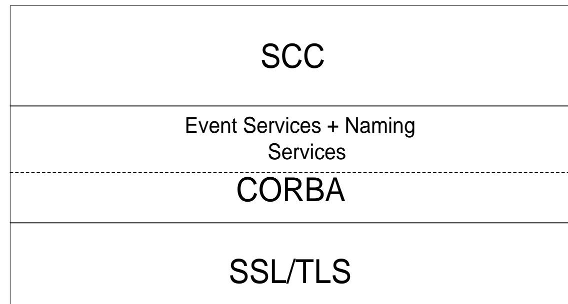

Figure 2.4: *Layered approach of SCC: SCCs are independent of lower level communication mechanisms, which increase their portability.*

2.2 A service based approach

In terms of security, DSI is meant to provide the cluster with the following services (Figure 2.5):

- an Access Control Service, that will make sure various entities (local, remote or external nodes) do not have access to unauthorized ressources. On the cluster, access to ressources is controlled by the distributed security policy,

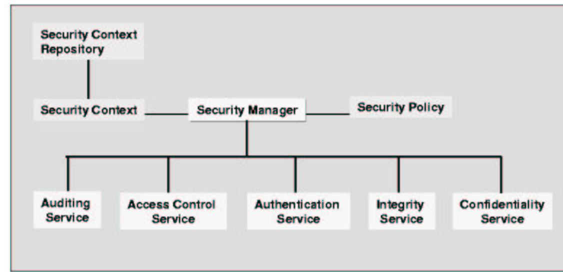

Figure 2.5: DSI Services

which basically grants or refuses access to a given ressource for such and such entity. See chapter and [?] for more information.

- an Authentication Service that should offer process-level authentication throughout the cluster. *This service has not been implemented yet.*
- an Auditing Service that would have the capability of recognizing typical attack scenarios (and possibly setting up a response strategy) from various alarms and warnings received by the security server. Design of such a service is currently under progress (no implementation yet).
- an Integrity Service, responsible for the cluster's integrity. Actually, this service is divided in two components: (1) the integrity part of the DisCI component which is responsible for the integrity of *communications* between cluster nodes, and (2) a component checking integrity of resources. (this part has not been developped yet - see §10).
- a Confidentiality Service, responsible for cluster's confidentiality. Same as the integrity service, there are actually two levels of confidentiality. The former concerns confidentiality of communications (and is implemented as part of DisCI). The latter concerns confidentiality of resource on a node, a topic which hasn't been addressed to yet.
- a (Distributed) Security Policy Management module, responsible for reading, writing and updating the distributed security policy.

Chapter 3

Installation

In this chapter, we discuss the installation of DSI components.

3.1 Requirements

DSI requires:

- the LSM kernel patch (see §3.2) in all cases.
- Xerces-C (see §3.5) in all cases.
- OpenSSL (see §3.3) if you require SSL support for CORBA channels (RECOMMENDED).
- OmniORB and OmniEvents (see §3.4), to make the SS and SMs communicate with each other. However, note that **for a quick install, you can test DSI without installing OmniORB and OmniEvents**.
- IPSec, if you want integrity and/or confidentiality between various processes of the cluster (see DisCI component at chapter 10.1).
- gcc-2.95 or gcc-2.96.

3.2 Linux Security Module kernel patch

First of all, currently, DSI will only work with a 2.4.17 kernel from `kernel.org`. Work is currently under progress with 2.4.20 and 2.5.66 kernels.

On top of a 2.4.x kernel, DSI uses LSM hooks for kernel level access control. LSM can be downloaded as a kernel patch from LSM's web site:

Kernel	Requirements	Download Web Site
2.4.17	patch-2.4.17-lsm2.gz	http://lsm.immunix.org/lsm_download.html

3.2.1 Applying the LSM patch and recompiling the kernel

To apply the patch, you should:

```
$ cd /usr/src/linux-2.4.17
$ zcat /patchdir/patchfile | patch -p1
```

Once the patch is applied, you should re-compile the kernel:

```
$ cd /usr/src/linux-2.4.17
$ cp .config .config.old
$ make config (or menuconfig, or oldconfig, or xconfig)
$ make dep
$ make bzImage
$ make modules
$ make modules_install
$ make install
```

The following kernel options should be enabled:

Section title	Option	Choice	Reason
Code maturity level options	Prompt for development and/or incomplete code/drivers	YES	to activate some experimental options
Loadable module support	Set version information on all module symbols	NO	Avoids versioning problems. Actually, you can say YES but this will complicate module building
Networking options	Network packet filtering	YES	Enables the netfiltering hooks for IP packet modification
	Kernel HTTPD acceleration (EXPERIMENTAL)	MODULAR	Includes the `tcp_sync_mss` in the kernel
Security options	Capabilities support	MODULAR	
	IP Networking support	NO	
	NSA SELinux Support	MODULAR	
	NSA SELinux Development Module	YES	
	NSA SELinux MLS policy (EXPERIMENTAL)	NO	
	LSM port of Openwall (EXPERIMENTAL)	NO	
	Domain and type enforcement (EXPERIMENTAL)	NO	

A few remarks:

- it is usually a good idea to create a symbolic link `/usr/src/linux` that links to your current linux directory.

  ```
  % ln -s /usr/src/linux-2.4.17-lsm /usr/src/linux
  ```

- Please note that the patch may not apply correctly on vendor specific kernels. We suggest you download "standard" kernels from the kernel repository `http://www.kernel.org`.

3.3 OpenSSL

3.3.1 Installing OpenSSL

OpenSSL is required to compile the secure communication channel (SCC) using CORBA with ssl features. We only tested SCC with original OpenSSL contrib downloaded directly from Open SSL web site. Therefore, we advice you to use the orginal OpenSSL contrib, even if SCC can perfectly work with modified versions of OpenSSL.

Be careful: by default, some RedHat distributions come with openSSL already installed, check for previous install in /usr/include/openssl/ssl.h. The version may be a stripped down version of the real one, so you should install the one that is listed here.

Requirements	Download web site
Latest version of OpenSSL[1]	`www.openssl.org`

Once installed (make, make install), the package's default location is `/usr/local/ssl`.

3.3.2 Creating certificates for DSI

For secure communication, DSI requires on each node:

- a file containing a list of trusted CA certificates, in PEM format,
- a file containing the keys and the certificate of the node, in PEM format. It is important to note that the file must contain both the private key **and** the certificate.

Typically, we'd have one key and certificate file per node, and a root CA responsible for signing each of these certificates (and optionally signed by an upper level CA). However, the actual architecture of certificates and keys is up to the administrator. Obviously, the strict minimum is to have one key and certificate file for the security server, and one for the security managers.

To use DSI, you may either use the dummy files provided in `dsi_home/etc` or create your own using any tool of your choice (recommended !). DSI's installation will default to the dummy files, but this is configurable.

3.4 CORBA

Secure Communication Channel is based upon CORBA to broke information. We use Omni ORB implementation for DSI for current version of SCC. However, SCC provides a portability layer to avoid dependency on specific ORB implementation. Therefore, it is possible to switch between different ORB implementations with minimal effort. As an example, it took a developer a day of work in order to switch from Mico ORB[20] implementation to OmniORB implementation.

Requirements	Download Web Site
omniORB-4.0.0.tar.gz or later	`http://omniorb.sourceforge.net`
omniEvents_2.1.2.tar.gz	`http://sourceforge.net/projects/omnievents`

Table 3.1: *Requirements for DSI's SCC.*

3.4.1 Compiling OmniORB and OmniEvents

To build OmniORB, **gcc-2.95** (or later) and **python** are required. Usually, you will already have python in your machine (check in `/usr/local/python` for instance). If not, you may download and install a minimal python from omniORB's web site at `omniorb.sourceforge.net` (named `omnipython-i586_linux_2.0_glibc2.1.tar.gz`).

DSI also requires installation of omniEvents, a contribution implementing OMG Event Services. However, omniEvents evolving slower[2] than OmniORB, its compilation is quite tricky:

- omniEvents 2.1.2 does not compile with recent gcc-3.x: you'll need a gcc-2.9x.
- omniEvents 2.1.2 supports both omniORB 3 and omniORB 4. So, it's no more use to make changes in the source code.

You should hence follow these steps:

- download OmniORB(suggested OMNIORB_TOP=/usr/local) and decompress it(tar -xzvf omniORB-4.0.0.tar.gz).
- download OmniEvents, and decompress it in OmniORB's directory so that it is to be found in `$OMNIORB_TOP/src/contrib/omniEvents` (with OMNIORB_TOP being the root directory for OmniORB installation).
- follow the manual building process of OmniORB, i.e, go to $OMNIORB_TOP/config, and modify config.mk to match your platform.
- edit `$OMNIORB_TOP/mk/platforms/xxx.mk` and set python and openssl's path.
- make export
- go to `src/contrib/omniEvents`
- make export

Another solution (neater) is to edit the $OMNIORB_TOP/configure.ac file, add the omniEvents directories that contain a GNUmakefile, run `autoconf` and `automake`, and recompile. **Do not forget to compile OmniORB with OpenSSL support** (or your channels will not be secure !).

[2]No blame meant here !

3.4.2 OmniORB config file

You have to edit the `/etc/omniORB.cfg` file to add the 2 following lines:

```
DefaultInitRef corbaname:rir:#services
InitRef NameService=corbaname::172.1.1.1/NameService
```

Where 172.1.1.1 is the IP of the Security Server (check out §?? for more information).

3.4.3 OmniORB troubleshooting

- if you are using gcc-3.x, omniORB will compile, but not omniEvents. Best choice currently is to use a platform that has gcc-2.95 or gcc-2.96.
- omniEvents refers to omniORB 3. To compile it with omniORB 4.0, replace all "ORB3" lines with "ORB4" in source code of omniEvents.
- you might need to create a symbolic link in /usr/include from python to your own python directory.
- when you type `make export` in omniEvents' directory, if you get the following message:

  ```
  You have not told me what platform you are using. Please edit
  $TOP/config/config.mk to set the platform.
  Note that you also need to set the location of Python in the
  $TOP/mk/platforms/<platform>.mk file
  ```

  This means omniORB4 Autoconf file didn't match with the right platform, so you have to edit the 2 files mentionned above according to your platform

3.5 XML

All messages exchanged between different security agents (Security server and security managers) are written in XML (c.f. §8.1.1).

Also, Distributed Security Policy (DSP) configuration file is written in XML.

Requirements	Download Web Site
Xerces-C 2.1.0 or later	http://xml.apache.org/xerces-c/index.html

For instance, you may use xerces 2.1.0. To build xerces, you should:

```
%> create symbolic link: ln -s xerces-c-scrc-2_1_0 xerces
%> export XERCESCROOT=<full-path-to-xerces>
%> cd $XERCESCROOT/src/xercesc
%> autoconf
%> ./runConfigure -plinux -cgcc -xg++ -minmem -nsocket
  -tnative -rpthread -P <install-dir>
%> gmake
%> su root
# gmake install
```

Note: A common error is to forget the 'C' between XERCES and ROOT...

For more explanations on how to build xerces, it is a good idea to always refer to `http://xml.apache.org/xercesc`, as this will contain the most up to date information.

3.6 DSI

Download latest DSI from http://sourceforge.net/projects/disec and extract in a directory. You need root privileges to load the DSM kernel module but you do *not* need to be root to run any other component of DSI.

Please note that DSI does not compile on gcc 3.2. Use gcc 2.96 or lower gcc..

3.6.1 Installing DSI

Make sure that you have installed the required packages (see §3.1). Then, compiling DSI should be pretty easy as it follows the common "configure and make" strategy:

```
% ./configure
% make or make --with-ssl=/usr if using the system default SSL.
```

Our configure script understands the following additional arguments:

- –with-ssl=<ssl directory>
- –with-omniorb=<omniorb directory>
- –with-xerces-c=<xerces directory>
- –with-linux_src=<src linux root directory>
- –with-omni_platform=<platform> (omniORB platform, see $(OMNI_DIR)/mk/platforms to list them. Default value is i586_linux_2.0_glibc2.1)
- –prefix=<directory> (base directory for installation of DSI, see below.)

If you do not use those arguments, `configure` will try and find the location of DSI's requirements in standard location (such as `/usr/local/ssl`).

Optionally, you may install the DSI binaries and libraries on your system:

- Make sure you are currently the root user.
- Then run `make install`. DSI will install its main binaries in <prefix-dir>/bin, and its libraries in <prefix-dir>/lib. The documentation will be in <prefix-dir>/share/doc/dsi-<version>
- To uninstall what you just installed, simply type `make uninstall`.

3.6.2 Advanced configuration of DSI

Table 3.2 should help you to edit the configuration files if you want to change their default values

3.6.3 Sending commands directly to DSM module

In the following you need to have root privileges.

In order to send commands to the DSM module, you need to create a character device dedicated to DSI:

- Load DSM
- Look at /proc/devices, you should see something similar to this : Character devices:

```
  1 mem
  2 pty
  3 ttyp
...
254 DSI_module

Block devices:
  2 fd
  3 ide0
 22 ide1
```

- Note the number near DSI_module.
- Make a char device file :

```
% mknod /dev/DSI_module c major_number 0
```

- No need to reboot.

You can obtain the major number corresponding to DSI_module device from /proc/devices file.

This character device is used to interact with DSM.

Once the file has been created by mknod command, you can use any tool to write into the /dev/DSI_module device. For example, write down your command line into a file and cat the file to /dev/DSI_module.

As for 2.5.x Linux releases, the security system call will be removed. From then, this device will be used for communication with DSM module.

3.6.4 Troubleshooting DSI Installation

Cannot find Xerces library, or refuses to compile XML directory

In your Xerces library, check that you have a libxerces-c library. If not, make the appropriate symbolic link:

```
%> cd /usr/local/xerces/lib
%> ln -s libxerces-c.so.21.0 libxerces-c
```

3.6.5 Troubleshooting DSM

Makefile options

First, in the Makefile, there is an option '-O2' specified. This is not just for fanciness. This option tells the compiler to optimise the code a little, among other things to resolve inline function references. Functions like ntohs and htonl are declared inline, which means that they aren't really functions, but more like macros. For a kernel module, only the compilation phase is done in userland. The linking process will be done using kernel symbols once it is loaded in kernel memory. If the '-O2' option is not specified, the symptom will be an unresolved symbol 'nthol' for instance. However, if the option is specified, the reference will have been resolved because the function wil have been inlined and the module will load correctly.

Unresolved symbol tcp_sync_mss

Another problem often encountered is an unresolved symbol tcp_sync_mss upon loading the module in the kernel. To add this symbol to kernel memory, there are a few ways to do it. The simplest way is to compile the kernel with IPv6 or khttpd as a *module*. Yes, a module. This will cause the kernel image to include the tcp_sync_mss symbol. It's sloppy but it works. Don't forget to recompile the entire kernel, not just the modules. By the way, this solution is valid for kernel 2.4.17, other kernel versions have not been tested yet with DSM code.

Kernel versioning

Often, many problems can be resolved by disabling kernel module versioning within the kernel. Module versioning is useful for making modules dependent of the kernel version they are running on. This is accomplished by appending a number after each kernel symbol before it is resolved. This number will vary for each kernel version. The natural consequence is that a module versioned for a specific kernel can no longer be loaded into the memory of another kernel version, since the kernel symbols will no longer match.

However, kernel versioning is a double sided sword. If kernel versioning is enabled, the module must be compiled to use kernel versioning. On the other hand, if versioning is disabled, the module must call regular kernel symbols. The solution opted by DSM is to disable kernel versioning and compile the module versionless. Although DSM is not versioned, it cannot be loaded into the kernel memory of another version. This is because LSM data structures have been changed from one kernel version to another.

Variable	Default value	Use	Files to edit
DSI_ROOT_DIR	your install dir	Location of the DSI root installation directory	dsi_setup.sh MakeVars
XSD_EVENT_PATH	`DSI_ROOT_DIR/etc/EventSchema.xsd`	Location of the XML Schema for DSI Events	dsi_setup.sh
DSI_CACERT	`DSI_ROOT_DIR/etc/dsi_root_cacert.pem`	Trusted CA file	dsi_setup.sh
DSI_SS_KEYCERT	`DSI_ROOT_DIR/etc/dsi_server_keycert.pem`	Private key and certificate for the Security Server	dsi_setup.sh
DSI_SM_KEYCERT	`DSI_ROOT_DIR/etc/dsi_client_keycert.pem`	Private key and and certificate for the node's Security Manager	dsi_setup.sh
LINUX_DIR	/usr/src/linux	Root directory of kernel sources	MakeVars
XERCESCROOT	set by configure	Root directory for Xerces-C	dsi_setup.sh
XERCES_DIR	same as XERCESCROOT	same as XERCESCROOT but for MakeVars...[3]	MakeVars
OMNIDIR	set by configure	Root directory for OmniORB and omniEvents	dsi_setup.sh MakeVars
OMNINAMES_LOGDIR	/var/omninames	Log dir for omninames	dsi_setup.sh
OMNIORB_CONFIG	/etc/omniORB.cfg	OmniORB configuration file	dsi_setup.sh
OMNI_PLATFORM	i586_linux_2.0_glibc2.1	seeks the omniORB files in ./bin/OMNI_PLATFORM and ./lib/OMNI_PLATFORM	dsi_setup.sh MakeVars
COS_DIR	`OMNIDIR/src/contrib/omniEvents`	Root directory for omniEvents	MakeVars
OPEN_SSL_ROOT	set by configure	Root directory for OpenSSL	MakeVars

Table 3.2: *Location of configuration variables*

Chapter 4

Setting up a sample scenario for DSI

In this chapter, we describe how to set up a simple scenario for using DSI. This chapter does not explain how to install DSI, but rather how to *use* it.

4.1 A lightweight scenario setup

This section describes a very simple scenario for your first use of DSI. This scenario requires :

- an LSM-patched kernel, supported by DSI,
- Xerces
- DSI code

Note that installing CORBA is **not** necessary for this limited scenario.

In this scenario, we'll show basically how to set an ScID, how this affects loading of binaries, and how to fix this by setting a new DSP.

Do the following on your machine:

1. Source the `dsi_setup.sh` file.
2. Compile and load the DSM module into the kernel.

   ```
   [alice@xx]> cd /home/alice/dsi/lsm
   [alice@xx]> make
   [alice@xx]> su
   [root@xx]# ./load
   or
   [root@xx]# insmod dsm.o
   ```

   To list the modules loaded in the kernel type: `/sbin/lsmod`.
3. Compile the tools to be used:

```
[alice@xx]> cd /home/alice/dsi/user/tools
[alice@xx]> make
```

4. Run DciInit to configure your node as part of this (limited !) cluster:

```
[alice@xx]> cd /home/alice/dsi/user/tools
[alice@xx]> echo ‘‘1 172 0 0 1’’ > dci_policy.conf
[alice@xx]> ./DciInit /home/alice/dsi/user/tools/dci_policy.conf
```

5. Check you can launch executables with default ScIDs (ScID=0 means there is no binary ScID, hence the default ScID is used):

```
[alice@xx]> cd /home/alice/dsi/user/server
[alice@xx]> ./UDPServer &
[alice@xx]> ../tools/ls_dsi .
PERMISSION      USER    GROUP   BSID    FILE
-rwxr-xr-x      alice   install 0       UDPServer
```

6. Now, kill the UDP server and set it a different ScID. Unfortunately, it doesn't load any longer:

```
[alice@xx]> ../tools/SetSID UDPServer 20
Changing from SID 0 to SID 20
[alice@xx]> ../tools/ls_dsi .
PERMISSION      USER    GROUP   BSID    FILE
-rwxr-xr-x      alice   install 20       UDPServer
[alice@xx]> ./UDPServer
bash: ./UDPServer: No such file or directory
```

7. Obviously, we need to write a DSP to tell DSI to allow transitions between ScID 2 (default) and ScID 20 (ours). Write the following DSP and load it:

```
[alice@xx]> vi DSP.xml
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://sourceforge.net/projects/disec/DSP"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://sourceforge.net/projects/disec/DSP
        /home/alice/dsi/etc/DSP.xsd">
 <dsi_policy>
   <version>
    <major> 1 </major>
    <minor> 0 </minor>
    <date> 2002-02-28 </date>
   </version>
   <mode>PERMISSIVE</mode>
   <default_ScID> 2 </default_ScID>
   <securityRules>
    <class_TRANSITION_rule>
      <parent_ScID> 2 </parent_ScID>
      <SnID> 2 </SnID>
      <binary_ScID> 20 </binary_ScID>
```

```
      <new_ScID> 30 </new_ScID>
    </class_TRANSITION_rule>

   </securityRules>
 </dsi_policy>
</policy>
[alice@xx]> ../tools/UpdatePolicy /home/alice/dsi/user/server/DSP.xml
Reading /home/alice/dsi/user/server/DSP.xml
```

8. Now, the UDP server will launch without any problem. Notice that the ScID of the process is 30 (new_ScID):

```
[alice@xx]> ./UDPServer &
[alice@xx]> ../tools/ps_dsi
18280   30       axelle R       UDPServer
```

4.2 A more realistic scenario: DisCI over UDP

The goal of this chapter is to provide an example of how to use DSI, and demonstrate its DisCI and DisAC features.

In this scenario, our cluster is made of 3 different machines (see figure 4.1): a security server (IP address 172.1.1.1), and two security managers : one running a simple UDP server application (IP address 172.1.1.3), and the other running the corresponding UDP client (IP address 172.1.1.2). Source code of UDP client and server is very basic . The client sends a test UDP packet to the server every second.

See `UDPclient.c` in `user/client`

See `UDPserver.c` in `user/server`

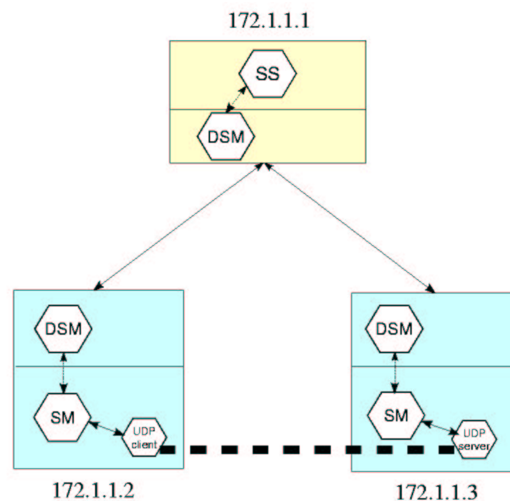

Figure 4.1: *Scenario architecture.*

Our demonstration proposes to show:

- how access control is managed over the cluster,
- how applications may communicate securely.

By now, we suppose you have x different running hosts, with DSI compiled and installed, IPSec installed, and certificates created for each host.

4.2.1 IP addresses

DisCI uses three different IP addresses, one for each security mode: no security, AH and ESP. So first, assign 3 IP addresses to each machine that will run the demo.

For instance, on the SS, you can use aliases of your running network interface:

```
[root@ss]# ifconfig eth0:1   172.1.1.1 up
[root@ss]# ifconfig eth0:2   172.1.2.1 up
[root@ss]# ifconfig eth0:3   172.1.3.1 up
```

Please notice that these addresses must be valid addresses of the machine.

Then, properly edit `<dsi_home>/user/tools/dai_policy.conf` to match each IP address with a given DisCI mode. Refer to `lsm/dsi_dci.h` for DisCI modes. For instance, the following `dai_policy.conf` file will map 172.1.1.1 to the DisCI no security mode, 172.1.2.1 to AH mode and 172.1.3.1 to ESP mode.

```
0x00000001 172 1 1 1
0x00000002 172 1 2 1
0x00000004 172 1 3 1
```

Finally, run `<dsi_home>/user/tools/DciInit` to load this policy file:

```
[alice@ss]> cd /home/alice/dsi/user/tools
[alice@ss]> ./DciInit ./dai_policy.conf
```

4.2.2 CORBA setup

On each machine, edit your OmniORB configuration file (usually `/etc/omniORB.cfg`) and add the following lines:

```
InitRef = NameService=corbaname::172.1.1.1/NameService
DefaultInitRef = corbaname:rir:#services
```

172.1.1.1 should be replaced by the IP address of your SS host. Those lines set the reference point for omniORB's naming service. Refer to [14] for further explanations.

omniNames and omniEvents both output logs in configurable directories. Those directories may be changed by setting the environment variable `OMNINAMES_LOGDIR` for omniNames, and `OMNIEVENTS_LOGDIR` for omniEvents. Refer to omniNames and omniEvents documentation for more details.

By default, DSI outputs those logs in `/var/omninames` and `/var/omniEvents`. Those directory must have rw access for the user that launches omniNames and omniEvents. For instance, you may type the following commands:

```
[alice@ss]> su
[root@ss]# mkdir /var/omninames
[root@ss]# mkdir /var/omniEvents
[root@ss]# chmod g+rw /var/omninames
[root@ss]# chmod g+rw /var/omniEvents
[root@ss]# chown alice /var/omninames
[root@ss]# chown alice /var/omniEvents
```

Then, start omniNames and omniEvents:

```
[alice@ss]> cd /home/alice/dsi
[alice@ss]> source dsi_setup.sh
[alice@ss]> omniNames -start 2809 &
[alice@ss]> omniEvents -s 7766 &
```

Note : If you have already launched omniNames before, you may probably just type in `omniNames &`. If you have already launched omniEvents before, you may probably just type in `omniEvents &`. If it doesn't work, delete omniEvents' log file in `/var/omniEvents` and re-try.

By this point, if you do a `ps` you should have multiple omniNames and omniEvents running (number of processes launched may be configured in `/etc/omniORB.cfg`. Refer to [14] for details).

```
[alice@ss]> ps
  PID TTY          TIME CMD
14365 pts/3    00:00:00 bash
 8881 pts/3    00:00:00 omniNames
 8882 pts/3    00:00:00 omniNames
 8883 pts/3    00:00:00 omniNames
 8884 pts/3    00:00:00 omniEvents
 8885 pts/3    00:00:00 omniEvents
 8886 pts/3    00:00:00 omniEvents
 8887 pts/3    00:00:00 omniEvents
 8888 pts/3    00:00:00 omniEvents
 8889 pts/3    00:00:00 omniEvents
 8890 pts/3    00:00:00 omniEvents
 8891 pts/3    00:00:00 omniEvents
 ...
```

4.2.3 DSP setup

Setup your cluster's security policy file. A sample security policy file may be found in `dsi/etc/SampleDSP.xml`.

When setting up the DSP, you should pay particular attention to the `schemaLocation` attribute. This attibute configures the location of the XML Schema for the DSP file. Usually, this file should be found in `dsi/etc/DSP.xsd`. **Edit the DSP so that the second argument of the schemaLocation reflects the absolute complete path of this XML schema.**

For instance, if you have unpacked dsi in `/opt/dsi`, then your DSP should probably start with:

```
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://sourceforge.net/projects/disec/DSP"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://sourceforge.net/projects/disec/DSP
          /opt/dsi/etc/DSP.xsd">
```

Notes:

- a space between the namespace URI (`http://sourceforge.net/projects/disec/DSP`) and its actual location (`/opt/dsi/etc/DSP.xsd`) is *mandatory*.
- you should not change the namespace URI of the DSP (`http://sourceforge.net/projects/disec/DSP`) unless you really know what you're doing.
- for more information on the DSP, please refer to chapter 7.

4.2.4 Load DSM

On each node, load DSM:

```
[alice@xx]> cd /home/alice/dsi/lsm
[alice@xx]> su
[root@xx]# ./load
or
[root@xx]# insmod dsm.o
```

When DSM is loaded, an initial version of the security policy is pushed from the Security Server (Node SS) to all processing nodes using SCC (Security Communication Channel). A security node ID is given by the Security Server to the nodes of the cluster (in this case 2 nodes).

Current limitation: SSH does not support IP options. If DSM has been compiled with IP options enabled (for DisAC features), you cannot use SSH...

4.2.5 Start the security servers and managers

Security server and managers communicate using XML events, so first, check the environment variable for the XML event schema is set on both SS and SMs:

```
> export XSD_EVENT_PATH=/home/alice/dsi/etc/EventSchema.xsd
```

Also, for SS and SMs to communicate securely, keys and certificates must be provided:

```
> export DSI_CACERT=/home/alice/dsi/etc/dsi_root_cacert.pem
> export DSI_SS_KEYCERT=/home/alice/dsi/etc/dsi_ss_keycert.pem
> export DSI_SM_KEYCERT=/home/alice/dsi/etc/dsi_sm_keycert.pem
```

To start the security server and security manager, you need a password to unlock the private key contained in DSI_SS_KEYCERT and DSI_SM_KEYCERT. You may provide this password either as an argument to the binary, or simply run it without any arguments and then you'll be prompted for the password. The latter is the recommended mode as the password is not echoed.

In this scenario, we use the dummy files provided in the dsi/etc directory. Their password is "password". On the SS node, start the security server:

```
[alice@ss]> cd /home/alice/dsi/SS/bin
[alice@ss]> ./dsiSecServer -p password &
```

On each SM node, start a security manager:

```
[alice@sm]> cd /home/alice/dsi/SM/bin
[alice@sm]> ./dsiSecManager -p password &
```

- Hack: if you get a warning about the SSL CA certificates, make sure to copy your certificates to `/etc/dsi` which is the default location for certificates[1]. If you haven't created your certificates, you may copy the sample certificates of `dsi/etc`.
- In case you get a message such as "Failure contacting events channel Factory", try either to stop omniEvents, erase its log file and restart it ; or restart the network services before you source dsi_setup.sh: `service network restart &`.

DSI is now running !

4.2.6 Start your UDP applications

First, change the ScID of the UDP server and client applications. You have to assign them an ScID for which the DSP has a correct transition rule.

Then, start the applications. If the application fails to start, you probably did not assign an ScID with the correct permissions.

For instance,

```
[alice@smserver]> cd /home/alice/dsi/user/server
[alice@smserver]> SetSID UDPServer 12
[alice@smserver]> ./UDPServer
```

And on the SM client machine:

```
[alice@smclient]> cd /home/alice/dsi/user/client
[alice@smclient]> SetSID UDPclient 12
[alice@smclient]> ./UDPclient 172.1.1.3
```

Note: There is a default value assigned to subjects and objects in the system on initialisation. So the ScID of the UDPClient and UDPServer must be different from this value to be able to test the policy.

Use the `ls_dsi` command to check the ScID of your binaries:

```
[alice@smserver]> cd /home/alice/dsi/user/server
[alice@smserver]> ../tools/ls_dsi
PERMISSION      USER    GROUP   BSID    FILE
drwrr-xr-x      alice   install -       CVS
-rwrr-xr-x      alice   install 12       UDPServer
-rwrr-xr-x      alice   install -       Makefile
-rw-r--r--      alice   install -       TCPServer.c
-rwrr-xr-x      alice   install -       UDPServer.c
-rwrr-xr-x      alice   install 0       TCPServer
```

[1] Work is under progress to use an environment variable instead.

4.2.7 Changing the DSP through the cluster

On the SS node, modify the DSP (`SampleDSP.xml` – see §4.2.3), and then notify the SS of the change:

```
[alice@ss]> cd /home/alice/dsi/SS/test/demoSecOM
[alice@ss]> ./dsiUpdatePolicy /home/alice/dsi/etc/SampleDSP.xml
```

There should be a message appearing saying that the policy has been sent. On the security server, we should see a message saying that an update policy message has been received. Then, on the security manager, we should see that the policy has been received.

When the policy is modified on the SS to give or deny the access privileges, it is propagated through the SCC to all SMs. Then, access control services reload the new policy locally, which signals the changes to the DSM module. Policy changes will be take effect immediately (pre-emptive security).

We suggest you 'play' with the XML DSP file, and change access control rules, and DisCI modes. For instance, if you switch a DisCI rule from 'no security' to ESP, you should see all communication between UDP server and client switch from IP addresses 172.1.1.2 → 172.1.1.3 to 172.1.3.2 → 172.1.3.3. And above all, if you sniff UDP packets, they should be encrypted.

4.3 Other scenarios

Other scenarios are available at [1].

Chapter 5

Distributed Security Module

In this chapter, we discuss different aspects of the DSM module.

5.1 Kernel Socket Functions

On socket level, control is implemented in all functions. Using the source ScID/SnID and the target ScID/SnID, each socket function validates the current process' permissions.

5.2 Socket Permissions and Alarms

The Access Vector List (AVL) is indexed by a hash of SScID, SSnID, TScID, TSnID, and a class. The class for sockets, DSI_CLASS_SOCKET, as well as socket permissions and alarms are defined in `dsi.h`.

See `DSI_CLASS_SOCKET` in `lsm/dsi.h`

For instance, suppose one wants to assign both SOCKET_CREATE (*0x1*)and SOCKET_CONNECT (*0x2*) permissions to a particular security context (SScID/SSnID and TScID/TSnID pair for the socket class), the correct permission would be *0x3*.

Similarly, to assign create, connect and send permissions, the correct number is *0x7*. It is simply a matter of adding individual bits to form the hexadecimal number.

5.3 Allocation of Security Structure in Kernel Memory

When patching a kernel with LSM, hooks and void pointers to structures, such as inodes, sockets, and tasks, are added at specific places in kernel source code. In this section, we are interested in the allocation of memory to the structure referenced by the void pointers.

DSM uses the void pointers to insert a security structure which contains among other things the SScID/SSnID and TScID/TSnID of the inode, or task, etc. Generally, one would allocate kernel memory with kmalloc, then assign the address to the void pointer. This is the approach taken by SELinux. DSI defines a second mechanism whereby it does not allocate memory to the structure.

For example, let's consider an inode. It will contain quite a few fields and a LSM-patched void pointer. In kernel memory, the inode is allocated a large enough amount of memory to store all its fields with quite enough memory to spare. The trick is to add our security structure (containing SScID and TScID...) to the end of this block of memory without allocating kernel memory. In other words, we are overwriting the memory and assigning a pointer directly to it.

The danger is if the kernel inode structure starts to expand, it might eventually overwrite the security structure we added at the end. On the other hand, since the security structure is only a few bytes, if the inode structure reached that point, it was probably going to overflow its bound.

This has been proven to work quite well without any glitches so far. No benchmarks have been done to see what performance gain is obtained with this approach.

See `get_task_memory` in `lsm/dsi_task.c`

The function that defines both implementations described above is get_task_memory. If END_OF_TASK_STRUCT is defined, no memory allocation will be performed. Otherwise, kmalloc will be used.

5.4 Speeding up policy rule matches

See `dsi_hash.c` in

To speed up policy search, we use a hash table. The hash function is in dsi_hash.c if you are interrested in its details.

Actually, when two policies have contradictory effect, the first to be found is the one that will be used to make a decision. If the two policy have the same 5 number (SScID, SSnID, TScID, TSnID, class) the first to be found will be the first entered in DSM. If the policy don't have the same number, they will be found in the order described in the next paragraph

The module searches first for a rule with specific argument like 1 2 3 2 1 3. The number correspond to SScID, SSnID, TScID, TSnID, class permission If it doesn't find it, it searches with wildcards (in our implementation, the wildcard number is represented by a 0) in the following order : 1 2 3 2 1 3
1 2 0 2 1 3
1 2 0 2 0 3
1 2 0 0 0 3
If no rule matches, then we consider that no rule applies to this specific case.

5.5 Restrictive/Permissive rule enforcement in DSM

This section's purpose is to inventory all checks in DSM that are permissive or restrictive. When functions in dsi_access_control.c are called, if the caller does not enforce errors, it is termed permissive. Otherwise, the caller is restrictive when negative matches or a lack of match is interpreted as a permission restriction.

When a LSM-hook function returns a negative value, it is effectively denying permission to whatever action called the hook.

```
Function                     Mode   Alarm Notes
--------                     ----   ----- -----
dsi_check_permission
```

```
-------------------
dsi_check_inode              res    no
dsi_ip_decode_options        res    no    applies only when IP options
dsi_socket_sendmsg           res    yes   dsi_socket_connect
dsi_socket_listen            res    yes   see 1
dsi_socket_sendmsg           res    yes   *** see 1!!!
dsi_socket_recvmsg           res    yes   see 1
dsi_socket_getsockname       res    yes   see 1
dsi_socket_getpeername       res    yes   see 1
dsi_socket_setsockopt        res    yes   see 1
dsi_socket_getsockopt        res    yes   see 1
dsi_socket_shutdown          res    yes   see 1
dsi_sock_rcv_skb             res    yes   see 3
dsi_check_task               res    yes

dsi_check_transition
-------------------
dsi_binprm_alloc_security    res    no    allows execution of binaries
internal_task_alloc_security perm   no    assign sid from policy, else default
dsi_socket_bind              perm   no    assign sid from port, else default
```

All function hooks that are marked restrictive either return a negative value due to a failed check or, if the security struct is null, DSI_DEFAULT_PERM is returned (see 2).

5.5.1 Global recommendation

- When rc=dsi_check_permission() is negative and an alarm is sent, error code treatment is pretty much a spaghetti. Should create a function to send alarm so security check can exit as soon as rc is negative rather than navigating through the entire function hook.

- DSI_DEFAULT_PERM should be negative? It's the default value returned when the hook should deny permission and is currently set to 0.

- dsi_sock_rcv_skb: there are 3 calls to dsi_check_permission. The first one is with the permission of the receiving socket. The 2nd one is only done if the 1st one succeeded and process and socket ScIDs are different. Only one check should be made and it should remain at the socket level. IP options are set, but if a packet arrives with no IP options, defaults are assigned and checks are done anyway. This is okay.

  The 3rd call is done when there are no IP options set (CIPSO) at compile time. This call uses ssnid=wildcard. This should not be the case, instead, the default values should set at the beginning should be used. This happens to be the same thing, but it is an issue of good coding.

Currently, DSM is in restrictive mode. If it cannot find a rule or it is not specified explicitly that permission is allowed (value=1), security check will fail in all cases. However, it is necessary that deny-permissions (value=0) exist, since often all permissions are or'ed into a single vector. Therefore, I can conclude that the function

of all rules in cache is to allow permissions. To deny a permission, it is either set to zero or deleted.

5.6 Structure locking in DSM

Since DSM is a loadable kernel module that accesses many kernel structures, the issue of locking was examined. Locking prevents other processes or CPUs from modifying or deleting a structure that is currently being changed or read. There are two issues to consider: locking of kernel structures and locking of DSM structures.

As for kernel structures, it was determined that LSM hooks were strategically placed in kernel functions so that any structure passed down to a LSM hook was already locked by the calling kernel function. However, there are some potential problems to consider. It is possible that a structure with a read lock is passed to the LSM hook, which in turn will try to write in the structure. Obtaining a write lock will deadlock the kernel, whereas directly modifying the structure will cause instability. This situation does not currently seem to be an issue. Another problem is the question of deadlocks, but for now, DSM does not need to acquire any locks on kernel structures.

For DSM structures, locking would only be necessary if DSM were to support SMP. Also, we assume that DSM module is alone in accessing its internal structures. Therefore, DSM does not need to support locking of its internal structures. Perhaps, an issue to investigate would be the allocation of the security structure when assigning it to the void pointer in inode, socket, and other structures patched by LSM.

5.7 New System Call: sys_security

LSM patch to the Linux kernel source code includes an extra system call named sys_security. DSM uses this system call to implement many userland functionalities; these are mainly tools described in 11.

At the time of writing, DSM system call (sys_security) has three parameters: unsigned int id, unsigned int call, and unsigned long args. The first parameter is simply DSI_MAGIC constant . This is a minimal check to make sure that no pointer mangling happened or anything else unpredictable happened to the values of the parameters. The second parameter identifies what function is being called because sys_security implements many functionalities. Inside the system call, DSM does a switch-case on that second parameter to execute the correct functionality. The last parameter is a pointer to an array of integers, which are the values to be used by the functionality.

See `DSI_MAGIC` in `lsm/dsi.h`

To invoke the sys_security system call from a userland program, the call must first be formatted with the _syscall3 macro. The number 3 means that the system call being called has 3 parameters. A typical syscall macro syntax:

```
_syscall3(int, security,
          unsigned int, id,
          unsigned int, call,
          unsigned int *, args);
```

The format is the return type followed by the parameter except for the first pair,

which defines the return value and name of the system call. To execute the system call:

```
ret = security( id, call, (unsigned int *)args);
```

The parameters id, call, and args are all internal to the userland program and must be defined prior to making the system call.

System call sys_security will execute with full kernel privileges.

5.8 Format of rules in DSM

The security system call used to talk to DSM takes an array of unsigned integers as argument (see 5.7. Consequently, when loading security policy rules of the DSP in the DSM, we cannot use a "complex" format such as XML, nor a structure of our own: we really need to pass information through unsigned integers. This means a translation between security policy rules (as represented in the DSP) and the actual format DSM understands needs to be done. This section provides the technical details of DSM's inner syntax to represent security rules. For more information, refer to `lsm/dsi.h`, `lsm/dsi_cache.c`, `lsm/dsi_access_control.c`, `lsm/dsi_dci.c` and `XML/src/dsiXMLDSP.cpp`.

Class	Format
DSI_CLASS_PROCESS	ScID SnID x x Class Perm
DSI_CLASS_SOCKET DSI_CLASS_DCI	SScID SSnID TScID TSnID Class Perm
DSI_CLASS_NETWORK	SScID SSnID x TSnID Class Perm
DSI_CLASS_TRANSITION	ParentScID SnID BinScID NewScID Class Perm (1)
DSI_CLASS_SOCKET_INIT	ScID SnID Protocol Port Class x (2)

Table 5.1: *DSM's inner format for representing security rules. The 'x' means that the field is ignored.*

Remarks: (1) Because of a quick bugfix workaround, we actually replace the permission column by the value of NewScID.
(2) Because of a quick bugfix workaround, we actually send the ScID in the permission: Protocol SnID Port x Class Perm=ScID.

Chapter 6

Distributed Access Control service (DisAC)

6.1 Introduction to Mandatory Access Control vs Discretionary Access Control

Numerous work [4, 22] have already proved that fighting efficiently against malicious code is utterly hard on UNIX systems that only implement *Discretionary Access Control* (DAC).

As a matter of fact, with DAC, objects' permissions are set by their owner. So, as soon as an attacker manages to get hold of a (buggy) process, he gains access to *all resources the process owns*. This security flaw is the general concept of several buffer overflow exploits for instance.

The concept of *Mandatory Access Control* tries to counter this problem. Basically, in MAC, access control no longer solely depends on owner's decision but also on a variety of security-relevant information. As an example, executing a given process requires the correct 'x' Unix permissions are set, but also that the current security context allows new processes to be created. Let us imagine Alice needs to execute two programs that she owns: `./secprg` that she trusts and `./handle-with-care` the she does not trust. Common sense requires that `./secprg` should probably be allowed to spawn new processes, but surely that `./handle-with-care` shouldn't. This is what MAC does: it clearly assigns different security contexts to both programs. With DAC, this might have resulted in a security flaw.

Although the Linux community has not yet come up with a federating way to implement MAC mechanisms, several interesting projects have been working in that area on Linux [9, 4]. However, those solutions are single node based, and need to be adapted to distributed networking.

6.2 Access control for processes using ScIDs

DisAC basically focuses (1) on providing coherent distributed security services across different nodes and (2) on simplifying cluster's security administration. Those requirements have been taken account in the following way.

First, DisAC implements the *Mandatory Access Control* (MAC) paradigm over the *entire* cluster with process-level granularity. This is discussed more precisely in §6.2.1.

Second, to help security administrator in his tasks, DisAC uses DSP's propagation through the cluster. The security administrator sets up the security policy on the security server, and then, the DSP gets propagated. There is no need for him to configure individually each node of the cluster (see section 7). DisAC also allows administrators to simplify access control rules by setting different categories of security contexts. This is described in §6.2.2.

6.2.1 Cluster-wide access control for DisAC

In the MAC model, access control depends on a variety of security relevant information (contrary to *Discretionary Access Control* model where access privileges are set by object's owner). For instance, in the FLASK architecture [21], access is a function of both source and target security information (named *SIDs*):

```
Access = Function(Source SID,Target SID)
```

DisAC extends this 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:

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

So, access privileges may be defined at process-level for both local and remote nodes. For example, it is possible to define that a process of type A is only able to access resources of type B on nodes M and N of a given cluster[1].

This is particularly useful for large clusters, where there is a need for compartmentalization into distinct sub-clusters with restricted/controlled connections between sub-clusters. For instance, this scenario is quite useful for telecommunication clusters that are shared among different operators: operators share the global infrastructure of the cluster providing different services, but they certainly do not wish to share their binaries or data with other operators.

6.2.2 Categorizing binaries for an easier management

In DSI, ScID are stored in binaries. Each binary can have a ScID stored in its Elf header. DSI also has mechanisms to support digital signatures for binaries in order to avoid these ScIDs to be tampered with by the intruders[2].

In chapter 2, 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.

From a practical point of view, this may be very helpful to the administrator. Instead of assigning an ScID to every single binary of the node (a heavy burden in reality

[1] Actually, for the time being, access control's granularity for a given process type has only been implemented at *node level* yet. This means the current implementation will only allow to say process of type A may access nodes M and N, but not which process types on those nodes.

[2] This is currently developed. However, there are several open source projects already implementing this.

!), he may just group them together according to the security needs (for instance beased on trusted vs. untrusted source for the binary).

At process creation, new processes are automatically given either their own specific ScID stored in their binary image; Or if they don't have any ScID stored in their binary image, they are assigned their parent's ScID.

Briefly, 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 DSP whether the process has such authorization or not. If authorization is granted, the forked process 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 the DSP allows transition from a process of a given binary ScID, with a given parent ScID to a new ScID. If the transition is granted then the new ScId is used. If either the parent of the binary do not have a ScID, a default ScID is used.

An administrator could consequently choose to classify his processes into several different groups: trusted processes that receive a specific ScID which bind them to a particular security context, and untrusted processes which simply do not have an ScID and either inherit their parent's ScID or a default restrictive ScID depending on DSP general mode: restrictive or permissive[4].

6.2.3 Access control at kernel level

For security not to be bypassed, the DisAC service has been implemented at kernel level, as part of the *Distributed Security Module* (DSM) [**?**]. The latest access control rules are always cached locally, at kernel level, in an Access Vector List which memorizes (1) the security node and security context identifiers, (2) the type of permission (called *class*), and (3) the permission.

Currently, DSI's implementation is limited to only a few permissions and classes (see table 6.1). As a matter of fact, we have focused on demonstrating DSI could provide cluster-wide access control for socket communications. we plan to extend access control mechanisms to all other useful permissions.

[3]This means that the parent's binary is also used for this new process.
[4]These modes are not currently implemented.

Class	Permissions
DSI_CLASS_PROCESS	PROCESS_FORK
DSI_CLASS_SOCKET	SOCKET_CREATE, SOCKET_CONNECT, SOCKET_SEND, SOCKET_LISTEN, SOCKET_RECEIVE, SOCKET_SHUTDOWN, SOCKET_GET_OPTIONS, SOCKET_SET_OPTIONS, SOCKET_GET_SOCKNAME, SOCKET_GET_PEERNAME
DSI_CLASS_NETWORK	NETWORK_RECEIVE
DSI_CLASS_TRANSITION	no specific permission.

Table 6.1: Permissions and classes currently implemented for DisAC

Chapter 7

Policy configuration file

In this chapter, we describe the policy configuration syntax and it's related mechanisms. We first start by describing the configuration file called DSP. Then, we take a look at the interpretation of the DSP made by the current implementation of DSI. Finally, we explain the different use's aspects of the DSP.

7.1 Distributed Policy File

The distributed security policy (DSP) is a file in which an administratively determined security policy can be written using a variable level of granularity over access control. The main goal of the DSP is to define, in a single file, a security policy to be enforced over a whole cluster. Alternative goals are to ease the human readability of the policy and to use a syntax flexible enough to support the frequent changes in the rules format. Those changes are mainly due to the early developpement stage of DSI. To fill those needs, the DSP syntax has been defined using the XML language. On top of filling the DSP requirements, XML comes with a variety of open source tools and many security mechanisms.

The DSP is composed of:

1. a list of security rules (cf. §7.1.1),
2. and a few other global items such as version, mode etc (cf. §7.1.2).

7.1.1 Security rules

Rule types are defined to manage permissions on different system object classes, ressources or operationnal modes. Before we describe the predefined types of rules, you have to know that all rules are build around the concept of security identifiers. ScID and SnID are the two kinds of security identifiers used in the DSP. An ScID is an integer value attributed to a ressource (process, socket, ...) and identifies a *security context*. ScIDs are *persistant* (i.e after reboot, resources retain the same ScID), *global* to the whole cluster (i.e known on all nodes), and identify *uniquely* a security context (but not a process, nor a binary – binaries may be grouped onto the same ScID if they should share the same security contexts). An SnID is an integer value assigned to a node of the cluster. It identifies uniquely a given node.

Process rules

The class_PROCESS_rule rule is made to define permissions that concern processes. The process rule applies to a given ScID and SnID. Currently, the only permission available is CREATE.

```
<class_PROCESS_rule>
        <ScID>1</ScID>
        <SnID>1</SnID>
        <allow>CREATE</allow>
</class_PROCESS_rule>
```

Figure 7.1: Allowing permissions to processes

The rule in example 7.1 specifies that, on node SnID 1, processes with ScID=1 may be successfully created.

```
<class_PROCESS_rule>
        <ScID>1</ScID>
        <SnID>ALL</SnID>
        <deny>CREATE</deny>
</class_PROCESS_rule>
```

Figure 7.2: Denying permissions to processes

The rule in example 7.1.1 specifies that, on all nodes, processes with ScID=1 cannot be created. Both ID elements can either take a non-negative integer value (between 0 and 65535) or the ALL keyword (meaning all valid IDs). The process permission set contains the following elements: CREATE.

ScIDs are allocated through the binary file called to initiate a process. The ScID value is stored in the ELF header of the binary file using the "SetSID" command (see §11.5).

Socket rules

The class of rules class_SOCKET_rule gives the possibility to an allow or deny specific operations on sockets for given processes. The rules uses four IDs: two source identifiers (context and node) and two target identifiers (context and node).

```
<class_SOCKET_rule>
        <sScID>1</sScID>
        <sSnID>ALL</sSnID>
        <tScID>1</tScID>
        <tSnID>ALL</tSnID>
        <allow>CREATE CONNECT SEND LISTEN RECEIVE
                SHUTDOWN GET_OPTIONS SET_OPTIONS
                GET_SOCKNAME GET_PEERNAME</allow>
</class_SOCKET_rule>
```

Figure 7.3: Allowing permissions to access given type of sockets

In example 7.3, the rule allows processes with a ScID of 1 on all nodes to perform create, connect, send (etc) operations on sockets handled by resources of an ScID 1 on any node.

It is very important to understand that neither source nor target identifiers necessarily refer to sockets. They may very well refer to processes. For instance, suppose we have a process A on node 1, of ScID=4 that communicates with process B on node 2, ScID=5. Process A and B both use a TCP socket, port 8000, on node 2, which is assigned ScID=31. On Unix operating systems, communication towards remote socket actually create a local socket (programmer does not control that socket - the system does that). So, actually, a socket on node 1 is also created, and we suppose it has ScID=30.

Sending and receiving information between process A and B results in the following rules being checked by the DSM (see table 7.1).

Rule	SScID	SSnID	TScID	TSnID	Perm.
1	4	1	30	1	Create
2	5	2	31	2	Create
3	30	1	31	2	Listen
4	30	1	31	2	Connect
5	30	1	31	2	Send
6	30	1	31	2	Receive
7	31	2	30	1	Send
8	31	2	30	1	Receive

Table 7.1: *Socket Rules checked for secure remote access control.*

We notice here that we have socket class rules (permission send, receive) for elements that are not processes (see rule number 5 for instance).

Socket init rules

Sockets ScIDs are determined by the socket_init rules, at the moment of their creation, based on the protocol it uses and the port on which it's connected. Here's an example:

```
<class_SOCKET_INIT_rule>
        <protocol>TCP</protocol>
        <port>50001</port>
        <ScID>1</ScID>
        <SnID>1</SnID>
</class_SOCKET_INIT_rule>
```

Figure 7.4: Attribution of an ScID to a socket

In example 7.4, the ScID of 1 is allocated to TCP sockets on the port 50001 created on the node 1. Has in the other kinds of rules, the SnID can take the value of "all" which means every SnID available. Allowed values for the protocol element are "UDP", "TCP" and "RAW"[1]. The port value must be a positive integer lower than

[1]Currently, RAW sockets are not supported

65536.

Network rules

The socket class rules can be used to configure access at the transport (and higher) layer of network stack, while the network rules are used to allow or deny permissions at the network (IP) layer[2].

Note that there is no tScID for network rules, as when the network rule is checked it is impossible to know to which ScID it maps. The only possible network permissions is: NETWORK_RECEIVE.

```
<class_NETWORK_rule>
        <sScID>1</sScID>
        <sSnID>ALL</sSnID>
        <tSnID>ALL</tSnID>
        <allow>NETWORK_RECEIVE</allow>
</class_NETWORK_rule>
```

Figure 7.5: Allowing network permissions

The example 7.5 figure shows an example of network class rules. The rule in example 7.5 shows us that processes with an ScID of 1 on every node can receive IP packets from processes with an ScID of 1 on any node. Note that network class rules, has for the socket rules, can also be of a denial type.

DisCI rules

One of the main features of DSI is DisCI, allowing the administrator of a cluster to decide, at run time, to switch the intra-cluster communications from beiing secured, using IPSEC, or not. DisCI can be configured thru the "class_DisCI_rule"s. DisCI rules look a lot like network and sockets rules: permissions are defined for a pair of IDs identifying the source, and another pair identifying the destination.

```
<class_DisCI_rule>
        <sScID>1</sScID>
        <sSnID>1</sSnID>
        <tScID>1</tScID>
        <tSnID>2</tSnID>
        <allow>
                <ipsec_mode>ESP<ipsec_mode>
        </allow>
</class_DisCI_rule>
```

Figure 7.6: Allowing the encrypted mode of IPSEC

The example 7.6 demonstrates how to secure the transmissions from the processes with an ScID of 1 on the node 1, to the processes with an ScID of 1 on the node 2. Contrasting with the other classes of rules, in DisCI rules the allow element can't be

[2]Consequently, network rules are, in a sense, redundant when used for socket communications, but they are needed for other types of communications (pipes, raw ip...).

replaced by a deny element. The allow element also differs by containing an element instead of a value. Current implementation of DisCI use ipsec to protect messages, but if other security mechanisms append to be implemented, more elements could be added to the allow section of the DisCI rules. Possible values for the ipsec mode element are: ESP, AH and NO_SEC. "ipsec_mode" can't contain a set of elements has the permissions in the other class of rules, the element can contain one and only one value.

Transition rules

Finally, the transition rules define the resulting ScID of a process launch by another process. When such process creation happens, the resulting ScID is determined by the ScID and SnID of the parent process and the ScID of the newly created process binary file.

```
<class_TRANSITION_rule>
        <parent_ScID>1</parent_ScID>
        <SnID>2</SnID>
        <binary_file_ScID>3</binary_file_ScID>
        <new_ScID>4</new_SnID>
</class_TRANSITION_rule>
```

Figure 7.7: A transition rule

The example 7.7 specify that when a process with an ScID of 1 on the node 2 tries to create a process from a binary file with an ScID of 3, the new process should have an ScID of 4. Has in the other class of rules, the keyword "all" can be used in the element identifying the node (SnID).

7.1.2 DSP structure

Apart from security rules, the DSP also contains a few other elements such as a "version" element, a "mode" element and a "default_ScID" element.

The **version** element contains three elements: "major", "minor" and "date" elements. Major and minor elements contains integer values identifying the version of the DSP, while the date elements contains a XML date datatype value.

The **mode** element defines the DSI behaviour when no rules exist and a check is performed. Two available modes are defined : "PERMISSIVE" and "RESTRICTIVE"[3].

- In permissive mode, what's not explicitly defined in the DSP is allowed. This is useful to set up a configuration without breaking the actual configuration of your machines. However, handle it with care, because it may be too permissive.
- In restrictive mode what's not defined in the DSP is denied.

Currently, DSI is in restrictive mode, however a few rules are automatically launched at startup so as not to "break" down the whole configuration of your machine.

[3]Currently, those modes are ignored in the DSP, and DSI always works in a restrictive mode, with some default rules added so as not to break the whole configuration of your machine.

The element called **"default_ScIDs"** defines the ScID value given to processes whose binaries do not have an explicit ScID set. This default ScID is also assigned to processes which are already running at the time the DSM module is launched. Currently, this field is ignored and the default ScID is always set to 2.

Finally, the XML Schema of the DSP introduces a **dsi_policy** element. This allows the DSP to optionally contain other policies than just DSI's. Currently, DSI will only analyze the `dsi_policy` element. See figure 7.8.

```
<policy>
        <dsi_policy>
                <version>
                        <versionID_major> ... </versionID_major>
                        <versionID_minor> ... </versionID_minor>
                        <date> ... </date>
                </version>
                 <mode> ... </mode>
                 <default_ScID> ... </default_ScID>
                 <securityRules>
                        ...(different rules)
                </securityRules>
        </dsi_policy>
 </policy>
```

Figure 7.8: DSP file structure

For more technical or syntaxical information, please refer to the XML Schema of the DSP located in `dsi/etc/DSP.xsd`, or the DSP_modif_help.xsd file also located in `dsi/etc/`.

7.2 Parsing of the DSP

In this section, we describe how the Security Server parses the XML content of the "Distributed Security Policy" (commonly called DSP).

7.2.1 Parser specification

The Security Server is written in C++ and it uses the version 2.1.0 of Apache's Xerces-C++ validating XML parser.

The path and name of the DSP file to parse are determined as an argument of the XML UpdatePolicy event (see chapter 8) and is received from the ORB channel in `SS/src/dsiSSOMSubscriber.cpp`.

The DSP is validated against the DSP's XML schema, whose location needs to be specified in the XML DSP file using the `schemaLocation` attribute:

```
<?xml version="1.0" encoding="UTF-8"?>
<policy xmlns="http://sourceforge.net/projects/disec/DSP"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://sourceforge.net/projects/disec/DSP
```

```
        /home/lmcaxpr/dsi/etc/DSP.xsd">
<dsi_policy>
```

It is very important to leave **one space** between the DSP's namespace (`http://sourceforge.net/projects/disec/DSP`) and the **location** of its schema (here, `/home/lmcaxpr/dsi/etc/DSP.xsd`).

More information about the schemaLocation can be found from its own schema, which is found at `http://www.w3.org/2001/XMLSchema-instance`.

7.2.2 Parsing the security rules

The current implementation of the security server only considers the elements named after the six security rule classes. Those rule class names are:

- "class_PROCESS_rules",
- "class_SOCKET_rules",
- "class_NETWORK_rules",
- "class_SOCKET_INIT_rules",
- "class_DisCI_rules"
- and "class_TRANSITION_rules".

Currently, elements that are not part of the security rule tags (such as "version", "mode", "default_ScID"...) are ignored by the parser. Newever versions of DSI ought to parse the DSP completely.

The parsing of the DSP file is handled by `XML/src/dsiXMLDSP.cpp`. Building a `dsiXMLDSP` object parses the file that is passed to it as argument, and validates it against its XML Schema. Internally, a bi-dimensionnal array called `m_RawDSP` is created. This array is made of six columns and a number of row corresponding to the total number of rules read in the DSP (the total number of rules read in the DSP is determined by the private `getNbRules()` method).

The rules stored in the `m_RawDSP` array comply to the DSM rule syntax which is described at §5.8. Briefly, values can be interpreted like this:

```
sScID   sSnID   tScID   tSnID   class   permissions.
```

Here's how the translation is done for each kind of rules:

- **ALL**: the ALL joker is translated the value of 0.
- **class_PROCESS_rule**: the process class rule does not contain any target ScID or SnID. Hence, in the array, the values of tScID and tSnID are left to 0, and will be ignored by the DSM.
- **class_SOCKET_INIT_rule**: the format of this rule is totally different to the usual scheme because of the very special meaning of the rule. The first column represents the ScID. The second column represents the SnID. The third column represents the protocol of the socket (TCP, UDP, RAW), and the fourth column represents the port number.

- **class_TRANSITION_rule**: this rule follows the following format: the first column to be stored is the Parent ScID, then the SnID, then the Binary ScID, then the NewScID. The class and permission columns are ignored.

7.3 Updating the policy

In this section, we explain how we update the policy. At first, we explain how we can edit and modify the policy file, then we explain how to load that DSP in DSI.

7.3.1 Modifying the DSP

To update the DSP file, you basically have two solutions:

- edit the XML file manually, referring to the XML Schema located in `dsi/etc/DSP.xsd`, and DSP samples in `dsi/XML/test/ex*.xml`.
- or use the graphical TCL/TK DSP editor. This should be your favored choice if you are not familiar at all with XML. This tool is located in `dsi/user/tools/dsp_generator`, and is called dsp.tcl. It is a simple TCL/TK interface script for writing a DSP. To use this tool, you need the Run-time library (rtl) to be installed on your system (available at `http://www.prs.de/cgi-bin/download/download.pl`). You can launch the interface by calling the dsp.tcl file from the command line. When a dsp file is created by the dsp_generator, it assumes that the corresponding XML shema is called DSP.xsd and that it is located in the current directory.

Next we describe how to write the different class of rules using the dsp_generator, but before writting any kind of rules, you should enter the path and name of the DSP file in the "DSP file name" textbox.

Configuring a Process rule with DSP Generator

A process class rule looks like this:

```
<class_PROCESS_rule>
        <sScID>1</sScID>
        <sSnID>2</sSnID>
        <allow>CREATE</allow>
</class_PROCESS_rule>
```

To write a process class rule, one should use the "Source : ScID" field for the sScID value and the "Source : ... SnID" field for the sSnID value. Remember that the '0' value stands for the "ALL" keyword, meaning that if you choose 0 for an ID value (i.e.: Source ScID and SnID, Target ScID and SnID, and New ScID), "ALL" will be written in the corresponding field of the DSP file. To set values in those fields: select the apropriate field and use the up going and down going arrows or just type the desired value in.

Then, you select "PROCESS" in the "Object Class" combobox. Selection is done by dropping down the list (clicking on the arrow or on the text part of the widget)

and by clicking once on the chosen value. At this point, if a process rule having the same sScID and sSnID value is already existing in the DSP, the allowed and denied permissions of that rule should appear in the apropriate listbox.

Finally, you have to include the desired permissions in the "Allowed" listbox. You can also include permissions in the "Denied" listbox. The tool won't let you include the same permission in the "Allowed" list and in the "Denied" list. This is to prevent the creation of contradicting rules.

After entering all the rule data, you have to press on the "Save" button. If you didn't enter the "DSP file name" yet, an error message will be written in the parent terminal. If you did enter the file path and name, your rule(s) will be written in the selected DSP file. If no permissions are in the "Allowed" or "Denied" lists: no rules will be written. If permissions are "Allowed", a new process allow rule will be written. If permissions are "denied", a new process deny rule will be written. If some permissions are "allowed" and some are "denied", a new process allow rule and a new process deny rule will be written. If process rules with the same sScID and sSnID where present in the DSP they will be replaced (erased) by the new ones.

Configuring a Socket rule with DSP Generator

Socket class rules are generated by this interface the same way then process class rules are. One should use the "Target : ScID" field for the tScID value and the "Target : ... SnID" field for the tSnID value. Remember that the '0' value stands for the "ALL" keyword.

Configuring a Network rule with DSP Generator

Network class rules have the same format than Socket class rules, and they are configured the same way in the dsp_generator.

Configuring a DisCI rule with DSP Generator

DisCI class rules are entered in the dsp_generator the same way then socket or network rules. The only difference is that the permissions (permissions should be called modes in this case) can only be allowed, and only one permission can be granted.

Configuring a Socket Init rule with DSP Generator

To write a socket_init rule in the DSP file, one should start by selecting "SOCKET_INIT" in the "Object Class" combobox. The "Permissions" list will then be filled by the available protocols. In this circonstances, "Permissions" (i.e.: protocols) can only be added to the Allowed list and the Allowed list can only contain one value. That value will be the value inserted in the protocol element of the rule.

The port value should be written in the "Port" field of the interface. A '0' in the Port field will be scripted has "ALL" in the resulting rule.

The ScID and SnID values should be written in the "Target : ScID" and the "Target : ... SnID" fields.

If socket_init rule with the same protocol, port and SnID was present in the DSP, it would be replaced (erased) by the new one.

Configuring a Transition rule with DSP Generator

A transition class rule looks like this:

```
<class_TRANSITION_rule>
        <parent_ScID>1</parent_ScID>
        <SnID>2</SnID>
        <binary_file_ScID>3</binary_file_ScID>
        <new_ScID>4</new_SnID>
</class_TRANSITION_rule>
```

Fields from the interface are mapped to the transition rule elements this way:

```
INTERFACE              <-->            RULE
- Source : ScID                        - parent_ScID
- Source : SnID                        - SnID
- Target : ScID                        - binary_file_ScID
- New ScID                             - new_ScID
```

If transition rule with the same parent_ScID, SnID and binary_file_ScID was present in the DSP, it would be replaced (erased) by the new one.

7.3.2 Loading the DSP

To load a DSP in all security managers of a cluster, you should use the `dsiUpdatePolicy` command found in `dsi/SS/test/demoSecOm`, with the absolute path and filename of the DSP you want to load. This will send a message to the Security Server, asking him to "push" that DSP. It should then be done immediatly and the new DSP should be enforced automatically. More information about this tool is available at §11.3.

Chapter 8

Secure Communication Channels

In this chapter, we detail how the different secure channels work.

8.1 Events

Each channel handles only one given type of event.

8.1.1 General format of events: XML

Events are sent using XML format. XML is convenient in our case because:

- it produces self describing messages that are easy to understand for the user.
- it is an *extensible* language, so it is simple to make messages' format evolve according to our needs. This is particularly useful in the early stages of development of DSI where we are not quite sure about what all messages should contain, and how to format information. Once we have a fixed number of message types, if performances require it, we can decide to switch from a "dynamic" time consuming XML parser to a more static and efficient approach based on `strcmp` and switches.
- XML has its own mechanisms for security. For instance, an XML document may be signed, or encrypted. During message intensive phases of DSI, it is even possible to sign or encrypt only the critical parts of an XML document.

By the way, note it is possible to use XML to transmit commands as well as data.

8.1.2 XML Namespace

All XML Schemas [23] of DSI are written with the following namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://sourceforge.net/projects/disec"
```

```
xmlns:dsi="http://sourceforge.net/projects/disec"
elementFormDefault="qualified">
```

This defines the tag 'dsi' for types DSI introduce.

8.1.3 Heart beat event

See `dsiSecServer.cpp` in `dsi/SS/src`

Heart beat is a very simple event sent from SS to SMs to check the availability of SMs. This is to detect the faulty nodes or nodes that do not run SM.

```
<element name=''Check'' type=''dsi:Check_type''/>
<complexType name=''Check_type''>
  <choice>
    <element name=''HeartBeat'' type=''dsi:HeartBeat_type''>
  </choice>
</complexType>

<simpleType name=''HeartBeat_type''>
  <restriction base=''string''>
    <maxLength value=''0''/>
  </restriction>
</simpleType>
```

A wrapping `Check` type is created so that other types of check events can be added later on. The value of the `HeartBeat` type has no importance and should be left blank.

8.1.4 Update policy event

See `dsiUpdatePolicy.c` in `dsi/SS/test/demoSecOM`

This event is sent to force the SS to read a new DSP. .

```
<element name=''UpdatePolicy'' type=''anyURI''/>
```

In this event, the **absolute path** to the DSP to load should be set in the `UpdatePolicy` element.

8.1.5 The DSM Rule event

This is an even sent from SS to SMs, to push the updated DSP rules. This even uses DSM's internal syntax to represent rules, instead of XML.

We do not use XML here because it would mean

1. parsing the DSP on the SS,
2. sending the Update Rule event in XML,
3. parsing the event on the SM,
4. enforcing the rule in DSM.

Hence, we would basically end up parsing the DSP N+1 times (N being the number of nodes on the cluster)...

So, instead, we have chosen to:

1. parse the DSP on the SS
2. send each updated rule in a pre-formatted DSM syntax,
3. enforce the rule directly in DSM (no parsing/analyzing needed).

The syntax for this event should strictly conform to:

```
BEGIN_DSM_RULE <sScID> <sSnID> <tScID> <tSnID> <class> <perm>
                        END_DSM_RULE
```

with :

- the whole event being a plain character string, ended by a NULL. No CR or LF.
- sScID, tScID, sSnID, and tSnID being 32-bit integers ranging from 1 to 65535. Special value 0 mapping to "ALL" (joker), and value -1 meaning value is not set.
- class being an integer with valid values in `dsi.h`.
- perm being an unsigned 32-bit integer, with valid values set in `dsi.h`
- in the special case of transition class, sScID is replaced by parent's ScID, tScID by the executable's ScID and perm by the new ScID.

Note: special caution should be taken at implementing DSM across various platforms so that the event remains portable. As long as 32 bit integers, unsigned integers and basic character strings are used (there's no special character in the header and footer tag), this event should remain portable. If this becomes a major issue, work should be done to use a simple portable syntax (DER ? XML ? ..).

Note 2: the Update DSM Rule event sends one rule at a time. Actually, it is intended to send only *updated* rules. However, in practice, currently, no such mechanism is implemented, and all rules get pushed at each DSP propagation.

8.1.6 Alarm and Warning events

See `dsiSecManager.cpp` in `dsi/SM/src`

The security managers send the following alarms to the SS :

```
<element name="Information" type="dsi:Information_type"/>
<complexType name="Information_type">
  <choice>
    <element name="Debug" type="string"/>
    <element name="Info" type="string"/>
    <element name="Warning" type="string"/>
    <element name="Alarm" type="string"/>
    <element name="Critical" type="string"/>
  </choice>
</complexType>
```

Currently, only warning and alarm levels are implemented. Moreover, this event is likely to evolve once the auditing service is designed.

Remark: instead of defining specifically a type for each level, an idea would have been to have a *level* field (containing severity) and a *content* field (containing the information). We have not done this, because, then in XML it is not possible to make a particular level field match with its corresponding type (ex: level specifies 'Alarm' so only an alarm content is valid)[1]

8.2 Parsing dispatched XML events

Both the security server and the security managers instantiate an XML parser to be able read dispatched XML events on the SCC.

The XML event undergo the following process:

1. First, they are created and pushed to the corresponding event consumer (for instance, see `dsi/SS/src/dsiSecServer.cpp`.
2. The event consumer forwards the event to the specified channel (for instance, see Push in `dsi/SCC/PL/src/dsiPLEventConsumer.cpp`).
3. The push() function of the corresponding subscriber gets called (for instance, see `dsi/SM/src/dsiSMServiceSubscriber.cpp`).
4. The event is parsed with a SAX Parser[19].
5. Message's syntax is checked.
6. The corresponding command is executed, and eventually the information is propagated to other components.

Note that XML events are not authenticated as they are sent on secure channels that already support authentication and encryption.

8.3 Integration of SSL in secure channels

To protect the integrity and the confidentiality of the messages between SS and SMs, SCC uses SSL/TLS. OmniORB 4.0 supports SSL, so we use this feature to add SSL support to SCC.

Private and public keys of each node should be generated by the cluster's administrator and stored in specified directory[2]. Note private key's protection is the responsibility of the administrator.

[1]If somebody thinks this is possible, please let me know...

[2]Currently, the name of that directory is fixed - hard coded - to /etc/dsi, but an environment variable should soon be provided

Chapter 9

Distributed Confidentiality and Integrity service (DisCI)

In this chapter, we explain implemented mechanisms to change IP addresses at the socket layer, using the security hooks LSM.

9.1 Introduction

DisCI concerns the communications inside the cluster.

9.2 The rationale

DisCI is the same thing as NAT with the difference that it can be used at process level inside a cluster in a dynamic way, i.e.; the administrator according to the general security context for each process type set can switch between secure and unsecure chanels.

Even if the primary goal of DisCI is for communications inside the kernel, we have extended the implementation to the external addresses. This is to allow to change at process level the ipsec, this is further more according to our approach to extend MAC into the cluster.

For example, a usage model can be the following: a linux cluster, with several nodes running Linux. Using SetSID, we divide the binaries into several sets: using secure, unsecure channels. For example, data base related binaries, statistic related binaries... We can imagine that data base binaries use secure channels as he ones in the static sets use non secure channels. We have the possibility of switching the channels used for each set of binaries according to the security context. For example, we switch from no secure channel for statistics to secure one or for some high volume we decide to pass from secure to no secure channel in order to save some comuting time.

9.3 Advantages/disadvantages of using DisCI

Advantages:

- DisCI can be used to set different security policies based on process. This can save the substentially amount of CPU power by avoiding the encryption/decryption of useless data.
- DisCI is transparent to application code.
- DisCI can be used by the admin to enforce security independant of how the application is implemented.
- DisCI is process level. for example, it is possible to decide to secure the cmmunications of parts of an application. or some processes without affecting the rest of the processes. For example a possible scenario can be that the admin decides to switch some applicaitons into secure connections, therefore the admin changes the dsp file to switch for processes in all the cluster, or its network of machines, then the admin rebootn all the processes and they are now all communicate to each other through ipsec/secure communications.
- DisCI allows to implement a dynamic policy. To go forth and back to secure and no secure depending on the load and general security context of the system and this at process level. For example, the administrator can choose to switch only some types of processes on a node according to the secuirty context.

9.4 How to use DisCI?

Using SetSID tool to set SIDs for different binaries, the administrator can divide binaries into different sets. Each set defines a security context. Then, the admin edits the Distributed Security Policy (DSP). Through this, he/she set the communication mode (secure, no secure) between the processes of the same set or the the processes of different sets. This at process level, transparently to the developers, i.e.; there is no need to modify the program.

9.5 Destination IP address modification

9.5.1 Modification during connection establishment

In the case of a connection established between the client and the server, there is a call to the connect(...) function, followed by the system call sys_connect(...)function. The destination IP address will be modified in the security hook (dsi_socket_connect(...)) inside the system call sys_connect(...). Whenever a client opens a connection to a server, the right IP address will be assigned to the socket.

9.5.2 Modification without connection establishment

In this case, there is no call to the connect(...) function since there is no connection to be established. The user level applications call directly sendto(...) or , sendmsg(...)function. Inside of the system call of both functions (sys_send(...) and sys_sendmsg(...)), there is a call to the sock_sendmsg(...) function which contains a security hook (dsi_socket_sendmsg(...)). The destination IP address can be modified inside of that security hook.

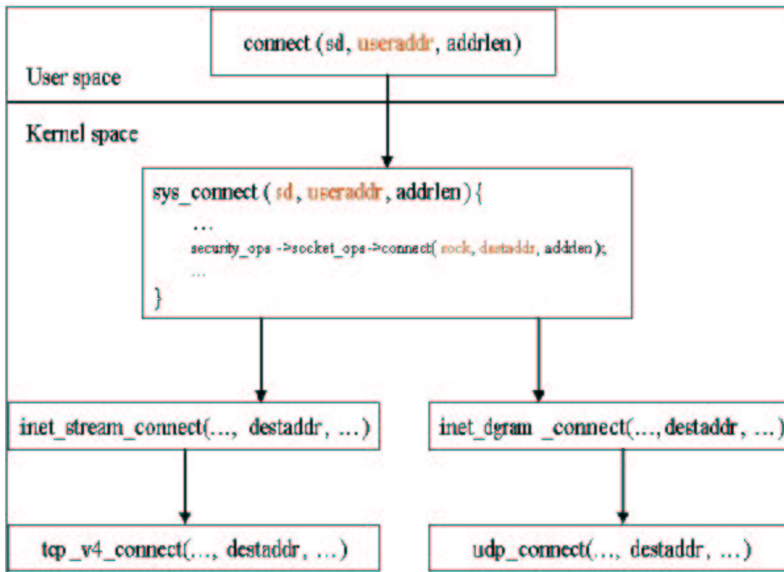

Figure 9.1: Connect method

9.6 Source IP address modification

9.6.1 Modification with connection establishment

In the security hook dsi_socket_connect(...) , the client can also choose its own source IP address to use for the connection. The source IP address of the client can be modified in that security hook.

On the server side, the source IP address can be modified in the security hook dsi_socket_bind(...) . Although, we assume that servers will listen on 0.0.0.0, so we have disabled this functionality.

9.6.2 Modification without connection establishment

The source IP address can be modified in the security hook dsi_socket_sendmsg for the client and the server.

Remark: the right source IP address is chosen automatically from the routing table depending on the destination IP address, which means that it doesn't need to be modified manually.

9.6.3 UDP IP transition during an active connection

DisCI now defines a mechanism to change destination IP address of a UDP socket at any time during its lifetime.

There are two possible scenarios that would require changing the destination IP of an UDP socket after its creation: a DSP change concerning DisCI modes of communication, and changing IP addresses of the node itself. The second case is not supported by DisCI, since it does not represent typical node behavior. However, if an administrator needs to change the communication mode from 'not secure' to ESP for certain SIDs, the DSP change will also trigger an IP change in UDP sockets.

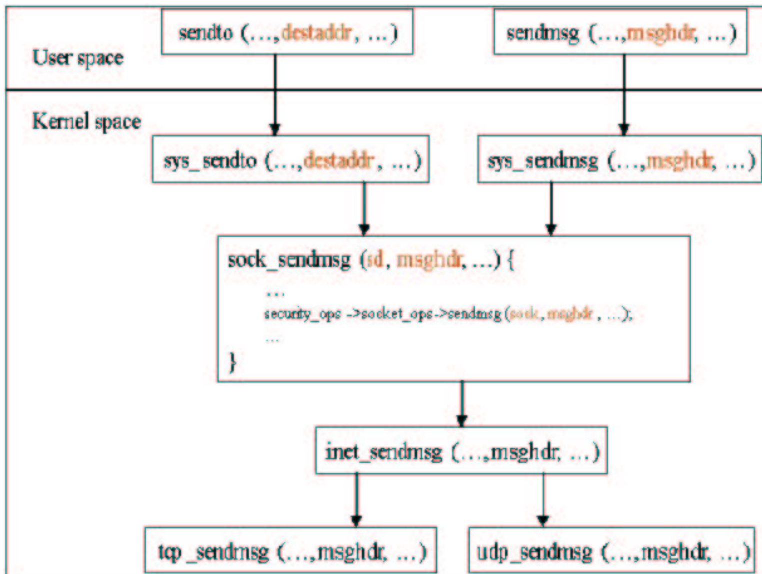

Figure 9.2: Send method

When changing destination IP of an UDP socket, the change is replicated throughout the cluster nodes by way of SCC. For any communication involving a server and a client within the cluster, both listening socket and sending socket will switch subnets. For instance, node 172.1.1.1 is serving UDP requests from 172.1.1.2. After DSP change from 'not secure' to 'ESP', node 172.1.2.1 will now serve requests from 172.1.2.2.

If an external server, listening on all interfaces (0.0.0.0), were serving requests from a client inside the cluster, communication would remain unaffected after a change of IP addresses on the client side. There are two reasons for this. First of all, external addresses remain unchanged in UDP sockets, on the client side in this case. Second, since UDP is connectionless and the server is listening on all interfaces, UDP packets will still reach their destination.

One difficulty in implementing this mechanism was understanding properly how the Linux network stack handled IP addresses. Changing the socket structure's IP address member was insufficient. The destination routing cache of the socket also had to be cleared in order for the Linux network stack to lookup the new route for the new IP destination.

9.6.4 TCP IP transition difficulties

Applying the same mechanism to TCP sockets is much more difficult and has not been implemented in DisCI at the time of writing. There are a few reasons why changing IP addresses in a TCP socket is difficult.

First consideration, TCP is stateful and reliable. TCP protocol implements different mechanisms to detect packet modification, whether it be within the kernel or on the wire.

Also, depending on network stack implementations and operating systems, TCP connections must be maintained internally since they are stateful. Manipulating a TCP socket's IP address, source or destination, can affect how the operating system

identifies a socket.

Thirdly, if during an IP transition one of the endpoints were to send a packet to the previous address, TCP protocol requires that an RST packet be sent if there is no longer any socket listening at the specified destination, which is our case. Unfortunately, a RST packet will partially destroy the TCP connection and confuse one of the endpoints.

Finally, instead of implementing a mechanism ourselves, we opted to integrate a third party solution. Some solutions are available, although we have not decided which one is most appropriate.

9.7 IP Options Modification

For coordinated network security to be possible, nodes within the cluster must be able to communicate ScIDs of client and server processes. Using the same approach as SELinux, DisCI sends ScIDs via the IP options field of a packet, the CIPSO option.

If each node were to have a unique node security ID (SnID), it would be possible to perform ScID mapping between two nodes with different security contexts. At the time of writing, DSI does assign a NSID to nodes, but assumes that a ScID has the same context on all nodes.

The IP option modification is based on CIPSO and FIPS-188 standards.

The IP option, CIPSO, follows this specific format recognized by DisCI:

```
                  +--------+--------+
                  | CIPSO  | Length |
+--------+--------+--------+--------+
|      Domain Of Interpretation     |
+--------+--------+--------+--------+
|Freeform| Length |  ScID  | Length |
+--------+--------+--------+--------+
|                Data               |
+--------+--------+--------+--------+
| NodeID | Length |      Data       |
+--------+--------+--------+--------+
| (con't) Data    |
+--------+--------+
```

The first field, CIPSO, is the number of the IP option. Since this option is officially recognized, the chance of a packet with this option passing routers is increased, however, this only applies inside a DSI cluster. Packets with outside destinations do not have IP options. Next, length defines the entire length of the option including the CIPSO and length fields. The domain of interpretation is not used by DisCI, but included to be CIPSO compliant. Freeform means that we are defining our own fields and not using CIPSO pre-defined fields. The second length is the remaining length in the option including the freeform and length fields. 'SID' is not actually the ScID of the process but an identifier indicating that following data will be a ScID. The third length is the length of the data including the ScID and length fields. The data field contains the SID. NodeID is an identifier, 'length' is length of data plus previous two fields. The data field contains the NodeID.

When a packet arrives at a remote node, the ScID is stored in the security structure of the packet.

Important remark: the process sending the packet might not be the current process when the kernel actually sends the packet, because there is a process of queueing involved. As a test case, we had 100 threads send 1 ping each with the unique ScID of the sending process in the data of the ping and the ScID of the current process in the IP options. 2 out of 25 pings did not have the correct SID.

9.8 Digital Signatures

There are different levels of operations defined by DSP as defined in the section 10. For time being, we plan to use already existing open source projects to do this. Precisely, Tripwire for the level 1 and CryptoMark for level 3.

9.9 Open Questions

9.9.1 User Mode Linux

Problem

Cannot apply the LSM patch to the UML kernel.

Possible solution

In fact, there are some arch-specific (I386) stuff in the LSM patch. It is possible and easy to patch UML manually by applying the same modifications to the 'uml' arch as those applied to the I386 by the patch. Patch UML first, then LSM and do the following modifications :

1.Add the following line at the end of the /usr/src/uml/um/config.in file

```
Source security/config.in
```

2.Do the modifications to the /usr/src/uml/Arch/uml/kernel/ sys_call_table.c file

```
a. add the following line below   extern syscall_handler_t  sys_gettid
extern syscall_handler_t sys_security

b. find the line below, remove it and replace it by the line at c.
[__NR_security] = sys_ni_syscall

c. add the line below
[__NR_security] = sys_security

To-Do : test whether it works.
```

9.9.2 Freeswan1.96

Problem

Freeswan1.96 doesn't support packets employing IP options.

Solution

The bug is fixed. Here is the patch that can be applied to freeswan1.96.

```
File freeswan1_96_ip_option-13-august-2002.patch

--- ipsec_tunnel.c Mon Aug 12 14:53:16 2002
+++ ipsec_tunnel.c.modif Mon Aug 12 15:10:03 2002
@@ -601,7 +601,7 @@
   * Sanity checks
   */

-if ((iph->ihl << 2) != sizeof (struct iphdr)) {
+ if (0)/*((iph->ihl << 2) != sizeof (struct iphdr))*/ {
  KLIPS_PRINT(debug_tunnel,
      "klips_debug:ipsec_tunnel_start_xmit: "
      "cannot process IP header options yet.  May be mal-formed packet.\n"); /*XXX*/
@@ -1142,6 +1142,7 @@
 #ifdef CONFIG_IPSEC_IPIP
  case IPPROTO_IPIP:
  headroom += sizeof(struct iphdr);
+ pyldsz += iphlen - sizeof(struct iphdr);
  break;
 #endif /* !CONFIG_IPSEC_IPIP */
  case IPPROTO_COMP:
@@ -1443,6 +1444,7 @@
 #ifdef CONFIG_IPSEC_IPIP
  case IPPROTO_IPIP:
  headroom += sizeof(struct iphdr);
+ iphlen = sizeof(struct iphdr);
  break;
 #endif /* !CONFIG_IPSEC_IPIP */
 #ifdef CONFIG_IPSEC_IPCOMP
```

9.10 DisCI Conclusion

The results show clearly that the socket layer approach works very well. It has several advantages. First of all, it satisfies DisCI needs. Second, the implementation is done in DSM, so there is no need to make another module. Third, the IP address modifications have been done at high level in such a way that it will not confuse the system and will have less impact if the TCP/IP stacks change in the future. Fourth, it is simple to implement, flexible to future changes and needs.

Chapter 10

Integrity service

In this chapter, we explain the preliminary ideas for the integrity service of DSI.

10.1 Introduction

The Integrity Service we refer to here does not deal with integrity of communications between processes (which is taken in charge by the DisCI module, chapter), but with integrity of files on various nodes of the cluster.

For instance, we should protect (1) the kernel from loading unauthorized or malicious kernel loadable modules, (2) executing unsecure code (such as virii) and (3) individual files from malicious modification (such as the DSP file etc).

The former is currently being investigated, maybe using a tools such as Cryptomark[2]. There is no implementation yet.

10.2 Levels of digital signature verification

Digital signature verification could be set through DSP to different levels:

- Level 1: Digital signatures verification at user-level: in order to avoid the installation of malicious software, digital signatures are added to binaries. These digital signatures are periodically checked (by preference as a background task, during low-level load period for the server) and if any incoherence is detected, the alarms are sent to the security server.
- Level 2: Digital signatures verification when loading from hard disk for the first time.
- Level 3: The signature for binaries is verified each time that they are used to create a new process (execv+fork).

Chapter 11

Tools

In this chapter we explain different tools implemented.

At the time of writing, LSM mailing list is reporting that the integration of LSM patch in kernel tree will possibly discard the system call sys_security. The majority of the tools described lower will be directly affected if this is the case.

In prevision of this removal and to prevent major modification of these tools, we have started implementing a char device interface to communicate with the module. The char device takes a string argument containing a number identifing the action to be performed followed by the number of arguments required by the action. For more detail see section 5.4.

11.1 DciInit

The DciInit tool directly calls a system call implemented by DSM, named sys_security (ref 5.7). DciInit is responsible for defining the IP addresses that DCI will be using (ref **??**). The only parameter passed to DciInit is the filename for DCI configuration.

11.2 UpdatePolicy

The UpdatePolicy tool is a helper tool that reads the DSP file passed as argument and directly calls the sys_security (ref 5.7) to load the rules in the DSM of the node. UpdatePolicy will load each line of the policy file into DSM's cache. The only parameter required by UpdatePolicy is the policy filename. Beware: this tools does not propagate the policy to other nodes.

Do not confuse this tool with the dsiUpdatePolicy tool in `dsi/SS/test/demoSecOM` which sends an XML UpdatePolicy event (see §8.1.4)to the local SS, the SS reading and analyzing the policy and then propagating the policy to all SMs.

11.3 dsiUpdatePolicy

This tool is currently found in `dsi/SS/test/demoSecOM`. It takes the DSP file as argument. It should be launched only on the SS node. It sends an XML UpdatePolicy

event to the SS with the DSP file as argument. The DSP file is then loaded, validated, and propagated to all SMs.

```
$ cd ~/dsi/SS/test/demoSecOM/
$ ./dsiUpdatePolicy ~/dsi-0.2/etc/DSP.xml
```

11.4 ChangeProcSID

The ChangeProcSID tool directly calls a system call implemented by DSM, named sys_security (ref 5.7). ChangeProcSID will change the ScID of the given Process ID (PID). This tool takes two parameters: the PID of the process and the new ScID.

11.5 SetSID

SetSID is similar to ChangeProcSID, but this tool will set the ScID of a binary. Subsequently, when the binary is executed, it will use the ScID stored in its disk image. SetSID takes two parameters: the filename of the binary and the new ScID. The ScID will be stored in the binary file. Note: this will not affect processes already running, use ChangeProcSID to that effect.

11.6 SS_Console

TO BE COMPLETED.

11.7 SetNodeID

TO BE COMPLETED.

11.8 ls_dsi

ls_dsi is a DSI tool, similar to the native linux *ls* command, which lists the content of a directory as well as the ScID embedded in binary files (BScID). To use *ls_dsi*, you can either enter the directory of interest and type *ls_dsi*, as shown below, or you can simply give the path of the directory as an argument. Since every linux configuration is different, you may need to give the whole path to *ls_dsi* in order to execute it.

```
[lmcgaii@lmcpc116076 server]# cd /dsi/user/server
[lmcgaii@lmcpc116076 server]# /home/lmcgaii/dsi/user/tools/ls_dsi
PERMISSION      USER    GROUP   BScID   FILE
drwxr-xr-x      lmcgaii lmcgaii -       CVS
-rwxr-xr-x      root    root    6       UDPServer
-rwxr-xr-x      lmcgaii lmcgaii -       Makefile
-rw-r--r--      lmcgaii lmcgaii -       TCPServer.c
-rwxr-xr-x      lmcgaii lmcgaii -       UDPServer.c
-rwxr-xr-x      root    root    3       TCPServer
```

```
[lmcgaii@lmcpc116076 tools]# ./ls_dsi /home/lmcgaii/dsi/user/server
PERMISSION      USER    GROUP   BScID   FILE
drwxr-xr-x      lmcgaii lmcgaii -       CVS
-rwxr-xr-x      root    root    6       UDPServer
-rwxr-xr-x      lmcgaii lmcgaii -       Makefile
-rw-r--r--      lmcgaii lmcgaii -       TCPServer.c
-rwxr-xr-x      lmcgaii lmcgaii -       UDPServer.c
-rwxr-xr-x      root    root    3       TCPServer
```

The BScID field is the field that presents the most interest; it gives for each executable file (e.g. emacs, apache, etc.) the ScID that is embedded in its ELF format. Note however that not all executables have ScIDs. Files that are not in the ELF format do not have an ScID, hence *ls_dsi* prints '-' when it encounters those types of file. One can easily assign or change the ScID of an ELF file by using the *SetSID* tool.

11.9 ps_dsi

ps_dsi is a DSI tool, similar to the native linux ps command, which lists important information about running processes including their respective ScIDs. Note that the ScID of a process may refer to both source or target security contexts. Before using the ps_dsi tool, ensure that the DSM module (/dsi/lsm/dsm.o) is loaded in the kernel since ps_dsi executes a system call to this module in order to get a process's ScID. If the DSM module is not loaded, ps_dsi will print out '-' instead of the process's ScID. As shown below, to list information about running pocesses, just type ps_dsi. Once again, depending on your linux configuration, you may need to give the whole path to ps_dsi in order to execute it.

```
[lmcgaii@lmcpc116076 tools]# ./ps_dsi
PID     ScID    USER    STAT    CMD
1       2       root    S       init
2       2       root    S       keventd
3       2       root    S       ksoftirqd_CPU0
4       2       root    S       kswapd
...
3912    1       telorb  S       bash
4059    2       telorb  S       bash
4534    2       telorb  S       bash
4565    2       telorb  S       bash
4601    3       telorb  S       xemacs
4626    2       telorb  S       xemacs
4709    2       telorb  S       UDPServer
4710    3       telorb  S       UDPclient
4717    2       telorb  R       ps_dsi
```

As the DSM module is loaded, each existing process gets a default ScID, which, by our convention, is 2 (DSI_SID_NORMAL). The creation of each new process is governed by the rules defined in the policy_test file (especially the class transition rules which defines what ScID a new process will get depending on its parent) for instance, a process having a BScID=3 may end up running with an ScID=2 if such

a class transition exists. It is possible to create new transition rules by editing the policy_test file and updating the policy using UpdatePolicy or to assign new ScIDs to running processes with ChangeProcSID.

11.10 PrintPolicy

This tool offers the capability to dump all the rules currently loaded in the DSM. The rules are dumped in the /var/log/messages log file. *PrintPolicy* can be useful for debugging purposes or simply to understand what goes on inside DSM.

PrintPolicy will print out rules in DSM numerical rule format as opposed to the DSP's XML rule format. Preceding each rule in between brackets, is the slot number where the rule resides in the dsi_cache array. There may be many rules residing in the same slot and all the rules in a single slot are joined together to create a chained list. Finally, the number of rules currently enforced by DSM and the number of free nodes available for new rules are printed out.

```
Apr  9 19:02:12 lmcpc116076 kernel: START OF THE CURRENT DSM SECURITY POLICY
Apr  9 19:02:12 lmcpc116076 kernel: [3] 12 0 0 0 3 0xffffffff
Apr  9 19:02:12 lmcpc116076 kernel: [3] 2 0 0 0 3 0xffffffff
Apr  9 19:02:12 lmcpc116076 kernel: [17] 12 1 12 1 1 0xffffffff
Apr  9 19:02:12 lmcpc116076 kernel: [17] 3 1 2 1 1 0xffffffff
Apr  9 19:02:12 lmcpc116076 kernel: [17] 2 1 2 1 1 0xffffffff
Apr  9 19:02:12 lmcpc116076 kernel: [17] 1 1 2 1 1 0xffffffff
...
Apr  9 19:02:12 lmcpc116076 kernel: [274] 1 1 3 1 2 0xffffffff
Apr  9 19:02:12 lmcpc116076 kernel: [274] 1 1 1 1 2 0xffffffff
Apr  9 19:02:12 lmcpc116076 kernel: [275] 12 1 1 1 3 0xffffffff
Apr  9 19:02:12 lmcpc116076 kernel: [275] 3 1 3 1 3 0xffffffff
...
Apr  9 19:02:12 lmcpc116076 kernel: [277] 2 1 1 1 5 0x1
Apr  9 19:02:12 lmcpc116076 kernel: [277] 1 1 3 1 5 0x1
Apr  9 19:02:12 lmcpc116076 kernel: [277] 1 1 1 1 5 0x1
Apr  9 19:02:12 lmcpc116076 kernel: TOTAL RULES: 55
Apr  9 19:02:12 lmcpc116076 kernel: FREE NODES:  355
Apr  9 19:02:12 lmcpc116076 kernel: END OF THE CURRENT DSM SECURITY POLICY
```

The printk messages used to print the rules loaded in DSM are sent to Syslog and usually, it redirects those messages to /var/log/messages. If you don't want the policy to be printed there you can always send it to the console by editing /etc/syslog.conf and removing the #kern.* commented line :

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
kern.*                                                  /dev/console
```

Once you've uncommented the line, you have to kill and restart Syslogd. Finally, as root, launch the xconsole.

Chapter 12

Testing DSI

In this chapter, we discuss a set of tests useful to check DSI's features and performance.

12.1 Client Server Test Programs

To test network functionality in DSI, TCP and UDP servers have been created along with their associated clients. Clients simply send 'Hello server' to the server. In UDP mode, the server does not answer. Whereas, in TCP mode, the protocol forces a response in the form of ACKs.

See `TCPclient.c` and `UDPclient.c` in `$DSI_ROOT_DIR/user/client`

Client programs are to be found in $DSI_ROOT_DIR/user/client. . The socket_client.c file tests all client-side socket functions.

See `TCPServer.c` and `UDPServer.c` in `$DSI_ROOT_DIR/user/client`

Server programs are to be found in $DSI_ROOT_DIR/user/server. . Servers listen on 127.0.0.1 (or on all interfaces if 0.0.0.0 is used), port 8800.

Those programs may be used to test DSI's network functionalities. Start the server first, and then the client.

12.2 DSM filesystem testing

The DSM filesystem tests concern different functionality in DSM. Mainly, tests are run at kernel level and the results are written to the /var/log/messages file.

1. Load DSM.
2. check the major/minor number through

```
[root@colby lsm]# cat /proc/devices
....
254 DSI_module
....
```

3. The major number being found (in above example 254), we make the char device file /dev/DSI_module

```
root@colby#mknod /dev/DSI_module c 254 0

root@colby# ls -l /dev/DSI_module
crw-r--r--    1 root     root     254,   0 Feb 14 16:24 /dev/DSI_module
```

4. Once the file has been created by mknod command, you can use any tool to write into the /dev/DSI_module device. For example, write down your command line into a file and cat the file to /dev/DSI_module. For example:

```
[root@colby lsm]#cat args_test
7 2 3 4 5 6 0x01

[root@colby lsm]# cat args_test > /dev/DSI_module
```

   Notice that there is only the first argument which is important. The other arguments are just to comply with the general rule of waiting for 6 arguments.

5. Check the results of the filesystem testing in /var/log/messages file.

12.3 DSM unit testing

DSM can be separated into eight main areas of action: device, cache, tasks, inode, netlink, socket, access_control, and DCI. For each area, separate unit tests have been designed and can be found in dsi/lsm/dsi_test.c. The function of unit testing is to make sure basic functionality is still present independent of its implementation. Some important exceptional cases are also tested, however, it must be noted that unit testing is not an ultimate certification of the code. Developers are encouraged to write unit tests for each new functionality added.

To start the unit testing, a userland tool is provided in dsi/user/tools named TestDSM. No parameters are required, simply that DSM module be loaded in kernel memory. To start testing:

```
% cd dsi/user/tools
% ./TestDSM
```

Results will be output to /var/log/messages or to console depending on the settings in syslog.conf.

IMPORTANT: Using TestDSM will make DSM module insecure! This tool changes the policy and other internal information for testing and has not been implemented to restore original internal settings. It does restore to the extent that the following test might need a consistent state for DSM, but DSM must be reloaded to ensure proper security.

12.4 DSM automatic scenario testing

To complement unit testing which is internal to the DSM kernel module, a functional testing framework has been defined. The perl module named scenario.pm in

dsi/user/test contains utilities to build scenarios for testing user applications with DSM.

- start_app: Start application and return its pipe. Caller is responsible for closing the pipe.
- stop_app: Kill application using SIGTERM, which allows program to define a signal handler to flush its stream to logfile (SIGKILL cannot be trapped).
- check_log: Check application log for a specified string.
- print_title: Print title of a test clearly.

An example scenario is shown in perl script dsi/user/test/scenarioA.pl. It performs three tests between client TelecomClient and servers RingBellBE/EP and PhoneManiaBE/EP. The first test simply makes sure that basic connectivity can be established between the client, entry point (EP), and back end (BE). The second test performs the same operations, but with DSM. The last test tries connecting an entry point with a back end it is not supposed to be able to talk to (because SCIDs are arranged for this in the policy file). After each timed test, the logs are checked for specified strings to confirm results. It is suggested to look at the requirements listed at the begining of scenarioA.pl before starting.

It's important that applications tested with perl script utilities meet certain requirements. In the case of scenario A, the client and servers write their PID in a file with the same name as the executable postfixed with '.pid'. They also write output to a log file with same name and postfixed with '.log'. These files are automatically deleted by the perl script after tests are done.

To start scenario A, current working directory is very important:

```
% cd dsi/user/test
% ./scenarioA.pl
```

Results will be summarized at the end of all tests on console and details of execution can be seen in console during execution.

12.5 DisCI functionality tests

12.5.1 Context

DisCI (Distributed Confidentiality and Integrity) services provide three channels associated to IPSec transmission modes: No Security mode, AH mode and ESP mode. Currently, each of those channels corresponds to an IP address[1] (see chapter ??). This justifies why each node is assigned three IP addresses that are associated to each transmission mode. The DSP specifies the communication channel that a given subject identified by a ScID on node SnID can use to reach a target object identified by a TScID on node TSnID.

At kernel level, permissions are represented using the following format:

```
SScID SSnID TScID TSnID CLASS PERMISSION
```

[1] Work is under progress to find another solution.

This is only an internal format, the end user writes rules according to DSP's syntax (see section 7).

See dsi.h in lsm

DisCI permissions are represented by the specific DSI_CLASS_DCI class (*0x5*). For instance, if a client application with SScID = 1 on node SSnID = 0 is allowed to connect to a server application with TScID = 1 on node TSnID = 1 in ESP mode (*0x3*) then, at kernel level, this is represented by :

```
1 0 1 1 5 0x003
```

Whenever a communication is initiated between a source node and a target node, the source node's IP address and/or the destination node's IP address are modified to match the permission granted in the DSP File for SScID/SSnID and TScID/TSnID involved.

12.5.2 Steps to Verify DisCI Functionality

1. Make sure you assigned the 3 IP addresses to each machine in the correct format. This can be done for the client by creating aliases to your network interface such as:

   ```
   % ifconfig eth0:1 172.1.1.2 up
   % ifconfig eth0:2 172.1.2.2 up
   % ifconfig eth0:3 172.1.3.2 up
   ```

   On the server side:

   ```
   % ifconfig eth0:1 172.1.1.3 up
   % ifconfig eth0:2 172.1.2.3 up
   % ifconfig eth0:3 172.1.3.3 up
   ```

See dsi_dci.h in $DSI_ROOT_DIR/lsm

We also assume you edited dai_policy.conf file at each node (client machine and server machine) to define the associations between each IP address and the mode it will be used for :

- 0x00000001 is for No Secure Mode,
- 0x00000002 is for AH mode
- and 0x00000004 for ESP mode.

First test the "no security" mode:

- Go to DSP file on the security server machine of DSI demo:

  ```
  % cd /root/dsi/etc/DSP
  ```

- Add to the file the following lines:

  ```
  1 0 1 1 5 0x00000001
  3 0 1 1 5 0x00000002
  3 0 3 1 5 0x00000004
  ```

- . Update the policy on the security server machine

```
% cd /root/dsi/SS/test/demoSecOM
% ./dsiUpdatePolicy
```

- Set the server ScID to 1 on the server machine

```
% cd /root/dsi/user/server
% SetSID UDPServer 1
```

- Start the server application

```
% ./UDPServer
```

- Set the client ScID to 1 on the client machine

```
% cd /root/dsi/user/client
% SetSID UDPClient 1
```

- Start the client application

```
% ./UDPClient
```

- Launch a sniffer on the client side to watch the traffic between the client and the server

```
% ethereal &
```

When SScID and TScID are both set to 1, DSP file allows "No Secure communication" mode between node SSnID = 0 and TSnID = 1. You can check that packets sent by the client to the server have the source IP address 172.1.1.2. No modification has been done to the server IP address.

Then, test for instance AH communication:

- Stop the client application on the client machine
- Set the client ScID to 3

```
% cd /root/dsi/user/client
% SetSID UDPClient 3
```

- Restart the client application

```
% ./UDPClient
```

When SScID is set to 3 and TScID to 1, DSP file allows AH communcation mode between node 0 and 1. You can check that packets sent by the client to the server have the source IP address 172.1.2.2 though the client IP address was previously 172.1.1.2. The packets destination IP address is now 172.1.2.3 (previously 172.1.1.3). So a modification has been performed on the server IP address to match the permitted specified IP address.

Finally, test the ESP mode:

- Stop both server and client application
- Set the server ScID to 3 on the server machine

```
% cd /root/dsi/user/server
% SetSID UDPServer 3
```

- Restart the server application

```
% ./UDPServer
```

- Restart the client application on the client machine

```
% ./UDPClient
```

Both SScID and TScID are now set to 3. DSP allows ESP communication mode between node SSnID = 0 and TSnID = 1.

12.5.3 Integration Tests

Integration means that the DSM code and the IP modification code are put together. The tests below show the results of the integration.

Case A: DSM + DisCI

TEST	Without IPSEC	with IPSEC
Mount nfs (predator)	Ok	Ok
Netscape direct connection	Ok	Ok
Netscape proxy (Internet)	Ok	Ok
Telnet, Ftp, ssh	Ok	Ok
Demo	Ok	Ok
Ping	Ok	Ok

Case B : DSM + DisCI + IP option

TEST	Without IPSEC	with IPSEC
Mount nfs	OK	Not tested (should be OK)
Netscape direct connection	OK	OK
Netscape proxy (Internet)	FAILED (normal)	FAILED (normal)
Telnet	OK	OK
Ssh	FAILED (normal)	FAILED (normal)
Demo	OK	OK

Chapter 13

Debugging DSM

This chapter describes the approach taken for debugging DSM.

13.1 dsi_debug.h

When programming LKMs, userland debugging techniques become unavailable. For instance, one way to interactively debug a kernel module would be with kdb in assembler or kgdb through the serial port. The other alternative is using printk.

For efficient use of printk, a number of debugging levels, or classes, have been defined. The global debugging level is represented by a vector of bits, each bit represents a different debugging class. This means that different levels can be combined or selected individually.

See `DSM_PRINT()` in `lsm/dsi_debug.h`

In dsi_debug.h , a global debugging variable is set at compile time. In the module, when debug output is necessary, the DSM_PRINT definition is used. It takes as parameters the level of debugging, the string format, and an arbitrary number of arguments. This is very similar to printf, except for the custom debug level. Next, a simple check (logical AND) is performed with the parameter debug level and the global debugging variable. If two bits correspond, then the string is output with printk.

13.2 Buffering and printk

The purpose of logging and debugging is obtaining information from the program in a reliable fashion. There is an issue with printk, since its output string is buffered. Here is an excerpt from printk source code:

See `printk()` in `kernel/printk.c`

```
394 /*
395 * This is printk.  It can be called from any context.  We want it to work.
396 *
397 * We try to grab the console_sem.  If we succeed, it's easy - we log the output and
398 * call the console drivers.  If we fail to get the semaphore we place the output
399 * into the log buffer and return.  The current holder of the console_sem will
400 * notice the new output in release_console_sem() and will send it to the
```

```
401 * consoles before releasing the semaphore.
402 *
403 * One effect of this deferred printing is that code which calls printk() and
404 * then changes console_loglevel may break. This is because console_loglevel
405 * is inspected when the actual printing occurs.
406 */
```

```
495 /**
496 * release_console_sem - unlock the console system
497 *
498 * Releases the semaphore which the caller holds on the console system
499 * and the console driver list.
500 *
501 * While the semaphore was held, console output may have been buffered
502 * by printk().  If this is the case, release_console_sem() emits
503 * the output prior to releasing the semaphore.
504 *
505 * If there is output waiting for klogd, we wake it up.
506 *
507 * release_console_sem() may be called from any context.
508 */
```

The problem is that printk might possibly never log an entry in certain rare conditions. A possible solution might be to look into the mechanism of the Linux Trace Toolkit. At the time of writing, this potential problem has not been solved.

Chapter 14

Benchmarking DSI

14.1 LMBench results

We ran LMBench 3.0 [11] 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 [22] 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 14.1).

A detailed explanation of tests can be found at [11]. 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.

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. This tool measures the performances for already established TCP connections. Results presented

Test type	Base	DSM	Overhead
Stat	1.98	1.94	-2.0%
Open / Close	2.68	2.68	0%
Fork	92.81	93.58	0.82%
Exec	322.56	328.33	1.78%
Sh proc	2150	2140.75	-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 14.1: *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.*

in table 14.2 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 14.2: *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.

14.2 DisCI Benchmarks

14.2.1 Dgram

```
Server: testar1, doesn't need to run anything.
Client: testar2, run dgram and ncdgram (no connection)
```

```
 Characteristics
```

Machine :	testar1	testar2
Processor :	Pentium III 400 MHz	Pentium III 500 MHz
Cache Size :	256 KB	512 KB
Memories :	256 MB	256 MHz
OS :	Linux, Red Hat 7.2	Linux, Red Hat 7.2
Kernel :	2.4.17	2.4.17

	Module			Overhead		
	NO LSM	LSM	LSM+DisCI	LSM	LSM+DisCI	DisCI
With connection	10.300	11.693	11.805	13.5%	14.6%	1.0%
Without connection	10.387	11.575	12.480	11.4%	20.1%	7.8%

14.2.2 Dgramresp

`Server: testar1, run servresp`

`Client: testar2, run dgramresp and ncdgramresp (no connection)`

	Module			Overhead		
	NO LSM	LSM	LSM+DisCI	LSM	LSM+DisCI	DisCI
With connection	8.915	10.354	10.623	16.2%	19.2%	2.6%
Without connection	9.016	10.343	10.678	14.7%	18.4%	3.2 %

References

http://www.kernel.org/ Kernel repository.

http://www.nsa.gov/selinux/ SELinux created by NSA and also based on LSM patch to the kernel.

http://lsm.immunix.org/ The Linux Security Modules (LSM) project provides a lightweight, general purpose framework for access control.

http://www.linuxjournal.com/article.php?sid=6053 The Linux Journal featuring an article on DSI.

Glossary

AVL Access Vector List

DisAC DIStributed Access Control

DisCI DIStributed Confidentiality and Integrity

DSI Distributed Security Infrastructure

DSM Distributed Security Module

DSP Distributed Security Policy

LKM Linux Kernel Module

ScID Security Context IDentification

SnID Security Node IDentifier

SScID Source Security Context IDentification

TScID Target Security Context IDentification

Bibliography

[1] Apvrille A., Pourzandi M., *Protéger un réseau de machines distribuées contre un débordement de buffer... d'un seul coup*, MISC 7, May - June 2003 (in French), http://www.miscmag.com

[2] *Cryptomark* http://www.immunix.org/cryptomark.html

[3] *Document Object Model* (DOM) http://www.w3c.org/DOM/

[4] Dragovic, B. *LinSec - Linux Security Protection System* University College London, April 2002, http://www.linsec.org/doc/final.

[5] Foster I., Kesselman C., Tsudik G., Tuecke G. *A Security Architecture for Computational Grids* 5th ACM Conference on Computer and Communication Security

[6] ITU-U Recommendation X.800 *Security Architecture for Open Systems Interconnection for CCITT Applications* ITU-T (then CCITT), 1991

[7] ISO 10181-3 *Security Frameworks for Open Systems: Access Control Framework* ISO, 1996

[8] Loscocco P. *Security-Enhanced Linux* Linux 2.5 Kernel Summit, San Jose (Ca) USA, 2001, http://www.nsa.gov/selinux/docs.html

[9] 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.

[10] LSM development team *Linux Security-Module (LSM) framework* 2001, http://lsm.immunix.org/

[11] 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.

[12] MontaVista *MontaVista Linux Carrier Grade*, Edition 2.1, White paper, http://www.montavista.com/dswp/index.html

[13] Morris, J. *Selopt: Labeled IPv4 networking for SE Linux* http://www.intercode.com.au/jmorris/selopt

[14] Grisby D., Sai-Lai L. *The omniORB version 4.0 User's Guide* August 2002, http://omniorb.sourceforge.net

[15] *omniEvents* http://sourceforge.net/projects/omnievents

[16] Open Source Development Lab, *Carrier Grade Linux*, http://www.osdl.org/projects/cgl

[17] Pourzandi M., Apvrille A., Gingras E., Medenou A., Gordon D., *Distributed Access Control for Carrier Class Clusters*, Parallel and Distributed Processing Techniques and Applications (PDPTA'03), Las Vegas, June 2003 (to appear), http://www.ashland.edu/~iajwa/conferences/2003/PDPTA/pdpta.html

[18] 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.

[19] *SAX*. http://sax.sourceforge.net

[20] Schreiner R., Lang U. *MicoSec User's Guide* http://www.objectsecurity.com/micosec.html

[21] 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.

[22] 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.

[23] *XML Schema*, W3C Recommendation, May 2001, http://www.w3c.org/XML/Schema

Index

Alarm event, 51

Benchmarking DSI, 75

Certificates, 13
ChangeProcSID, 64
Client Server Tests, 67
cluster, 1
CORBA, 13, 24

DAC, 35
DciInit, 63
Default ScID, 43
destination IP address, 54
digital signature, 61
Digital Signatures, 58
DisAC, 35
DisCI, 53
DisCI benchmarks, 76
DisCI conclusion, 59
DisCI functionality tests, 69
DisCI preemptiveness, 58
DisCI rule, 42
Distributed Access Control, 35
DSI installation, 11
DSI testing, 67
DSI_SID_NORMAL, 65
dsiUpdatePolicy, 63
DSM, 26, 37
DSM automatic scenario testing, 68
DSM filesystem testing, 67
DSM rule, 33
DSM Rule event, 50
DSM unit testing, 68
DSP, 25, 39
DSP Generator, 46
DSP update, 28, 46

freeswan, 58

Heart beat, 50

integration tests, 72
integrity, 61
IP options, 57

Kernel memory, 29
Kernel module, 73
Kernel patch, 11
kernel versioning, 18

LKM, 73
ls_dsi, 64
LSM, 11
LTT, 74

MAC, 35
Makefile options, 18
module versioning, 18

NAT, 53
Network rule, 42

omniEvents-troubleshooting, 15
omniORB.cfg, 15
OpenSSL, 13

permissive, 43
policy file, 39
Policy rule matching, 30
printk, 73
PrintPolicy, 66
Process rule, 40
ps_dsi, 65

Requirements, 2, 11
restrictive, 43

SCC, 49
Scenario, 21
secure communication channel, 8
security architecture, 5
security manager, 7
Security rules, 39
security server, 6
security service, 9
SetNodeID, 64

SetSID, 64
socket alarms, 29
Socket init rule, 41
socket permissions, 29
Socket rule, 40
source IP address, 55
spinlock, 32
SS_Console, 64
sys_security, 32

TCP, 56
Transition rule, 43

UDP, 23
unresolved symbol, 18
UpdatePolicy, 63
user mode Linux, 58

Warning event, 51

Xerces, 15
XML advantages, 49
XML events, 49
XML information event, 51
XML namespace, 49
XML parsing, 52
XML Schema, 44
XML Update Policy, 50