
The Impact of Linux Superuser Privileges on System and Data Security within a Cloud Computing Storage Architecture

Diploma Thesis

Author: Steffen Schreiner

Computer Science, April 2009

Prof. Dr. Johannes Buchmann



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Department of
Cryptography and
Computeralgebra

Diploma Thesis in Computer Science

The Impact of Linux Superuser Privileges on System and Data Security within a Cloud Computing Storage Architecture

by Steffen Schreiner

April 2009

The thesis was developed in cooperation with a company under a nondisclosure agreement. This document is public, all company and product informations were pseudonymized and it has been edited for content.

Supervisor and Examiner:

Prof. Dr. Johannes Buchmann

Cryptography and Computeralgebra

Department of Computer Science

Technische Universität Darmstadt

Declaration of Academic Honesty

I hereby declare that this diploma thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, April 2009

Steffen Schreiner

To Hanne and Gerhard

Du kannst! So wolle nur!
(You can! Just will it!)

J.W.v.Goehe, Faust I

Abstract

Discretionary access controlled operating systems like Linux are unsuited to be used as a basis for service-oriented environments or within the scope of cloud computing, as it is not possible to partition or decompose the power of superuser privileges in a convenient way. This deficiency of possible decomposition is not only a threat in the face of potential privilege escalation and exploitation or a hostile administrator, but is a major functional drawback.

This thesis develops seven problem statements with respect to the application ITComp Clustered Data Services (CDS). These concern the Linux superuser privileges' characteristics *omnipotence*, *resistiveness*, *unavoidable obtainment*, *cover-up assistance*, *undermined confidentiality*, *unfeasible compartmentation*, and *degraded compliance*.

Analyzing four available Linux security mechanisms, SELinux stands out as the only mechanism suited for a potential solution. SELinux is analyzed and described in detail with respect to its potential application in CDS. Finally, the thesis gives an outline for the advancement of the SELinux Reference Policy to incorporate the software components of the ITComp CDS cluster.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objective of the Paper	3
1.3	Outline	3
2	Background and Fundamentals	5
2.1	Security Goals	5
2.2	Access Control	6
2.2.1	The Principle of Least Privilege	8
2.2.2	Discretionary Access Control	8
2.2.3	Mandatory Access Control	9
2.2.4	Role Based Access Control	10
2.2.5	Superuser Privileges	10
2.2.6	Reference Monitor	11
2.2.7	Type Enforcement	11
2.2.8	Bell-LaPadula Model	12
2.3	Threats, Vulnerabilities and Attacks	13
2.3.1	Flawed and Malicious Software	13
2.3.2	Privilege Escalation	14
2.3.3	Rootkits	14

2.3.4	Zero-Day Attacks	15
2.3.5	Covert Channels	15
3	The Linux Operating System and the Superuser	17
3.1	Linux Architecture	17
3.2	Access Control	19
3.3	The Superuser	19
3.3.1	Setuid	20
3.3.2	Sudo - Substitute User Do	21
3.3.3	POSIX Capabilities	22
3.3.4	Chroot	23
3.3.5	Conclusion	24
4	ITComp Clustered Data Services	25
4.1	Introduction	25
4.2	ITComp Very Large File System	26
4.3	Architecture	27
4.3.1	Clustered Trivial Database	30
4.3.2	Services	31
4.3.3	Directory Service	31
4.3.4	Backup and Archiving	32
4.4	Administration	32
4.5	Signed Software Updates	33
4.6	Multi-Client Capability	34
5	The Problem of the Linux Superuser	35
5.1	Omnipotence	36
5.2	Resistiveness	37
5.3	Unavoidable Obtainment	38
5.4	Cover-up Assistance	38
5.5	Undermined Confidentiality	39

5.6	Unfeasible Compartmentation	40
5.7	Degraded Compliance	41
6	Requirements to Secure the CDS Cluster	43
6.1	Security Goals of CDS	43
6.2	Prerequisites and Dependencies	44
6.3	CDS Cluster Requirements	46
6.4	A Role Design for Administration	47
6.4.1	Analysis and Reporting Role	48
6.4.2	Access Management Role	48
6.4.3	Service Management Role	49
6.4.4	Resource Management Role	49
6.4.5	System Maintenance Role	50
6.4.6	Data Management Role	50
6.4.7	Remaining Privileged Access	51
7	Mechanisms to Confine the Linux Superuser	53
7.1	Linux Security Modules	54
7.2	AppArmor	55
7.3	Grsecurity	56
7.4	RSBAC	57
7.5	SELinux	59
8	SELinux in Detail	61
8.1	Architecture and Functionality	61
8.2	Type Enforcement	64
8.3	Defining Roles	66
8.4	Multilevel Security	67
8.5	Reference Policy	67
8.6	Policy Analysis	69

9	Outline for a Secure ITComp CDS Cluster with SELinux	71
9.1	Prototyping a SELinux Policy extension	71
9.2	Applying File Contexts in VLFS	74
9.2.1	Security Context Labeling in VLFS' Extended File Attributes	74
9.2.2	Specifying VLFS Security Contexts in the Policy	76
9.2.3	Specifying VLFS Security Contexts as Mount Option	78
9.3	Multi-Client Capability	78
9.4	Performance Influence	79
9.5	Future Work	80
10	Conclusion	81
	Abbreviations	83
	List of Figures	85
	Bibliography	87

... once someone learns the root password who sympathizes with the ordinary users, he or she can tell the rest. The "wheel group" feature would make this impossible, and thus cement the power of the rulers. I'm on the side of the masses, not that of the rulers. If you are used to supporting the bosses and sysadmins in whatever they do, you might find this idea strange at first.

Richard M. Stallman, Gnu Core Utils

Applications derive their functionality and assurance from the strength of the underlying infrastructure; unfortunately, the dependability of that infrastructure has largely been ignored.

Dixie B. Baker, Fortresses Built Upon Sand

1 Introduction

1.1 Motivation

By the end of 2007, the companies IBM and Google announced an initiative¹ to encourage the development along the so called *cloud computing* paradigm.

¹ <http://www.nytimes.com/2007/10/08/technology/08cloud.html>

Cloud computing can be considered as a relative of the grid computing approach while also comprising aspects of service-oriented architecture (SOA), Software as a Service (SaaS), and hardware management and cost of ownership arising from utility computing [AFG⁺09]. The idea of cloud computing is to pool formerly separated computing infrastructure into major environments that support scalability on demand while hiding localization and implementation, and offer a variety of applications to a multitude of users or clients as an application service.

In June 2008, the information technology research firm Gartner published a report [HN08], declaring seven major security issues regarding the emerge of cloud computing, whereas the first two stated issues are *privileged user access* and *compliance*. Although, the report concentrates on a totally opaque scenario of computing, with unknown service providers and a priori unwarranted service qualities, it points out an issue, which already applies to less complicated environments.

Discretionary access controlled operating systems like Linux are unsuited to be used as a basis for service-oriented environments or within the scope of cloud computing, as it is not possible to partition or decompose the power of superuser privileges in a convenient way. This deficiency of possible decomposition is a threat from both potential privilege escalation and exploitation and hostile administrators. While the user side of such distributed applications can be secured and more strictly controlled through additional layers within and around the application, a similar approach regarding the activity of system administration is much more difficult. The reason for this is the need for privileged access, respectively superuser privileges, in the course of administration. Since discretionary access controlled superuser privileges can circumvent security layers by modification of system resources' ownership and their permissions, it is very difficult to limit a superuser's access. Although, only selective operations actually need privileged access during system administration, the coarse grained concept of the discretionary access controlled superuser represents only an all or nothing freedom of decision.

1.2 Objective of the Paper

The application Clustered Data Services (CDS) is a high-performance network-attached storage (NAS) application developed by the company ITCComp. Following a clustered, horizontal scaling architecture and along with the idea of cloud computing, ITCComp CDS is intended to centralize formerly separated technology into one massive system. The application's clustered central unit is based on a Red Hat Enterprise Linux version 5 and provides storage access by several network protocols as different services.

Subject matter of this thesis is the analysis of superuser privileges in the operating system Linux with respect to its utilization in the application ITCComp CDS. The goal is to identify the problems arising from superuser privileges and to discover and discuss potential methods of resolution and their capabilities. Finally, the thesis gives an outline, how and to what extent a solution can be achieved and realized.

1.3 Outline

Starting with the background and fundamentals of computer security and access control in Chapter 2, the thesis gives an overview of different access control models and attacks. Subsequently, in Chapter 3 the operating system Linux and its superuser *root* are presented and Linux' mechanisms regarding its utilization and scope of validity are described. Chapter 4 describes the architecture and design of the application ITCComp Clustered Data Services (CDS), its components and their interaction. Within Chapter 5 seven problems are developed and formulated concerning superuser privileges in Linux with respect to its appliance in the application CDS. Chapter 6 specifies the prerequisites and requirements for

a solution to the seven problems described in the previous Chapter. Beyond, the Chapter contains an exemplary abstract role design regarding the administration of CDS. During Chapter 7 four available security and access control extensions for the operating system Linux are presented and discussed with respect to the specified requirements and a potential utilization within CDS. Chapter 8 describes in detail the architecture and functionality of the Linux security extension SELinux and Chapter 9 gives an outline for a potential appliance of the extension within CDS, especially concerning the association of the CDS data storage. Finally, Chapter 10 presents the conclusion.

2 Background and Fundamentals

This chapter presents an overview of the background and fundamentals of security and access control in computer systems. The individual sections are not meant to discuss the particular topic exhaustively, but to supply the necessary foundations for this thesis in order to discuss superuser privileges in the scope of security.

2.1 Security Goals

This thesis is based on the security goals *confidentiality*, *integrity*, *availability*, *authenticity* and *accountability*, which are outlined as follows:

Confidentiality

Confidentiality represents the concept of carrying or transferring information while avoiding any unauthorized leakage, respectively keeping the information confidential or secret. Not providing any information may include even to keep

the simple existence of the information confidential or secret as well.

Integrity

Integrity states that an information or system is not modified in an unauthorized manner. Analogue, integrity concerns the detection and the prevention of its violation, whereas both aspects are assumed to be crucial.

Availability

An information or system needs to be protected from being retained or withheld. An example for a violation of a system's availability is a denial of service attack.

Authenticity

Assuring authenticity means to guarantee the genuineness of an object, such as to guarantee the identity of the senders in the course of a message exchange or the identity of the originator of a data set. Within access control an authentication verifies the identity of a subject based on a predefined procedure, like e.g. the presentation of a user identification and a corresponding password.

Accountability

A system needs to process and keep logging and audit records in a tamper-proof manner and provide authorized subjects access to these records.

2.2 Access Control

Access constitutes the distinction of a *subject* and an *object* and describes a relation between them. An object can be protected by supervision over its access, which is termed *access control*. Consequently, the extend of access a subject is provided with, is limited by access control. In order to be able to protect resources and

data, computer systems use access control mechanisms to decide which kind of accesses should be either allowed or denied.

A simple but very fast access control mechanism is a two dimensional matrix, stating which subjects may access which objects. Yet, matrices have certain drawbacks as e.g. their maintenance and storage. Another mechanism is an *Access Control List (ACL)*, which represents a list associated with the concerned object that includes all allowed accesses and the corresponding subjects.

Permissions in respect to access control within the scope of computer systems are verbalized in several ways, such as rights, privileges or permissions. Within the remainder of this document the term *rights* will be utilized in order to refer to concrete technical representations of certain permissions, such as e.g. the right to read a file. The term *privileges* will be employed whenever an abstract extent of permissions is discussed.

An important aspect while discussing access control concerns the distinction of different kind of accesses. Well known access rights are e.g. *read*, *write*, and *execute*. The strength of expression of an access control mechanisms depends on both the possible ways of describing subjects and objects and what kind of access rights are distinguished and classified. For instance, a system might distinguish the permission of writing to a file into either the operation of altering and possibly deleting existing content and the act of only adding new information. This would describe the differentiation of a real write and an append only access. Another special form of access rights regards their delegation, as a subject may have the permission to pass on rights or abstract sets of privileges to other subjects, which will be further discussed in Section 2.2.2.

A system or mechanism can allow all possible accesses by default and only constrain, control, or prohibit certain accesses. This behavior is called *default allow* and also referred as *blacklisting*, since all entities named in a list represent forbid-

den accesses. Correspondingly, the contrary behavior of only allowing by explicit designation is called *default deny* or *whitelisting*.

2.2.1 The Principle of Least Privilege

One rule or principle for designing security systems is the *principle of least privilege*, which delineates a guideline about the amount of privileges subjects should be granted with.

"The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task." [Bis03]

The definition of the principle of least privilege presumes a given access control method with a given strength of expression. The extent of effectiveness or gain of the principle in a concrete system depends fundamentally on that strength. The coarser the access control is implemented, the more access privileges a subject will be awarded by the principle, since *"the least"* is a statement that is specified relative to the utilized access control mechanism.

2.2.2 Discretionary Access Control

Discretionary access control (DAC) is based on the idea of *identity* of subjects and objects and *ownership* as a relation between them. The ownership entitles an *owner* to determine the access privileges regarding the owned object, respectively grant and revoke other subjects access to that object. Beyond, it may be

even allowed to hand on the ownership of an object to other subjects. Hence, DAC is also called identity-based access control [Bis03]. DAC can be described as a mechanism due to which an object's access privileges are at least partially determined by its owner. Accordingly, the owner of an object needs to foresee the possible consequences of her influence and actions regarding access control adjustments. This aspect highly influences both system and data security.

A discrepancy regarding DAC concerns the abstract concept of ownership, as the DAC ownership represents the intention to translate a social real world concept into the scope of computer systems. However, an operating system and the running software has the power and sovereignty over its resources and data. Thus, the abstraction of ownership can be only represented within this limits and a user can only access data and resources due to the utilization of the operating system and the running software.

2.2.3 Mandatory Access Control

In contrast to the concept of ownership and delegation an alternative is to instantiate an authority that rules over access in a mandatory manner. *Mandatory access control (MAC)* bases access decisions on a given system of rules, which is referred to as *access policy*. Neither a decision derived from the access policy nor the policy itself can be modified by any normal subject or entity in the concerned system and the derived decisions must be enforced. MAC mechanisms can also consider other concepts or mechanisms of access control. However, regarding genuine MAC implementations, the policy must always be evaluated within the process of decision making and the final decision must be dominated by the policy. For instance, a MAC system can support a concept of ownership and delegation, yet only within the boundaries of legality in regards to the access policy.

In order to maintain the access policy, it is possible to entitle a system entity with the privileges to access and modify the policy. Another approach is to enforce the system into a certain state in order to load a new policy and therefore ensure that the policy is immutable during normal system operation.

2.2.4 Role Based Access Control

Another approach in order to model extents of access privileges is called role based access control (RBAC) [FK92],[FCK95]. By insertion of a new level of abstraction, access privileges are not directly applied to user accounts, but to *roles*. The user's extent of access privileges is defined by the roles she is permitted to utilize. Depending on her designated actions, a user needs to switch her active role and therefore performs tasks with different sets of privileges, whereas a role can be utilized by several users. Users may be allowed to use several roles simultaneously, or they are restricted to only one active role at a time. Further, roles can reflect hierarchical relationships.

The concept of RBAC can be combined with either DAC or MAC mechanisms.

2.2.5 Superuser Privileges

Within this thesis the term *superuser* will be generally utilized in order to denote a user account within a computer system that is equipped with a superset of all available privileges. Thereby, this superset is made up of concrete privileges and the right to change ownership of all data and resources in matters of delegation.

2.2.6 Reference Monitor

Computer systems that are shared by several participants employ an entity that controls and enforces the access relationship between each user and its environment within the system. Such an entity, called the *reference monitor* [And72], can validate all references to programs and data, while it complies with the following properties.

- The reference validation mechanism is tamper proof.
- The reference validation mechanism is always be invoked.
- The mechanism is small enough to be tested (exhaustively if necessary).

2.2.7 Type Enforcement

Type Enforcement is an access control mechanism first described by Boebert et al. [BK85]. The underlying idea is to classify a system's subjects and objects along a set of equivalence classes, whereas classes of subjects are called *domains* and classes of objects are called *types*. All domains and types receive identifiers called *labels*. Legitimate accesses are defined by the association of domains and types or of domains among one another. Two tables called the *Domain Definition Table* and the *Domain Interaction Table* are utilized in order to model the associations, containing concrete access rights along the existing labels.

An extension of the Type Enforcement approach is the *Domain Type Enforcement* [BSS⁺95], at which the two tables are replaced by a high-level language that provides more flexibility in expressing associations. Further, the extension introduces the concept of *implicit typing*, the maintenance of type and domain labels

during system run-time. Whereas within the original Type Enforcement mechanism, it is necessary to maintain these labels in an auxiliary task. Along with implicit typing, subjects and object are determined based on system hierarchy and listed exceptions.

The Type Enforcement or the Domain Type Enforcement approach can be utilized in order to implement mandatory access control within a system, as the access control is based on enforceable rule sets.

2.2.8 Bell-LaPadula Model

A famous security model called the *Bell-LaPadula Model* [BL73, BL76] was designed in order to enable the utilization of different levels of confidentiality and is based on the following two properties:

Property 1 (no read-up): A subject may only read an object if its security level is as least as high as the objects security level and the access is discretionary allowed.

Property 2 (no write-down): A subject may only write to an object if its security level is at most as high as the objects security level and the access is discretionary allowed.

The simple Bell-LaPadula Model can be extended in order to function not only based on different levels, but as well on different categories or compartments. This combination of levels and categories presents a lattice, whereas the level

based properties above need to be adjusted in order to function as a dominance predicate within this lattice [Bis03].

2.3 Threats, Vulnerabilities and Attacks

The opponents of security or protection goals are *threats*, their corresponding *risks*, *attacks*, and the underlying *vulnerabilities*. A *threat* is a potential security breach with respect to the defined security goals [Bis03]. A particular threat corresponds to an underlying *vulnerability* of a system. A threat can be suppressed by control of the corresponding vulnerability [PP06]. An *attack* is the exploitation of a vulnerability. Yet, threats are not necessarily connected to attacks, as there are also threats like human or system errors and environmental dependencies. Finally, the *risk* is a combination of probability and impact of a certain threat.

A major goal in the scope of security is the control and consequently the determination and mitigation of vulnerabilities in order to reduce their corresponding threats.

2.3.1 Flawed and Malicious Software

Regarding computer security the term *flaw* with respect to software does not necessarily imply a piece of program code is not serving its purpose correctly. A flawed software might be working perfectly, but the flaw has not taken effect or may even never take effect. Software flaws may be not only errors, but as well gaps, holes or opportunities, which are exploitable for attacks and therefore

represent vulnerabilities.

The exploitation of software flaws or the attack on vulnerabilities are representatives of malicious software appliance. *Malicious code* or *malicious logic* is a piece of software, that violates a system's security policy [Bis03].

2.3.2 Privilege Escalation

While having access to a system either by normal authorization or due to an attack, *privilege escalation* describes the act of gaining additional privileges due to the exploitation of vulnerabilities [PP06]. Thereby, the gain of superuser privileges may be of special interest, as they represent the maximum of possible privileges (see Section 2.2.5).

2.3.3 Rootkits

A special kind of malicious software that utilizes privilege escalation is represented by so-called *rootkits*. Rootkits are based on the infiltration of malicious code into a computer system while hiding their existence as long as possible, respectively as long as the duration of a further agenda [Eck08]. The infiltration is accomplished due to privilege escalation, potentially combined with other attacks, whereas a major target is the gain of superuser privileges. Figure 2.1 shows possible ambushes for rootkits in operating systems. Whereas, the first two cases reflect a scenario, in which the rootkit undercuts the operating system, the remaining three cases each denote a different hide-out within the operating

system. Each of the different ambushes is associated with a foregoing privilege escalation in order to gain a sufficient amount of privileges to enter the corresponding location within the computer system.

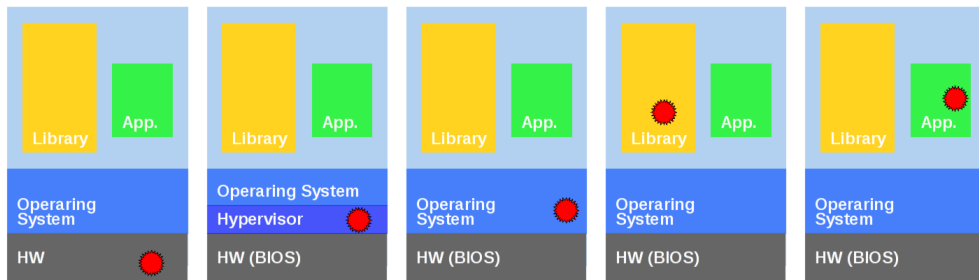


Figure 2.1: Rootkit Ambushes, Source: Modified [Tan08]

2.3.4 Zero-Day Attacks

Zero-day attacks are based on vulnerabilities that are not known or fixed yet. The term refers to the time frame between the moment an individual gains knowledge about the vulnerability and develops a feasible attack and the moment the attack is actually executed. Since a target has no knowledge about the threat, it has no time and opportunity for preparation or protection.

2.3.5 Covert Channels

Covert channels are a technique to overcome access control and security mechanisms by using shared system resources for communication [Bis03]. Typical

examples are statistical information about a system, such as memory or processor workload, an operating system's process list, or the meta information stored in a file system. By implementation of a protocol that was agreed upon before, such as a binary encoding based on existence and non-existence of a resource, entities or subjects are able to exchange information. The leakage of confidential information is only one example of the threat a covert channel can constitute, violations of communication protocols or circumvention of defense mechanisms are others. Covert channels are used for example, to infiltrate malicious code into a system in a covert or secret manner. Thus, covert channels are not only connected to attacks on confidentiality.

3 The Linux Operating System and the Superuser

This chapter gives a presentation of the relevant characteristics of the Linux operating system. A short introduction on the Linux architecture and its access control is followed by a description of the Linux superuser account *root*. The remainder of this chapter discusses available proprietary Linux mechanisms that concern the area of validity and power of the system's superuser and the potential decomposition of its privileges.

3.1 Linux Architecture

The Linux kernel follows a monolithic design, whereby the kernel is responsible for all core features of the operating system and organizes these features within one address space. By utilization of kernel modules it is possible to load and unload program code into the kernel during system run-time. However, the mod-

ules' program code resides also in the kernel's address space and is consequently part of the monolithic kernel layer called the *kernel space*. On top of the kernel space resides the system's *user space*, which is also known as the Linux *userland*. The user space consists of the operating system's libraries, applications and its multi-user environment. Figure 3.1 shows this layered Linux architecture.

In order to switch between the operating system's user and kernel space, Linux utilizes different mechanisms for each direction. The Linux kernel receives *system calls* from user mode through its *system call interface* and uses *kernel signals* to send information into user mode. In order to submit system calls without further restrictions, a user or subject needs to possess superuser privileges. Normal system users can only use a subset of the available system calls.

Other connections between the kernel and the user space are the virtual file

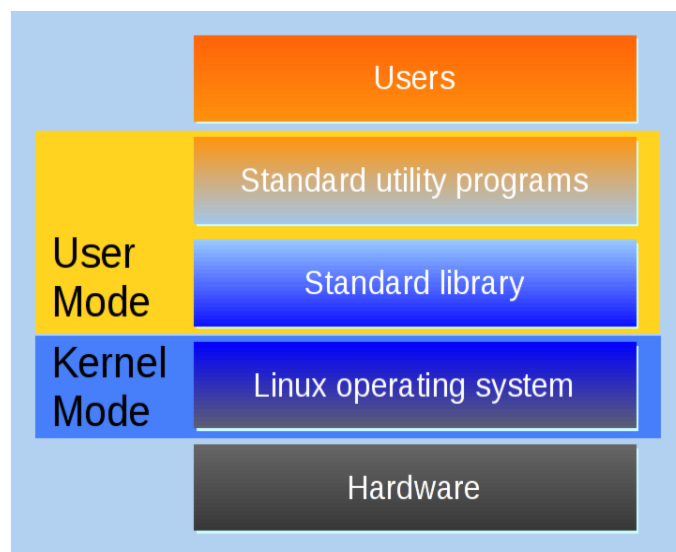


Figure 3.1: Linux System Layers, Source: modified [Tan08]

systems *procfs* and *sysfs*. Both file systems exist for special purposes and as an alternative to system calls, e.g. in order to configure kernel modules, read the system's process list, or trigger the kernels network behavior.

3.2 Access Control

Linux utilizes a DAC mechanism based on users and groups to organize and control accesses. A user has certain privileges based on her identity and her membership in groups. The file system distinguishes the three access rights *read*, *write*, and *execute*, which are organized as a triplet. Each object in the file system holds an attribute with three entries of these triplets, for each the owner, the owning group, and a third one concerning other subjects. This last entry represents the access privileges applying to anybody that is neither member of the owning group nor the owner herself. An objects owner can alter all of the three triplet entries and therefore delegate access to the object.

3.3 The Superuser

Within the operating system Linux, the superuser is the only user account that has the right to write the topmost entry in the file system hierarchy '/', which is called *root*. The file system mounted at this location is called the *root file system*. Consequently, the superuser account is named *root* likewise. This characteristic is derived from Linux's ancestry in the Unix world and can be found in a lot of other Unix operating systems as well.

The Linux root user is the first and only user account present in the system during start up and has the numeric user id '0'. During normal system operation it is the only user that can enter the kernel space without any restrictions. Accordingly, the root user owns all system resources at least through delegation. In other words, the root user possesses a superset of all user privileges in a Linux system. Running a genuine Linux system without further extensions or restrictions, the system and its user's data is completely at the mercy of the persons that have root privileges at their disposal. This is also the case if the root privileges where

acquired unauthorized as described in Section 2.3.3.

The following five sections briefly discuss four different mechanisms related to the validity and propagation of superuser privileges in a Linux operating system.

3.3.1 Setuid

Whenever a Linux program is executed, the program's process is started and running thereafter by utilization of the executor's user and group identifier. Consequently, the program receives the privileges the user and its group are entitled with.

Setuid is a bit attribute of files in Linux file systems, that triggers the operating system to execute a program with the privileges of the corresponding file owner. The *setgid* bit offers the analogue functionality for group privileges.

By utilization of this mechanism it is possible e.g. to allow unprivileged users to execute programs with superuser privileges. A standard utilization scenario of the *setuid* bit regards the program *passwd*, which enables users to change their system password. Obviously a normal user should not be able to edit the system's password file, but only to write the entry concerning her own password. By setting of the *setuid* bit attribute for the *passwd* program file owned by root, the program is not started with the caller's but rather with superuser privileges. Thereby, it is the responsibility of the *passwd* program not to enable a user to do anything more than to edit her own password.

3.3.2 Sudo - Substitute User Do

The Linux program *sudo* enables users to run programs by appliance of other user accounts, and consequently can be used in order to run programs with superuser privileges. A central system configuration file controls the usage of the program. The file contains information about which users can run what programs by utilization of which other user accounts and offers some elementary confining possibilities, such as e.g. shell parameter matching and time out specification. By appliance of *sudo* it is possible to allow listed users to run only certain programs. For instance, a user could be allowed to run the *passwd* program in order to change other users passwords, but exclude certain arguments respectively certain users by further specification in the configuration file [Ano01]. Thus, *sudo* is e.g. able to allow only the editing of specific files, yet it does not provide any methods to confine or control the editing process itself.

More precisely, the appliance of *sudo* has fundamental limits concerning the program execution after its foregoing control. If an editor provides the feature of system command execution, as for instance, *vim*¹ or *emacs*², a user limited by *sudo* to the utilization of this editor can use its command execution interface to send system commands to the operating system. *Sudo* will not control this behavior and the commands will be executed with the foreign privileges specified within the *sudo* configuration for editor usage only. Further, as the identification of programs and files is only possible by specification of file names in the configuration file, *sudo* is vulnerable to attacks on these. E.g. considering a user allowed to run a certain trivial program with superuser privileges by appliance of *sudo*. In order to get superuser access to the system, the user would only need to get the program file replaced by another file containing a shell code. *Sudo* would not

¹ <http://www.vim.org/>

² <http://www.gnu.org/software/emacs/>

realize this kind of fraud and consequently supply the user with a shell running with root's superuser privileges.

3.3.3 POSIX Capabilities

POSIX Capabilities describe a feature in order to split up superuser privileges in Unix operating systems. *POSIX* is an acronym for "*Portable Operating System Interface for Unix*", which is the name of a corresponding set of international standards for Unix operating systems. POSIX Capabilities were part of the Linux kernel since kernel version 2.2, yet the feature was limited to processes. In early 2008, file system capabilities were additionally introduced within the Linux kernel version 2.6.24.

The POSIX Capabilities describe a split up of superuser privileges operations into certain fragments. Currently, 32 capabilities can be applied to processes and executable files, that describe certain privileged tasks. These are e.g., to send a *kill* kernel signal to foreign processes, change system time, reboot the system, change of ownership, and rights of foreign owned files or to bind processes to privileged network ports. A process carries a set of capabilities it is permitted to use at most and another set defining which capabilities are used effectively. The latter can be changed by the process, yet needs to be a subset of the former. Whenever a process starts another new process, either its complete set of permitted capabilities or a freely specifiable subset of these capabilities is passed on.

The functionality of the POSIX Capabilities' file system feature is analogue to the *setuid* bit. An executable file can be equipped with capabilities while the information is stored in extended file attributes within the file system. Whenever the file is executed by a user, its running process has the specified capabilities.

Consequently, this can be seen as an extension or a partial replacement for the `setuid` functionality.

POSIX Capabilities offer a limited possibility to split up superuser privileges as their functionality only comprises a small and fixed set of hooks. Further, a process needs to possess all necessary capabilities while it is not possible to adapt a process' effective set of privileges during run-time, or to enforce a certain behavior. Analogue to the utilization of the `setuid` bit attribute, file system capabilities are based on the accessibility of the corresponding files and it is not possible to independently specify their users.

3.3.4 Chroot

The command `chroot` allows to change the system's root file system to a specified directory [red], whereas the Linux kernel gets attached to an additional new root file system. The specified directory needs to include at least a partial copy of a Linux installation functioning as a Linux root environment. The caller of the `chroot` command will end up in the new root file system environment, that is serving concurrently to its ancestor while sharing the Linux kernel.

Linux systems use this technique during their boot up procedure. Thereby, the Linux kernel uses a given initial ram disk file system as its very first root environment. After initializing the storage devices a device and partition that is specified by kernel parameter as the system's later root file system is mounted within the initial root file system. The `init` process executes a `chroot` to the location of the new root file system and initializes the user space.

`Chroot` is generally utilizable in order to separate entities, yet this comes by the cost of additional storage space and maintenance for the additional root file sys-

tems. Moreover, all potential root environments share the virtual file systems */proc* and */sys* and due to the sharing of the Linux kernel, the root users of all environments are identical at kernel level.

3.3.5 Conclusion

In its standard form, the Linux operating system is fundamentally based on the usage of superuser privileges. By utilization of *setuid*, *sudo*, or POSIX Capabilities it is possible to allow normal users to run programs by utilization of other user accounts including root. Thereby, *sudo* and POSIX Capabilities offer each a very limited functionality to further restrict this utilization. The Chroot mechanism in Linux is a possibility to separate entities in the same system to a certain extent. Yet, this separation does not apply to superuser privileges.

4 ITComp Clustered Data Services

This chapter describes the application ITComp Clustered Data Services (CDS), a network storage application that is designed and developed within the scope of cloud computing.

4.1 Introduction

CDS is a decentralized high-performance network-attached storage (NAS) application, based on a clustered architecture following the approach of horizontal scaling. Along with the idea of cloud computing, CDS is intended to centralize formerly separated technology into one massive system. More precisely, the goal is to integrate different storage applications into one system that is capable of providing these applications as different services.

The difference of CDS to classical clustered NAS approaches is the utilization of a single file system namespace that is apparent and accessible in parallel by all

cluster nodes. This is accomplished by appliance of the ITComp Very Large File System (VLFS) based on Storage Area Network (SAN) disk array storage devices. Currently, CDS supports only single-client installations, whereas the file system's access control is provided and maintained by an external directory service, such as Microsoft Active Directory [Dia02].

This thesis is based on the CDS version 1.2.3, which was released in March 2009.

4.2 ITComp Very Large File System

The ITComp Very Large File System (VLFS) is a parallel, clustered file system designed for high-performance demands such as supercomputers or large computing clusters. It allows to access clustered storage in parallel due to its own proprietary distributed locking, journaling and replication mechanisms. Whereas the concrete data and its meta data are stored together without appliance of a centralized meta data functionality in a SAN. A VLFS cluster is connected by a Local Area Network (LAN) based on the Internet Protocol (IP) and is able to maintain its own cluster management by dynamically designating nodes while still performing. Currently, IBM Advanced Interactive eXecutive (AIX), Microsoft Windows and Linux operating system environments are supported. Whereas this thesis concerns the application of VLFS in CDS, it will only consider a scenario based on Linux as a cluster environment.

The VLFS cluster approach used within the application CDS is based on a Linux environment while utilizing a shared-disc storage model in a SAN (Figure 4.1). Within the SAN, the VLFS cluster is connected with the storage units via Fibre Channel, whereas the storage units are hard disk arrays.

In a shared-disk model, all nodes in a cluster have equal access on all storage

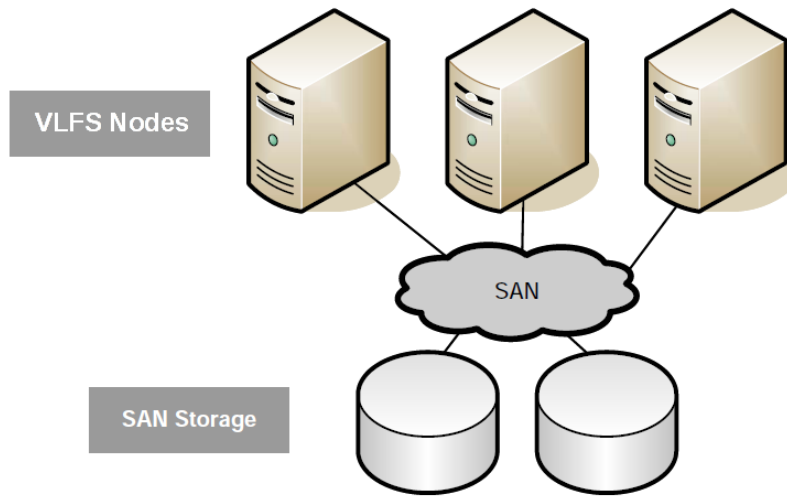


Figure 4.1: VLFS SAN Shared Disk Architecture

devices, thus each VLFS node has a full consistent view on the complete file systems the cluster nodes are holding within the SAN.

The VLFS software packages for Linux include a user space application with a server program for cluster control and several kernel modules in order to integrate the file system and its functionality into the Linux kernel. By that kernel integration, the VLFS file systems can be provided transparently over a virtual file system layer and are accessible in Linux as native POSIX compatible file systems.

4.3 Architecture

Conceptually, the ITComp CDS application cluster represents a redundant, high-performance NAS extension for a VLFS file system, accessible by different service protocols. Accordingly, CDS as a complete system describes the interconnec-

tion of Internet Protocol (IP) based LAN storage clients and SAN based storage devices. Figure 4.2 shows the interconnection of the different system entities around the CDS cluster. The application provides a NAS extension as a second

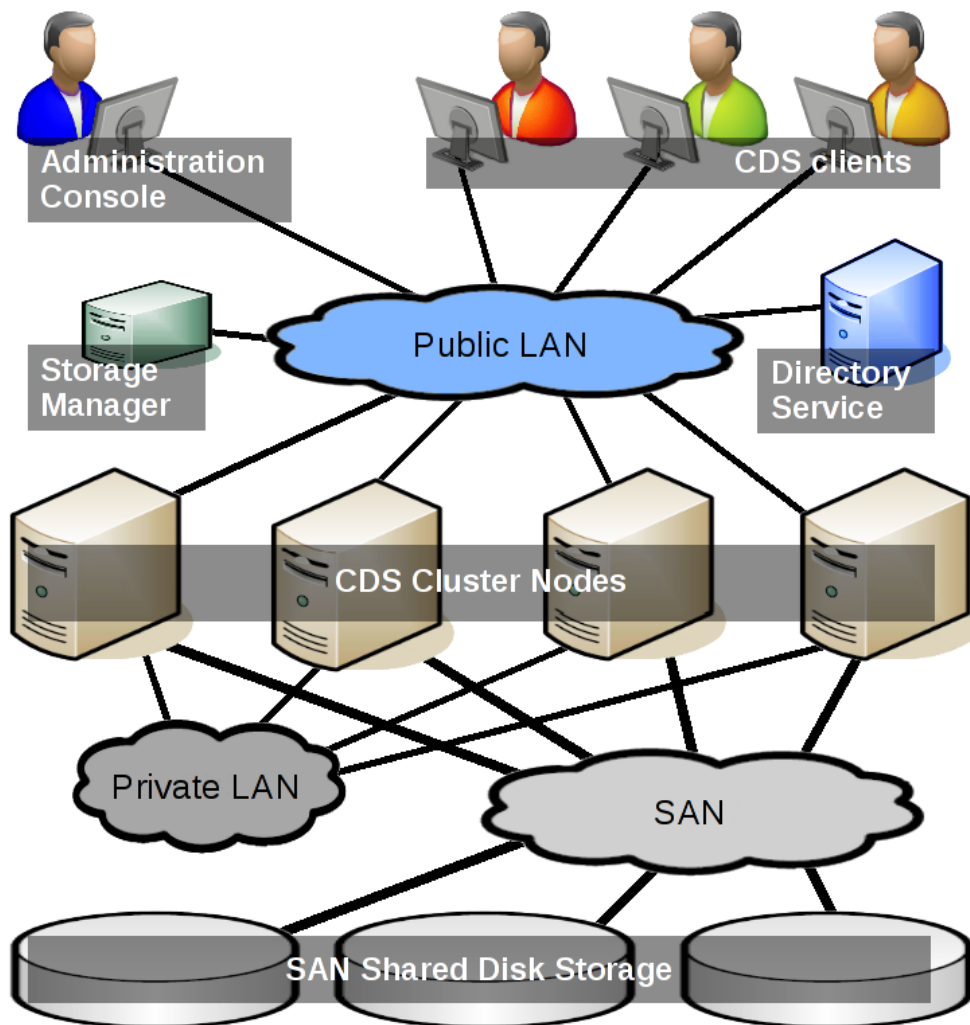


Figure 4.2: CDS Overview

clustered application, the Clustered Trivial Database (CTDB), on top of a VLFS file system version 3.2.1 (see Figure 4.3). The CTDB manages parallel and concurrent (see Section 4.3.1) accesses to the VLFS file system and is running next

to the VLFS application on all CDS cluster nodes. Further, all cluster nodes are based on a Red Hat Enterprise Linux (RHEL) version 5. Both clustered applica-

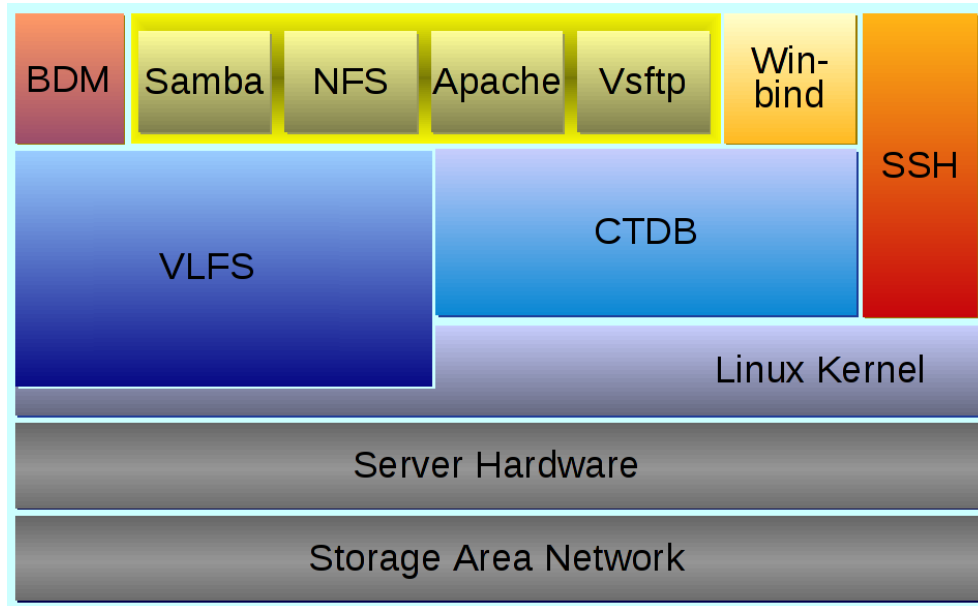


Figure 4.3: CDS Cluster Architecture

tions use a private LAN for communication, whereas a public LAN is utilized for the services. Each of the equal CDS cluster nodes provides access to the clusters VLFS file system by Common Internet File System (CIFS), Network File System (NFS), Hypertext Transfer Protocol Secure (HTTPS), and File Transfer Protocol (FTP) services. All nodes, respectively the CDS cluster, utilize one directory server for addressing and access control (see Section 4.3.3). Finally, each CDS cluster node is accessible by the ITCComp Backup Data Manager (BDM), a storage management application (see Section 4.3.4). Figure 4.3 gives an overview of all these components within a CDS cluster node.

4.3.1 Clustered Trivial Database

The Clustered Trivial Database (CTDB) is based on clustering efforts of the *Trivial Database* utilized by the unclustered NAS Linux application *Samba*¹. Along with the approach of Samba, the CTDB is holding a database in order to store meta data that is necessary to export POSIX file systems by CIFS, e.g. to Microsoft Windows based NAS clients. The CTDB application cluster manages the parallel and concurrent accesses on the VLFS file system by all available protocols (see Section 4.3.2). Beyond, it utilizes the VLFS file system to store information about its state within private files in order to coordinate the cluster.

Further, the CTDB is responsible for fail over and recovery actions and load balancing while performing. It enables the CDS cluster to change both device and cluster node infrastructure in a non-disruptive way. In case of a node failure, the CTDB is able to recover the lost connections by taking over the degraded node's IP address to one of the remaining nodes while delaying the services data transfer momentarily and therefore protecting the clients utility and data.

If nodes are reconnected or added to the cluster, the load is redistributed by using the same functionality, whereas the CDS cluster follows an all-active design. Consequently, the cluster utilizes all nodes in order to balance load and works without spares. This seamless integration of nodes allows the cluster to grow bit by bit along with necessity while supporting the ability to be dynamically adjusted on demand.

¹ <http://www.samba.org>

4.3.2 Services

Each CDS cluster node provides data access over the protocols CIFS, NFS, HTTPS, and FTP while utilizing the Directory Service in order to authenticate the accesses. Thereby, CDS is based on the existing Linux applications *Samba*, *NFS*, *Apache*, and *Vsftp* (see Figure 4.3). The applications are adjusted in order to be orchestrated with the CTDB and the VLFS applications inside each one of the CDS cluster nodes.

The application Samba provides the CIFS service and its appendant application *Winbind* supplies the corresponding connection of each CDS cluster node with the Directory Service. The NFS service is supplied by the corresponding Linux NFS server application, the Apache web server provides HTTPS accessibility and the Vsftp server is responsible for FTP.

4.3.3 Directory Service

Within the public LAN, a directory service is necessary in order to provide naming services and network authentication for the CDS cluster (see Figure 4.2). This service has to offer identity management, central authorization, and access control. These responsibilities comply with a Microsoft Active Directory service, which is expected to be utilized. Yet, it is possible to supply the required functionality otherwise, as e.g. by appliance of separate open source systems.

More precise, the CDS cluster requires a Domain Name Service (DNS) for name space custody and address resolution and a Lightweight Directory Access Protocol (LDAP) service in order to supply information about users and groups and to enable a centralized access control for CDS. Finally, a *Kerberos* [Bis03] Service

supplies network authentication and centralized authorization within the public network.

4.3.4 Backup and Archiving

CDS enables the appliance of a storage management application, the ITComp Backup Data Manager (BDM). The application comprises information life-cycle management features, as e.g. centralized and automated data archiving, and backup and recovery features, whereas it is possible to utilize both disk and tape storage devices.

The storage management is maintained within an external computer system and is connected to the CDS cluster by a designated program running on every node.

4.4 Administration

Within upcoming releases, CDS will employ an administration setting with a separated standalone administration console (see Figure 4.2) that will communicate via the Secure Shell (SSH) protocol with the CDS cluster. This administration console will offer a graphical user interface over HTTPS and a command-line interface over SSH. The command-line interface will use a proprietary shell that accepts only a limited, predefined command set and consequently prevents an administrator from arbitrary command execution, which normal Linux shells would offer. Both the commands derived from the graphical user interface and the ones read from the command-line interface will be translated by the console into standard Linux commands. These commands will be verified by syntax and elemen-

tary semantic checks and consequently submitted via SSH to a CDS cluster node, where they will be executed by the root user account. This utilization of the Linux root account should be avoided as outlined within the following chapters.

Currently, CDS utilizes a cluster node in order to provide an administration interface by running a service application as a normal Linux process. The administration interface is communicating internally via SSH by utilization of the Linux root account and is accessible by graphical user interface and command-line interface as described above.

This scenario is considered to be vulnerable, as e.g. failures or attacks on the administration interface could interfere directly with the cluster responsibilities and will not be further discussed.

4.5 Signed Software Updates

In order to impede the infiltration of malicious or flawed software and to guarantee the authenticity of software during install and updates of CDS, ITComp supplies digitally signed software updates by appliance of the Yellow dog Updater Modified (YUM) [red]. YUM is the package-management utility included and utilized in the Red Hat Enterprise Linux Distribution. ITComp maintains its own YUM repository, which consists of both update packages provided by Red Hat and proprietary ITComp CDS packages. The integrity and authenticity of the packages is assured by GNU Privacy Guard (GPG) public key certificates, whereas the YUM package-management is provided with certificates during system installation and invokes a verification of each package that is about to be installed.

4.6 Multi-Client Capability

Along with the approach of Cloud Computing, multi-client capability is a feature headed to be realized within future versions of CDS. Currently, it is not yet possible to separate different clients within the same CDS installation while giving each client a partial system view with sovereignty over its file systems and the file system's access control.

Even if this feature is not even specified in detail, this thesis will account for a generic multi-client capability, whereas two aspects are adopted as a general abstract requirement. At first, the Linux operating system as the basis of the CDS cluster nodes needs to be capable of compartmentation. More precisely, it needs to be possible to partition the CDS storage service into areas that are protected and shielded against mutual accesses, as e.g. by user groups or domains provided by the directory server. Secondly, in order to enable an independent administration of each storage compartment or area, privileges for administration need to be partitioned or decomposed as well along the layout of storage compartments.

When I was a kid, one day I realized mysteriously markings on the car keys of my grandfather's old light mint Audi. I retrieved a chair and grabbed all the keys of the old wooden key board at the wall. I encountered two identical car keys wearing different markings. Instantly, I questioned my grandfather about this dramatic finding. He explained me diligently one key would be an exclusive ignition key that could neither open the glove locker nor the trunk. As far as I can tell my grandfather never utilized a valet parking service, yet his car keys would have allowed him to do so.

5 The Problem of the Linux Superuser

This chapter formulates and describes seven problems arising from the existence of DAC superuser privileges in Linux.

The problems are derived and defined in the face of an ITComp CDS cluster based on a genuine Linux operating system. However, each of the problems is formulated as general as possible, since the underlying issues are assumed to apply to a multitude of usage scenarios, concerning operating systems based on the appliance of DAC super privileges in service oriented environments or within the scope of cloud computing. The general problems will be translated into concrete requirements within the subsequent chapter.

All problems are formulated while assuming a CDS cluster and its environment is performing normally and with integrity, and accordingly excluding the case of preceding foreign influence or attacks or a biased installation. Yet, persons entitled to utilize superuser privileges are not assumed to be necessarily candid or well-meaning, nor can human failure be excluded. Along with the idea of a hostile administrator and the principle of least privilege, the following problems are intended to reason why superuser privileges denote a fundamental problem to the Linux operating system.

5.1 Omnipotence

A system administrator that possesses superuser privileges in Linux can access any kind of information and data represented in the corresponding computer system. Furthermore, she has unconditional possibilities to compromise or sabotage the system and can delegate an arbitrary amount of its superuser privileges to any other user present in the system. Thus, no user can protect her data from access based on superuser privileges.

This level of access is way beyond the principle of least privileges concerning the administrator's original function of system maintenance. In the face of a hostile administrator it means a drastic system threat as the potential impact of successful attacks needs to be assumed as paramount.

Problem of Omnipotence: *The Linux superuser represents a super set of all privileges and has unconditional and illimitable power within the operating system. Consequently, it constitutes a major violation of the principle of least privilege and a drastic system threat in the face of hostile or careless administrators.*

5.2 Resistiveness

Superuser privileges own all entities in a Linux operating system at least by delegation. This is observable e.g. during boot time or in the single user mode, where *root* is the only user account present and available in the operating system. Whereas, this *root* account does not differ from the one present during other system modes, respectively the multi-user mode.

As superuser privileges are omnipresent and defined to be omnipotent, their validity spans even future introduced system entities and resources. Respectively, superuser privileges are not a fixed amount but rather the maximum of privileges by definition and are by definition always present in the system.

Problem of Resistiveness: *The omnipotence of superuser privileges in Linux is immutable and resistant within time.*

5.3 Unavoidable Obtainment

Since the menace of software flaw exploitation or privilege escalation cannot be controlled easily, superuser privileges represent a fundamental problem and a drastic threat for Linux systems beyond their delivery to authorized personnel. Also apart from hostile administrators, the impact of misuse of superuser privileges gained due to privilege escalation attacks needs to be estimated equally as paramount.

Problem of Unavoidable Obtainment: *As it is not possible to avoid potential illegitimate obtainment of superuser privileges due to attacks, their simple existence constitutes a fundamental drastic threat.*

5.4 Cover-up Assistance

Superuser privileges may be misused in order to hide, cover, or clean up attacks or malicious activities. Consulting the figure of speech "*opportunity makes the thief*", the opportunity may be both the fact of getting access to a property and the expected chance to run away after making an achievement without being caught.

An example of this facet is an administrator that even with a *least privilege* level of access may have possibilities that are highly exploitable in a malicious way. Yet, a difference to a case with possessing of superuser privileges may be the chances to hide the unauthorized actions and to get away successfully with the benefit. Another aspect concerns attacks that last a certain time period, whereas the action of hiding is an integral part of the attack itself, as e.g. rootkits.

Consequently, it is possible to distinguish two forms of covering up an attack. While on the one hand, the cover-up is crucial for the technical success of the attack, on the other hand, it is crucial with respect to the success of the attacker.

***Problem of Cover-up Assistance:** Superuser privileges enable to hide ongoing attacks, to clean up and cover tracks, and to wipe out evidence.*

5.5 Undermined Confidentiality

In otherwise fully encrypted or protected environments, Linux systems with superuser privileges, that are processing unencrypted data, dominate the overall level of confidentiality by their level of system security. More precisely, superuser privileges, in Linux systems working with unencrypted data, abandon any achievement made elsewhere in the environment through the use of encryption or protection, since their access regarding the unencrypted data is not confinable. Thus, the existence of superuser privileges in Linux undermines confidentiality.

Linux systems processing unencrypted data within subsystems like separated programs, compartments, or virtual hosts which share the system in a transparent or visible way are highly vulnerable to covert channel attacks. This situation can be exploited at least by utilization of superuser privileges. Even though it is not possible to eliminate any kind of covert channels in systems based on interference (see Section 2.3.5), superuser privileges can not only lead to the total loss over the control of a system, but as well to the total loss of data.

Problem of Undermined Confidentiality: Superuser privileges foil and undermine confidentiality gained in an environment from encryption or protection, if the corresponding system has access on decrypted data. Beyond, superuser privileges support or even enable the establishment of covert channels between subsystems that share a Linux system.

5.6 Unfeasible Compartmentation

In Linux systems that are hosting subsystems like e.g. chroot environments, superuser privileges apply unhindered to these subsystems. Regardless how they are obtained, superuser privileges in a Linux system can control, compromise or sabotage subsystems that share the same Linux kernel. Therefore, it is not possible to protect entities within a Linux system from superuser privileges, which is a direct consequence of their *omnipotence* (see Section 5.1).

Problem of Unfeasible Compartmentation: Linux superuser privileges cannot be divided or split up in order to be valid only within the borders of system compartments or subsystems. Thus, it is not possible to protect entities from superuser access within a Linux system.

5.7 Degraded Compliance

In the scope of distributed applications and computing, the potential enforcement of policies concerning e.g. data processing, security measures, or access control is a general functional requirement. As DAC based Linux systems cannot delimitate their superuser privileges, they are not able to comply with such requirements.

Problem of Degraded Compliance: *The existence of superuser privileges represents a hindrance to the introduction and enforcement of mandatory security or access control policies within the operating system Linux.*

6 Requirements to Secure the CDS Cluster

This chapter describes the fundamental prerequisites and requirements of the application CDS with respect to a security enhancement based on the problems outlined in Chapter 5.

After defining the security goals for the application CDS and its administration, a scenario is presented in order to subsequently derive requirements for a solution. Finally, the chapter outlines the specification of an exemplary role design concerning the administration of CDS.

6.1 Security Goals of CDS

Regarding the data stored and processed within the application CDS and its function as a data storage service, the general security goals are *confidentiality*, *integrity*, and *availability*. Moreover, concerning the application as a distributed service the communications between all application entities as well as all interactions of users are required to comply with *authenticity*.

Concerning the access and activity of system administration, the communication between the administrator and the CDS cluster nodes needs to conform with the goals *confidentiality*, *integrity*, *availability*, and *authenticity*. Moreover, the administrator needs to prove her *identity* during the process of authorization.

6.2 Prerequisites and Dependencies

In order to assess the requirements for a security enhanced CDS cluster with respect to superuser privileges, the application CDS is divided into three zones (see Figure 6.1), which are expected to comply with certain conditions and prerequisites.

The CDS cluster, its complete SAN including the storage devices, and the private LAN are assumed as a protected zone (highlighted in green in Figure 6.1) within the application CDS. A potential interference or attack in this zone is assumed to have paramount impact and undermines any control or security potentially supplied by CDS cluster. This concerns not only the SAN as the applications storage platform, but as well the private LAN as the clusters internal communication media. Consequently, the complete SAN including the storage devices and the private LAN are explicitly demanded to reside in a physically protected area (see Section 6.4.7).

The zone represented by the public LAN (highlighted in orange in Figure 6.1) is assumed as generally insecure and a priori open to attacks, as all its entities are directly accessible at least by all participants of the application CDS. Yet, with respect to the ongoing assessment the only considered attacking scenarios are based upon superuser privileges as represented by the seven problems described in Chapter 5. Accordingly, all communicating entities apart from the CDS cluster

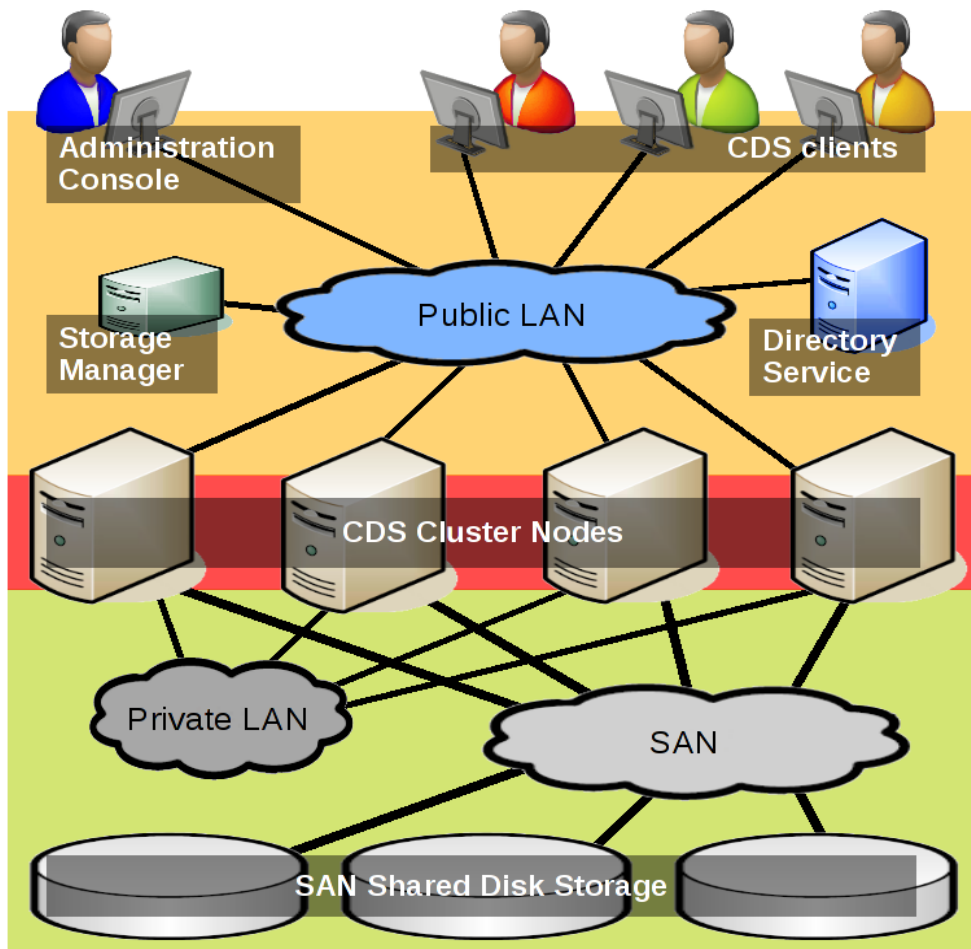


Figure 6.1: CDS Protection Scenario Overview

nodes are considered to be neither attacked nor biased and working properly by definition. Beyond, attacks within the network infrastructure of the public LAN are excluded from the analysis.

Moreover, concerning the activity of administration, the utilized SSH protocol and applications are assumed to generally suffice the defined security goals, as the communicating computer systems are *authentic*, the communication is en-

encrypted, respects both *confidentiality* and *integrity*, and the user is required to *authenticate* herself by knowledge of a password or the possession of a key.

6.3 CDS Cluster Requirements

In order to assess security mechanisms as potential solutions to the outlined problems in Chapter 5, the problems need to be translated first into criteria reflecting concrete requirements with respect to the underlying security goals. The criteria and their underlying problems are listed as follows:

1. Confining the threat of privilege escalations and attacks based on superuser privileges (*Problem of Unavoidable Obtainment*).
2. Providing the possibility of dis-activation or extinction of superuser privileges in Linux (Problem of Resistiveness).
3. Offering methods to decompose superuser privileges along a role design, as for instance the role design presented in Section 6.4 (*Problem of Omnipotence*).
4. Supplying methods to introduce and enforce mandatory access control policies that exceed the competencies of all users. The policies need to be both adaptable and analyzable (*Problem of Degraded Compliance*).
5. The introduced mandatory access control policies need to be analyzable regarding the information flow and covert channels within the system (*Problem of Undermined Confidentiality*).

6. Providing a tamper-proof logging facility that is capable of giving detailed information based on the specified access control policy (*Problem of Cover-up Assistance*).

7. Supplying methods within the mandatory access control policies to limit the access of entities regarding a general multi-client capability (*Problem of Unfeasible Compartmentation*).

Finally, a general criterion that is crucial, but will not be adopted as a functional requirement, concerns a potential influence on the performance of the CDS cluster.

6.4 A Role Design for Administration

The following seven sections present an exemplary role design in order to demonstrate a decomposition of superuser privileges with respect to the administrative responsibilities within CDS. Each role is specified in an abstract, enumerating its *function* or utility, its *view* on the system and the system's data, and its *privileges*. The abstract is followed by a brief description of each role.

The system needs to enforce users to have only one active role at a time. Thus, even if one administrator is allowed to use all roles, the effective level of privileges at a given time is never provided by more than one role.

6.4.1 Analysis and Reporting Role

- Function: Statistics, Logging, and Error Detection
- View: Abstract System Information and Meta data
- Privileges: System Information Processing and Error Submission

This role is responsible for the processing of system statistics, analysis of the system's state, logging maintenance, and error detection. Potential system errors are assumed to be relayed to the System Maintenance Role described in Section 6.4.5.

6.4.2 Access Management Role

- Function: Add, Delete, and Modify System Users
- View: User Meta Data
- Privileges: Access to User Management and Meta Data

This role is responsible for the maintenance of the directory service. As the NFS service utilizes its own access control mechanism based on fixed IP addresses for subject identification, the role has also privileges to trigger these mechanisms within the CDS cluster. Nevertheless, the role may only access the meta data concerning the user authorization and identification in both the directory service and the CDS cluster.

6.4.3 Service Management Role

- Function: Add, Delete, and Modify the System's Storage Configuration
- View: Exported Storage Resources and Configuration
- Privileges: Service Configuration and Exported Storage Layout

In order to manage the CDS cluster's service side, this role is associated with the internal CTDB application. The privileges comprise the control of the different services and the exported storage areas.

Regarding a future multi-client capability in CDS, this role along with the Access Management Role will exist for each client present in the system.

6.4.4 Resource Management Role

- Function: Add, Delete, and Modify the System's Storage Configuration
- View: Internal Storage Resources and Configuration
- Privileges: Modify Storage Configuration and Layout

This role is designed as the counterpart to the Service Management Role and is responsible for the configuration of the VLFS storage system.

6.4.5 System Maintenance Role

- Function: Error Handling and System Software Updates
- View: System Health and Software Status
- Privileges: System Interference through defined Procedures

In order to guarantee the integrity of the CDS cluster, only this role has the privileges to change the cluster's installation and to upgrade or update software packages. Further, the role has the capability of processing system errors by predefined procedures.

6.4.6 Data Management Role

- Function: Backup and Archiving
- View: Access on Data and Meta data
- Privileges: Data access

As the ITComp Backup Data Manager (BDM) has a direct access to the CDS cluster, this role has read and write access on files and directories. The access needs to be limited to the VLFS file system and the interconnection between the CDS cluster and the external BDM system.

6.4.7 Remaining Privileged Access

- Utility: Emergency Intervention
- View: Unconfined system view
- Privileges: Superuser privileges

The availability of a remaining privileged access is reasonable, as it is demanded and required in severe or fatal error, or in emergency situations. As the CDS cluster is exposed to physical access, a highly privileged or superuser access linked to physical access describes no additional threat. Consequently, a reasonable implementation of this access is the involvement of a physical intervention procedure, for instance, the usage of hardware tokens. Another additional or optional approach is the appliance of *secret sharing* [Buc04].

Before administrative intervention, the CDS cluster needs to designate a node to switch into a service mode and consequently remain only a passive member of the cluster. In this mode the node should be accessible by an exclusive interface, as e.g. a ssh server bound to a proprietary network port. After the administrative intervention, the node needs to switch back into normal operation and become again an active member of the cluster. Thereupon, the changes triggered by the administrative intervention become propagated to all other cluster nodes.

An approach that conditions the opportunity of superuser access solely to a local Linux terminal is assumed to be insecure and generally not recommended. As local terminals based on keyboard and screen interactions can be redirected into networks by special devices, this approach could be exploited to circumvent physical access.

7 Mechanisms to Confine the Linux Superuser

This chapter briefly describes four known Linux security extensions and analyzes informally their technical quality and capability with respect to the criteria described in Section 6.3. Beyond, these criteria applied as functional requirements, the analysis accounts also for each extensions development background, its adoption and insertion in the scope of existing Linux distributions and the availability of commercial support and warranty. The latter is of crucial significance, since two mechanisms involve to patch the Linux kernel source and consequently to build custom kernels.

ITComp CDS runs on a Red Hat Enterprise Linux and it is based on Red Hat's product support and warranty. As the utilization of a custom Linux kernel is not covered by Red Hat's support and warranty, this is a major non-functional drawback.

7.1 Linux Security Modules

Linux Security Modules (LSM) is a security framework that is part of the Linux Kernel since version 2.6. Its development was motivated by the demand of Linus Trovalds to enable Linux to utilize different security mechanisms and access control implementations and make it unnecessary to further change the kernel [WCS⁺02]. Due to the framework, it is possible to load security mechanisms as Linux kernel modules.

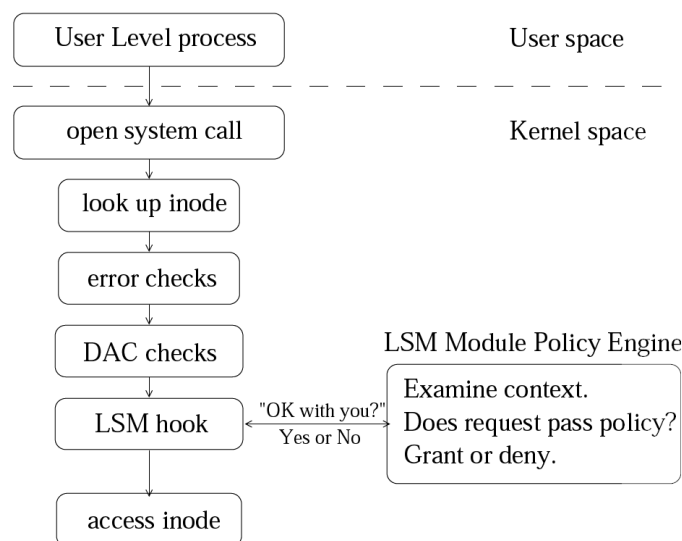


Figure 7.1: LSM Hook Architecture - Source: [WCS⁺02]

The framework provides an interface for a security mechanism to hook into the Linux access control decisions, whereas access requests are intercepted in order to include the additional access control logic introduced by the new security mechanism (see Figure 7.1). Thereby, the original Linux access decisions can be either combined with or overruled by the new mechanism.

7.2 AppArmor

AppArmor is a security extension that allows to protect certain processes in Linux. The extension is maintained and developed by the company Novell under the GNU General Public License (GPL). AppArmor is included in the Novell Linux distributions *SUSE Linux Enterprise Server (SLES)* and *SUSE Linux Enterprise Desktop (SLED)* and is covered by operating system's support and warranty.

The protection offered by AppArmor is reduced to a set of certain programs and consequently the mechanism separates the Linux system into confined and unconfined processes. The informal idea behind this approach is to secure a set of programs which are estimated to be highly vulnerable, such as web browsers or e-mail clients while minimally interfering with the user.

AppArmor utilizes the Linux Security Modules interface for intercepting access control decisions and genuine file names in order to identify files. Its rule set is represented by a policy, which is composed by a set of modules. The mechanism provides a given set of modules that can be configured and further adjusted by end users. Beyond, it is possible for users to assemble new modules by utilization of a graphical interface.

In the face of the criteria described in Section 6.3, AppArmor is completely unsuited to be utilized in order to make up a solution. It provides no possibility to enforce a general default deny directive and it is not equipped in order to protect or confine an entire Linux system. Hence, it is not possible to reason clearly about the extent of achievable security. Further, it provides no mechanism to confine the Linux root account. Finally, the identification of files by their file names in the face of AppArmor's limited protection scenario is highly questionable, since file names might be easily altered.

7.3 Grsecurity

The *Grsecurity* package is a composition of Linux kernel patches combined with a small set of control programs licensed under the GPL. The primal focus of the package is to harden certain known vulnerabilities in the Linux system while especially focusing on privilege escalation and root exploits. The project started in 2001, and according to the extensions website (<http://www.gresecurity.org>) there is only one contact person, which is also the author of all primary documentation. The extension includes an independent standalone set of patches named PaX, which supplies memory address space protection and randomization. *PaX* is also licensed under GPL and developed by an independent developer team, whereas the original author is anonymous.

Grsecurity provides a MAC mechanism based on ACL and RBAC capabilities combined with trusted path execution, that allows to limit the right of program executions to certain specified file names. The set of patches comprises protection mechanisms for file systems, executables and networks and additional kernel logging features. Grsecurity enables e.g. to harden the chroot environment against certain known attacks, prevent unprivileged users from reading kernel information and is able to limit and isolate the operating system's process view. Further, also anticipatory counteractive measures are possible, as e.g. to erase the TCP fingerprint of a Linux system in order to prevent attackers from gaining information about the running operating system in order to particularize an attack.

Informally, the concept of Grsecurity can be described as the intention to harden the Linux operating system and its proprietary mechanisms while restraining system entities like users and processes. Accordingly, the underlying idea is not only to place additional logic aside of the genuine Linux kernel, but also to alter the kernel's own mechanisms to comply with the desired behavior. Yet, Grsecurity does not follow any formal model of security, but emerged as a composition of

countermeasures against several known weaknesses, vulnerabilities, or concrete attacks.

Grsecurity has only little acceptance within available Linux distributions and it is not adopted by any major Linux distribution as a standard mechanism. Since the majority of configuration options needs to be done directly in the Linux kernel source, there are also no pre-built Linux kernels available. Consequently, the extension is not included in any commercial Linux distribution's support or warranty.

From a functional perspective, Grsecurity is not a convenient mechanism to meet the requirements represented in Section 6.3. It lacks a general systematic approach or model, which makes it difficult to meet predefined security goals. Many features can only be generally activated without further qualification or dependence on e.g. users, programs, or certain conditions. Further, the extension offers no mechanisms to analyze or reason about the achievement emerging from its application.

7.4 RSBAC

A Linux security extension called *Rule Set Based Access Control (RSBAC)* is based on an idea for a general security framework named the *Generalized Framework for Access Control (GFAC)* [ALEO90]. RSBAC was started out of a diploma thesis at the University of Hamburg and according to the project's website (<http://www.rsbac.org>) the development is still lead by its originator. The extension is licensed under the GPL and had its first stable release in the year 2000.

The underlying framework is integrated into the Linux kernel as a patch, in-

tercepts system calls to hook into access decisions during run-time, and provides an interface for registration of so called decision modules. It is possible to activate multiple modules simultaneously, whereas in that case, all modules have to approve on an access decision in order to lead to an actual access permission. Amongst others, the decision modules included in RSBAC provide a hardened user management, restrictable authorizations, advanced audit features and capability of pseudonymous logging, a jail module in order to confine programs, and a root-user protected ACL implementation. Furthermore, RSBAC comes with a module called *Role Compability* that follows the model described in [OFH01].

Within the Role Compability module it is possible to define roles as subjects associated with certain amounts of privileges. Users are entitled to use a specified set of roles, whereas they have default roles and can only use one role at a time. Objects of access, such as files, directories, inter process calls, or network devices are organized into groups of so called types. A rule set defines for each role which types may be accessed in what way analogue to the concept of ACL.

Like Grsecurity, RSBAC is not utilized as a standard mechanism by any major Linux distribution and it is necessary to alter the Linux kernel source and build a custom Linux kernel in order to utilize the extension. As mentioned before, this would have major consequences in the face of product support by e.g. Red Hat. Consequently, commercial support and warranty is only available from third-party companies.

By combination of the Role Compability and other available decision modules, the capabilities offered by RSBAC nearly suffice the requirements described in Section 6.3. However, RSBAC cannot provide a method to build a rule set in a centralized and integrated manner and there are no known tools or mechanisms to support the analysis of its policies. The extension provides guided configuration by setting of options according to standard use cases, such as service encapsulation. Yet, this imprecise concept of adjustment and the absence of analysis

tools for the rule set limit the extensions applicability in the face of a complex usage scenarios as the matter in hand.

7.5 SELinux

Security-Enhanced Linux (SELinux) is a security system that is strongly influenced by the U.S. National Security Agency, which released the code of its early versions in the year 2000 under the GPL. Whereas at first, the kernel source needed to be patched, SELinux could be used with the mainline Linux kernel due to utilization of the Linux Security Modules interface by the end of 2003.

In 2007, RHEL version 5 became certified [rhe07] under the Common Criteria for Information Technology Security Evaluation CC [com06], an international standard for computer security certification, by appliance of SELinux as a MAC and RBAC mechanism in Linux. The certification stated the operating system to be above the Evaluation Assurance Level 4 (EAL4), which is described as "*methodically designed, tested, and reviewed*" [com07].

Currently, the development is driven by the companies Tresys Technology and Red Hat. SELinux is today part of or available for the mainline distribution releases of Debian, Gentoo, Fedora, and Ubuntu, was recently introduced into SUSE Linux, and is a mature standard component of the Red Hat Enterprise Linux. Consequently, it is covered and actively maintained by the Red Hat product support and warranty.

The design of SELinux follows the idea of the *Flux Advanced Security Kernel (FLASK)* architecture that describes the flexibility of diverse security policies [SS⁺99]. Depending on the utilized policy, SELinux is able to follow a default

deny directive and consequently protect a complete Linux system, but also to limit the protection to only selected programs. The actual policies are developed independently, formulated in a policy language that was designed especially for SELinux, and are getting deployed as a compiled piece of source code. Due to its modularized policy layout, it is possible to independently change existing modules or to develop new modules based on an existing policy.

SELinux is an implementation of the Type Enforcement approach that supplies a MAC mechanism while combined with additional RBAC capabilities. The policy language contains both Type Enforcement and RBAC logic and beyond, supplies constraints that can be used in order to formulate assertions, whereas these constraints dominate the remaining parts of the security policy. Finally, SELinux policies can be analyzed concerning information flow and covert channels.

The capabilities of SELinux are considered to fully comply with the requirements delineated in Section 6.3. By appliance of SELinux it is possible, to confine the availability of the Linux superuser *root* to certain conditions. A Linux system is protectable as a whole and the policies can be enforced while overruling superuser privileges. Privilege escalations and attacks are limited along the tolerance of the employed policy, whereas the system supplies a logging facility that reflects potential violations. Moreover, it is possible to realize a role design due to the capabilities of RBAC. Finally, SELinux policies are analyzable regarding information flow and covert channels. Although, the SELinux policy is highly complex, its systematic and mature design enables to develop an access control rule set in a goal-oriented and methodical manner.

8 SELinux in Detail

This chapter describes the functionality and architecture of SELinux in detail and delineates its realization of Type Enforcement and the creation of roles. The MLS policy is discussed shortly, followed by a description of the SELinux Reference Policy and an outline of SELinux policy analysis.

8.1 Architecture and Functionality

Along with the FLASK architecture, the SELinux mechanism consists of two conceptual entities, the *Object Manager* and the *Security Server* (see Figure 8.1). Whereas, the Object Manager is responsible for the brokerage of access requests and the enforcement of access decisions, the Security Server has to make the actual decisions by utilization of a rule set called the *Security Policy*. Together these two components constitute a reference monitor.

Beyond that, SELinux introduces a cache component between the Object Manager and the Security Server called the *Access Vector Cache (AVC)*, which handles all communications between the two entities. During system run-time, the AVC stores an amount of derived access decisions that have been requested by the

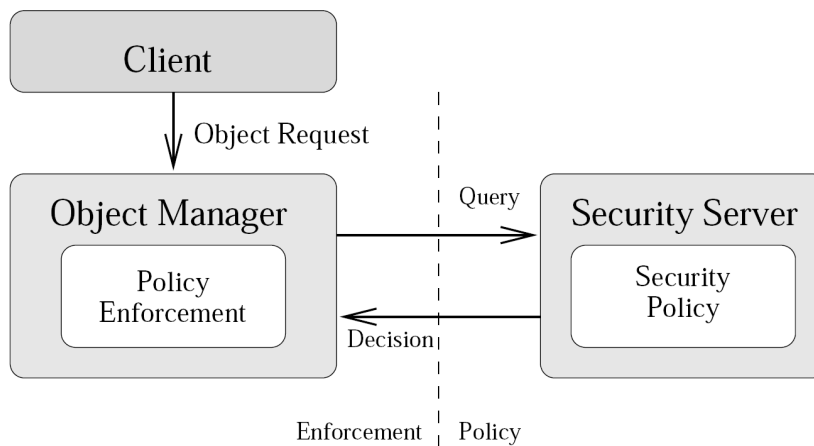


Figure 8.1: SELinux Architecture according to FLASK - Source: [SS⁺99]

Object Manager before. Due to the cache, recurring identical requests may be handled without deriving a decision anew within the Security Server component and accordingly accelerating the SELinux invocation as a whole.

SELinux offers several options concerning the logging of its access control activity. By declaration of rules it is possible to log all accesses requested by the Object Manager, only requests denied by the Security Server, or specific subsets of one of these cases. The log entries will be produced directly by the AVC and the log files along with all entities of SELinux within the Linux operating system are specially protected against any kinds of accesses, including privileged accesses.

Currently, SELinux supplies three different policy types, whereas each can be compiled and deployed as a monolithic or a modularized policy. Within the later, the policy is divided into a core unit called the *base module* and dynamically load and unload-able modules, which can be developed and deployed independently. The base module comprises the components responsible for the Linux core features while the modules on top represent the Linux user-space programs (see Section 8.5).

The three available standard policies for SELinux are all based on the approach of Type Enforcement while additionally supplying RBAC mechanisms. The policies are listed below:

Targeted Policy

The targeted policy confines only certain parts of the system. The fundamental idea is to protect programs and system entities that are assumed to be threatened in particular while minimally interfering with the usability of the Linux system. The policy can be seen as analogue to the concept of the AppArmor mechanism. However, an additional purpose is the enabling of a stepwise transition to an appliance of the *strict* policy.

Strict Policy

By utilization of the strict policy, SELinux activates a default deny behavior and consequently denies and blocks any access that is not explicitly allowed. In line with the extent of SELinux' granularity of control, this policy can deliver a complete MAC solution for Linux.

Multilevel Security (MLS) Policy

The MLS policy is an implementation of the Bell-LaPadula Model for multilevel security and extends the strict policy. By disabling the hierarchical leveling, the policy can be reduced in order to form a subtype, the so called *Multi Category Security (MCS)* policy.

The SELinux Type Enforcement utilizes, similar to the approach of Domain Type Enforcement, a high-level language rather than tables to model the association between types. In order to enable RBAC, SELinux policies contain user and role definitions. SELinux users are mapped onto standard Linux users either by an explicit or a default-user fallback definition in the policy. If neither an explicit nor a general classification can be applied, a user may not even log into the Linux

operating system. The SELinux user definitions include a set of permitted roles and both user and role definitions can include further policy rules that specify the corresponding extent of access.

In order to identify objects within the Linux operating system, SELinux utilizes additional identification attributes called the *security context*. A security context consists of the three attributes *user*, *role*, and *type*, plus additional information for levels and compartments in case of MLS or MCS policy usage (see Listing 8.1). Thereby, the user and role identifier are only considered for access control if an entity acts as a subject.

```
system_u : object_r : program_exec_t : s0:c0
```

Listing 8.1: A SELinux Security Context

The security contexts are either declared or defined within the policy or in case of file objects stored in a file's extended file attributes, if the corresponding file system is capable of this feature. E.g. the standard Linux file system *ext3* supplies extended file attributes and SELinux utilizes these by default. Consequently, the complete security context information for a Linux operating system is stored not only in the SELinux policy, but is spread as well over certain file system structures within the system. The consequences and side effects of the different security context locations on the VLFS file system within CDS are emphasized in detail in Section 9.2.

8.2 Type Enforcement

The Type Enforcement implementation of SELinux does not distinguish between *domains* and *types*, yet the phrases domain and type are utilized in order to dis-

tinguish types when applied to either subjects or objects. Consequently, within the remainder of this document the phrase *domain* depicts a type concerning a subject.

The first line in Listing 8.2 shows a type enforcement rule, giving *user_t* the permission concerning all *files* of the type *program_exec_t* to locate and execute them. A potential Security Context for a file object this rule would apply to is given in Listing 8.1.

A fundamental problem with respect to the Type Enforcement approach is the transition of domains, respectively how a domain may be entered or how a domain may change into another one if allowed and necessary. In order to enable standard Linux program code to be mapped into a Type Enforcement environment without making any changes necessary, the SELinux policy language supplies a method to automatically initiate domain transitions.

Listing 8.2 shows an example [MMC07] of a domain transition due to execution of a program file in SELinux.

```
allow user_t      program_exec_t: file    {getattr execute};
allow program_t  program_exec_t: file    entrypoint;
allow user_t      program_t: process    transition;

type_transition  user_t    program_exec_t: process    program_t;
```

Listing 8.2: SELinux Domain Transition

In order to start a program by a corresponding executable file, at first a user needs to be allowed to execute the file within the domain specified by the files type *program_exec_t*. Second, this domain needs to be allowed as an *entry-point* to the program's specified domain *program_t*. Third, the user needs to be entitled to make a *transition* into the program's domain. Finally, as Linux programs are neither aware of the domains they are associated with nor they possess any

methods to initiate domain transitions, this has to be specified within the SELinux policy. Consequently, a fourth rule specifies the initiation of an automatic domain transition whenever the program file is executed with its corresponding domain. Since SELinux knows only types, this is called the *type_transition*.

8.3 Defining Roles

Along with its RBAC capability, SELinux utilizes the concept of roles in order to decouple users from privileges, whereby both users and roles exist within the SELinux policy. A SELinux user is connected to a real Linux user by her identifier. Depending on the policy, a Linux user that has no explicit counterpart in SELinux is either rated as a predefined default user or may not log into the system. A SELinux user is only specified by the roles she is allowed to use, whereas she has one default role and needs to switch to another role manually.

A role is mapped to the Type Enforcement as a set of domains. Consequently, a role definition is an aggregation of permissions either formulated explicitly or by interfaces and templates. Listing 8.3 shows an exemplary role definition for a role *logreader_r*. The role receives a default unprivileged domain named *logreader_t* due to the template *userdom_unpriv_user_template*.

```
role logreader_r;

userdom_unpriv_user_template(logreader)

corecmd_exec_shell(logreader_t)

logging_read_audit_log(logreader_t)
```

Listing 8.3: SELinux Role Example

Thereafter, the role gets the permission to start a command shell and to read audit logs within its domain by the two consecutive interfaces.

8.4 Multilevel Security

The Multilevel Security Policy in SELinux is based on a compartment extended version of the Bell-LaPadula Model and utilizes two additional identifiers in the SELinux security context in order to associate entities with levels and categories. Listing 8.1 shows a security context with the MLS definition *s0:c0*, whereas *s0* defines the entities level and *c0* its category association. The identifier for levels and categories may be chosen arbitrarily. The model is expressed within the SELinux policy language by a special set of constrains that can be used aside of the Type Enforcement declarations.

8.5 Reference Policy

All three default policies of SELinux are based on the so called *Reference Policy*, that is currently developed by the company Tresys Technology [Tec]. The Reference Policy is source code of the corresponding SELinux policy language, which can be compiled into all of the tree standard policies (see Section 8.1).

The reference policy represents the source code for the MLS policy, plus the ability to shut down certain rules and to provide an optional mechanism to match unconfined accesses. While the latter is used only in order to provide the default allow behavior utilized within the targeted policy.

The SELinux Reference Policy is based on a modular design along with the ideas of encapsulation and abstraction. In order to provide the capability of complete access control, the Linux system including all programs is divided into responsibilities, which are mapped to a modularized policy design. All modules are arranged into one of six categories, which are *kernel*, *system*, *apps*, *services*, *admin*, and *roles*. A module represents a certain program or functionality in Linux. The policy may be compiled and deployed as a monolithic policy file or as a modular set of files. In the latter case, the policy consists of a *base module* that comprises all core responsibilities and dynamically loadable modules along the design of the modular Reference Policy source code.

Each module's source code is separated into three sections, regarding the modules *file contexts*, its *type declarations*, and its *interfaces*. The file contexts specify the file names and their corresponding security contexts, which will be applied during deployment (see Section 8.1). The type declarations comprise all rules regarding the module while all defined types and structures are private entities to the module, and consequently are not accessible outside the module. Only *interfaces* or *templates* defined within the *interfaces* section can supply mutual connections and accesses between the modules. Due to interfaces and templates it is possible to encapsulate the decision logic and interconnect the different modules in an abstract manner.

A further element of the SELinux policy language are *constraints*, which are part of the modules type declarations. By utilization of constraints, it is possible to make assurances such as e.g. the prohibition of a certain access up to more complex logical conditions based on users, roles, and types. Constrains are global restrictions and consequently dominate any other rules in the policy. Finally, the policy language supplies the utilization of Boolean variables to dynamically activate or deactivate rules within a module.

8.6 Policy Analysis

The SELinux Reference Policy is a well-defined, yet very complex structure. The release 20081014 as provided by Tresys Technology consists of about 170.000 lines of code and initially declares more than 2000 types organized in over 250 modules, which are connected by about 5000 interfaces and templates. Its module for the *Apache* web server, for instance, has about 2000 code lines and utilizes 22 initially declared types. Consequently, the development and analysis within the Reference Policy cannot be assumed to be trivial.

The Program *Apol* is a tool in order to analyze a compiled SELinux policy that is developed by Tresys Technology. Figure 8.2 shows a screenshot of the direct

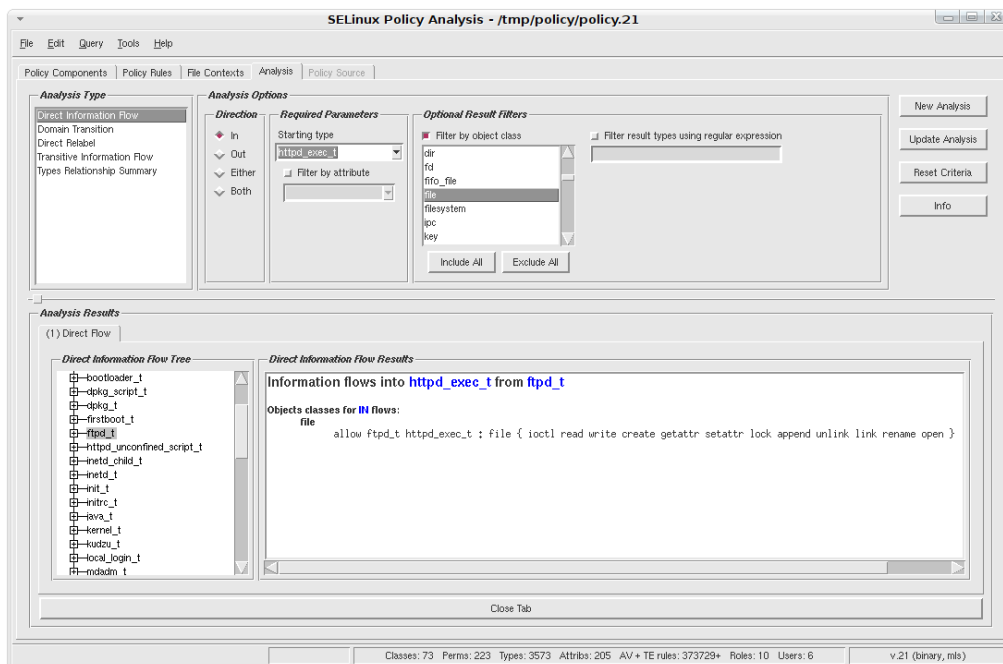


Figure 8.2: SELinux Policy Analysis with *Apol*

information flow analysis between two selected SELinux types. The program offers an overview of several policy language elements, as e.g. types, roles, users, and security file contexts and its attributes. It is possible to search and review policy rules based on different criteria, e.g. the kind of rule or its different attributes. Finally, Apol provides different analysis methods, for instance, to examine direct or transitive information flow between types or to inspect domain transitions.

9 Outline for a Secure ITComp CDS Cluster with SELinux

This chapter gives an introduction to a SELinux Policy advancement, concerning the CDS cluster. Different possibilities in order to adopt the VLFS file system in SELinux will be discussed. The utilization of SELinux will be further motivated due to the multi-client capability in CDS, followed by remarks on SELinux' influence on the performance of CDS.

9.1 Prototyping a SELinux Policy extension

The first step in order to develop a SELinux policy for the ITComp CDS cluster is to choose the kind of policy that should be utilized as the basis. The development of a complete policy from scratch is both unnecessary and unreasonable and will

therefore not be further discussed.

The following discussions and development descriptions regarding SELinux are based on ITComp CDS version 1.2.3, RHEL version 5.3, and the SELinux Reference Policy release 20081014 as provided by Tresys Technology. Further, the Multilevel Security Policy is assumed as the underlying policy concept, since the single Type Enforcement approach within the Strict Policy is not assumed to be enough (see Section 9.3).

The identification of responsibilities and analysis of already available modules in the Reference Policy leads to the following SELinux module design. The existing SELinux modules *samba*, *rpc*, *apache*, and *ftp* within the policy's *services* module section need to be modified or extended along with requirements concerning the control invocation of the CTDB (see Figure 9.1). Thereby, the module *rpc* is responsible for the NFS service, as this service's implementation in Linux is based on *Remote Procedure Calls (RPC)*. The Linux Program *Winbind*, which interconnects the CDS cluster with the Directory Service is included in the SELinux module *samba*. New modules need to be developed regarding the applications CTDB, VLFS, and BDM, whereas each should be encapsulated in its own module and introduced in the policy within the policy's *services* module section. Within the base module it is necessary to make additions at least in the modules *file system* and *corenetworks*, for instance, in order to introduce the VLFS file system and to register network ports for the different CDS applications.

The mutual control interference within the different programs in the CDS cluster needs to be translated into the policy by utilization of the SELinux policy method *interfaces*. Figure 9.1 shows an overview of the interface invocation reflecting the control interaction of the different applications in the CDS cluster. An arrow stands for the invocation of an access interface of another SELinux module. As e.g. the CTDB needs to be able to control the different services, its module needs to invoke certain interfaces in each service's module, which allows the respective

control. Another example is the VLFS module that needs to invoke interfaces in the base module, which allows VLFS to register its storage devices in the operating system.

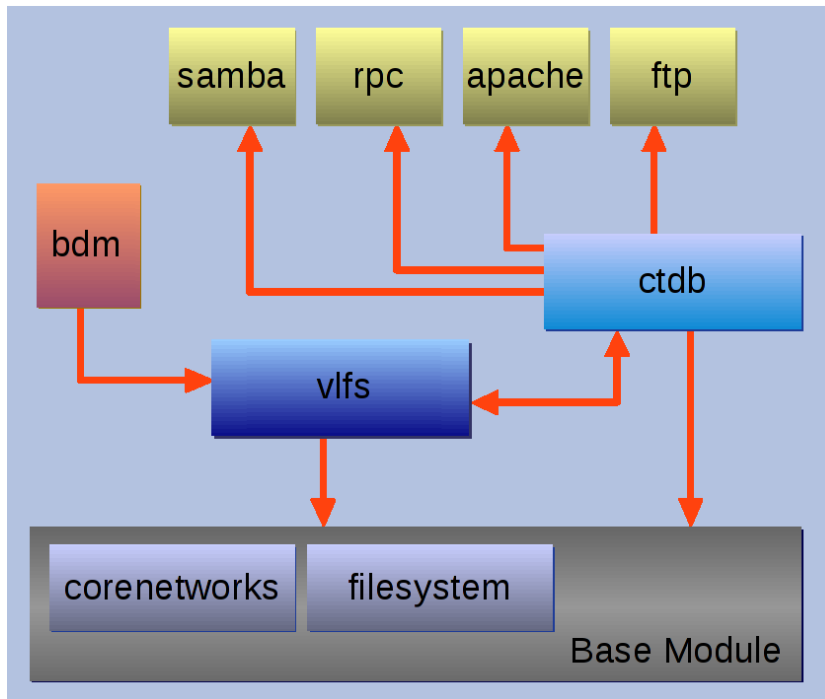


Figure 9.1: CDS with SELinux - Mutual Interface Invocation of Modules

A further step during the development of a SELinux policy for the CDS cluster is to exclude unused modules. As the Reference Policy is designed to comprise as much Linux programs as possible, it consists of several modules that are unnecessary for the CDS cluster. As each SELinux policy module corresponds to the permissions for a certain program, it is necessary to exclude all modules of unutilized programs in order to avoid potential access gaps.

9.2 Applying File Contexts in VLFS

SELinux utilizes its concept of security contexts in order to connect files with its policy rule set. Once decided to utilize SELinux as a mechanism to secure the CDS cluster, a fundamental aspect is how the data inside VLFS is identified by SELinux and accordingly how VLFS is connected to the SELinux policy. This section gives an overview of the three possible approaches while highlighting their differences and consequences. While the first approach identifies any file system based on its identity, the second approach is able to recognize a file system based on its type while ignoring if more than one file system of the given type is present in the operating system. The third approach represents a reduced, trivial version of the second one.

9.2.1 Security Context Labeling in VLFS' Extended File Attributes

As VLFS provides extended file attributes, its specification in the policy as a file system capable of native SELinux security context labeling is both feasible and straightforward.

Listing 9.1 shows how VLFS gets specified as a file system capable of extended file attributes by the `fs_use_xattr` macro. A type that was created and defined as applicable for file systems is utilized as a default security context for all instances of VLFS file systems present in the operating system. This specification must be done in the `file system` module, which is an inherent part of the policies base module.

```
type vlfs_fs_t;
fs_type(vlfs_fs_t)

fs_use_xattr    vlfs    gen_context(system_u:object_r:vlfs_fs_t,s0);
```

Listing 9.1: kernel/file system.te

The actual security context labels can be applied in the *VLFS* SELinux module. Listing 9.2 shows how two types of files are declared while Listing 9.3 shows how they are applied recursively to actual directories based on names and location relative to the operating system's root file system. Consequently, it is possible to separate data of the CDS shares and the meta data needed by the CDS cluster during run-time.

```
type vlfs_cluster_t;
files_type(vlfs_cluster_t)

type vlfs_data_t;
files_type(vlfs_data_t)
```

Listing 9.2: services/vlfs.te

```
/vlfs(/.*)?    gen_context(system_u:object_r:vlfs_cluster_t,s0)

/vlfs/share(/.*)?    gen_context(system_u:object_r:vlfs_data_t,s0)
```

Listing 9.3: services/vlfs.fc

During deployment of a later policy or policy module, SELinux is labeling all files and directories based on file context specifications, like the ones in Listing 9.3. Beyond, a deployed policy holds a central registry of all file context specifications. Recapitulating, at first SELinux utilizes an auxiliary list of file names and according file context specifications in order to apply these specifications to each files extended attributes. These actual security contexts are read out and utilized later during system run-time.

SELinux verifies and restores all files' security context labels in their extended attributes during system start-up, whereas VLFS is not present in that system state, yet. Since SELinux is not aware of VLFS as an external file system, the labels need to be manually verified by utilization of the appropriate SELinux programs, after each policy modification. Moreover, regarding data that is brought into the CDS environment while installation, or data of CDS installations upgraded with SELinux, this means the complete data would need to be processed and provided manually with the appropriate file security context labels.

By storing the security contexts in the extended file attributes of VLFS each CDS cluster node's SELinux policy will read out and utilize the identical security context entry for each file or directory. Yet, as all CDS cluster nodes must utilize the exact same SELinux mechanism and policy, this has only limited significance. Thereby, the context information stored within VLFS denotes a crucial and vital element of the complete SELinux security architecture and potential failures will affect all CDS cluster nodes in a consistent way.

9.2.2 Specifying VLFS Security Contexts in the Policy

Another approach of identifying VLFS files and directories within SELinux is to store security context information within the policy based on VLFS' identity as a file system type, rather than on each VLFS file system's identity. Consequently, any mounted VLFS file system will be treated the same by SELinux. Listing 9.4 shows how a file system type is defined and applied for all VLFS file systems

by utilization of the so called *genfscon* macro. The Listing 9.4 represents the corresponding layout to the alternative approaches' Listings 9.2 and 9.3.

```
type vlfs_cluster_t;  
fs_type(vlfs_cluster_t)  
  
type vlfs_data_t;  
fs_type(vlfs_data_t)  
  
genfscon vlfs /          gen_context(system_u:object_r:vlfs_cluster_t,s0)  
  
genfscon vlfs /share     gen_context(system_u:object_r:vlfs_fs_data_t,s0)
```

Listing 9.4: kernel/file system.te

As a side effect of this approach, SELinux identifies files based on their file names. Yet, it will not utilize their absolute location within the operating system's root file system (see Listing 9.3), but a partial location relative to the VLFS file system root. This special possibility exists within the SELinux policy e.g. in order to match the Linux *procfs* file system.

By utilization of the *genfscon* macro, the directory layout of the VLFS file system needs to be specified within the *file system* SELinux module, which is part of the base module. A change would result in an update of the complete policy. VLFS file systems could then be mounted at any location, whereas every VLFS file system would receive the exact same security context layout as specified. Since CDS is currently using only one VLFS file system, this can be disregarded. Yet, it may become a crucial concern, if several VLFS file systems with differing directory layouts shall be supported in a future version of CDS.

A further fundamental difference to the preceding approach (see Section 9.2.1) is the existence of the security context information independently of the concrete VLFS file system. As the context information is represented within the policy, the VLFS file system stays untouched. Each CDS cluster node holds the same context information within its own SELinux policy.

9.2.3 Specifying VLFS Security Contexts as Mount Option

A final possibility to introduce SELinux security contexts in VLFS is to specify them as a mount option (see Listing 9.5).

```
mount /dev/vlfs -t vlfs -o "fscontext=system_u:object_r:vlfs_fs_t,s0" /vlfs
```

Listing 9.5: Linux Mount Command

Basically, this option has similar characteristics as the usage of *genfscon*, yet supports only one security context type for the entire file system. Beyond, the information is not stored within the policy, but together with each client's VLFS management commands.

9.3 Multi-Client Capability

The SELinux multilevel security policy not only introduces the possibility to ensure certain information flows along its security model, but generally introduces two additional levels of abstraction. While within the sole model of Type Enforcement, programs such as the CTDB within CDS are based on one set of types representing the applications internals as an entity, Type Enforcement alone is not sufficient in order to separate clients within the same CTDB application process. Within the model of Type Enforcement a once running process cannot arbitrarily change its identity based on types. In order to separate equal program execution sequences within one and the same Linux process, such as serving two clients out of the same CTDB process, it is necessary to identify both the client's enti-

ties and their program execution sequences within the process. By utilizing the additional level of abstraction introduced with the multilevel security policy, entities and program execution sequences of potential clients could be identified by SELinux within the same processes as belonging to different categories and could therefore be protected from mutual access. Moreover, even the subset of the multi category security policy would be sufficient.

9.4 Performance Influence

As one key feature of CDS is its performance as a storage system, the influence of a security mechanism on the application's performance is considered the most important aspect after the functional requirements.

In 2001, Loscocco and Smalley presented SELinux benchmarks showing roughly about 10% performance overhead [LS01]. In 2006, Fedora stated the impact on system performance to be at about 7% while qualifying the actual impact to be *"heavily dependent on the tuning and usage of the system running SELinux."*¹ In 2007, Strand determined an average overhead of about 6%.²

Compared to a normal desktop operation, the SELinux use case represented by the CDS cluster can be assumed to be less scattered in terms of program utilization. The running applications within the CDS cluster during system run-time are nearly stable, whereas a typical desktop engine is subject to a constantly changing layout of processes, such as web browsers and office applications. Due to this effect, the efficiency of the access vector cache is expected to be beyond average cases.

¹ <http://docs.fedoraproject.org/selinux-faq-fc5/>

² <http://blog.gnism.org/article.php?story=RHEL5-SELinux-Benchmark>

The discussed approaches of labeling the VLFS data with SELinux security contexts (see Section 9.2) have different influences on performance. The two latter approaches, the generation of context information within the policy and the setting due to a mount option, only imply a marginal additional overhead within the computation of the SELinux policy and are therefore assumed to be negligible.

Regarding the approach of security context labeling by utilization of VLFS' extended file attributes, the file system readout is expected to introduce a very small I/O cost, as SELinux is requesting this information continuously during operation. Additionally, the approach leads to a minimal, yet additional storage utilization in order to store the security context labels in the VLFS file system.

9.5 Future Work

Beyond security context labeling of files, SELinux provides as well the possibility to label and control IP network packets. This offers the possibility, to spread the enforcement of a mandatory access control over heterogeneous, distributed system architectures. Regarding the CDS cluster, the labeling of network packets is not necessary, as all cluster nodes utilize an identical operating system installation and consequently an identical SELinux policy. Moreover, the VLFS file system is shared equally by all nodes in a transparent manner. Yet, the storage manager and the directory service are both running on different operating systems or at least different types of installations. Due to a synchronized IP network packet labeling, SELinux could be interconnected with other MAC mechanisms that are also capable of this feature. For instance, regarding compliance and multi-client capability, memberships of data or rules for data treatment could be preserved beyond the scope of the CDS cluster.

10 Conclusion

Discretionary access controlled operating systems like Linux are unsuited to be used as a basis for service-oriented environments or within the scope of cloud computing, as it is not possible to partition or decompose the power of superuser privileges in a convenient way. This deficiency of possible decomposition is not only a threat in the face of potential privilege escalation and exploitation or a hostile administrator, but is a major functional drawback.

Concerning the application ITComp Clustered Data Services (CDS) this thesis developed seven problem statements, expressing the concrete consequences of discretionary access controlled Linux superuser privileges for the applications clustered central unit, the CDS cluster. These problem statements concern the superuser privileges' *omnipotence, resistiveness, unavoidable obtainment, cover-up assistance, undermined confidentiality, unfeasible compartmentation, and degraded compliance.*

The problems were used in order to derive criteria for solutions with respect to the security goals of the application CDS and the activity of its administration. Analyzing four available Linux security mechanisms, SELinux stood out as the only mechanism that meets the criteria. Consequently, architecture and functionality of SELinux were analyzed and described in detail with respect to its potential application in CDS. Finally, the thesis gave an outline for the advancement

of the SELinux Reference Policy, in order to incorporate the proprietary software components of the ITComp CDS cluster. The incorporation of the ITComp Very Large File System (VLFS), as it is utilized within CDS as the underlying high-performance storage system, has been discussed in particular.

The Type Enforcement approach utilized by SELinux implements a mandatory access control that is able to protect entire Linux operating systems. Combined with its own role based access control model it is feasible to express even complex role designs. By employment of the system's multilevel security policy, it is possible to partition a system along a layout of levels and compartments and to decompose superuser privileges in accordance with that layout. Further, the SELinux policy is able to express and enforce compliance rules, as e.g. information flows, which can be analyzed and tested during policy development.

The threat of new emerging software vulnerabilities cannot be expected to decline in the future. A mandatory access control system like SELinux is able to avoid the exploitability of software flaws, restrain the applicability of malicious software, or at least contain the impact of attacks. Further, its fine-grained logging facilities empower a system in order to detect attacks and consequently impede attackers' possibilities to take advantage out of their actions.

The correctness of a SELinux based solution requires both a perfectly working SELinux mechanism and a policy that reflects the desired behavior in a correct and complete way. Beyond, neither the mechanism nor the policy may be flawed. Accordingly, an imperative requirement in order to protect the potential gain in terms of access control and security is the enforcement of authenticity and integrity of software, for instance, by utilization of digitally signed software. With special care about development and maintenance, Linux based systems like CDS can highly benefit from mandatory access control mechanisms such as SELinux.

Abbreviations

CDS Clustered Data Services

SELinux Security-Enhanced Linux

CIFS Common Internet File System

NFS Network File System

RPC Remote Procedure Call

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

VLFS Very Large File System

NAS Network Attached Storage

CTDB Clustered Trivial Database

BDM Backup Data Manager

ITComp Information Technology Company

IP Internet Protocol

LAN Local Area Network

SAN Storage Area Network

AIX Advanced Interactive eXecutive

SSH Secure Shell

AVC Access Vector Cache

YUM Yellow dog Updater, Modified

GNU GNU's Not Unix

GPL General Public License

POSIX Portable Operating System Interface for Unix

FLASK Flux Advanced Security Kernel

RSBAC Rule Set Based Access Control

MAC Mandatory Access Control

DAC Discretionary Access Control

LSM Linux Security Modules

ACL Access Control List

SOA Service Oriented Architecture

SaaS Software as a Service

List of Figures

2.1	Rootkit Ambushes, Source: Modified [Tan08]	15
3.1	Linux System Layers, Source: modified [Tan08]	18
4.1	VLFS SAN Shared Disk Architecture	27
4.2	CDS Overview	28
4.3	CDS Cluster Architecture	29
6.1	CDS Protection Scenario Overview	45
7.1	LSM Hook Architecture - Source: [WCS ⁺ 02]	54
8.1	SELinux Architecture according to FLASK - Source: [SS ⁺ 99]	62
8.2	SELinux Policy Analysis with <i>Apol</i>	69
9.1	CDS with SELinux - Mutual Interface Invocation of Modules	73

Bibliography

- [AFG⁺09] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Dept., University of California, Berkeley, CA, USA, Feb 2009.
- [Age] National Security Agency. *SELinux Mailing List*. <http://www.nsa.gov/research/selinux/list.shtml>.
- [ALEO90] M.D. Abrams, L.J. LaPadula, K.W. Eggers, and I. M. Olson. A Generalized Framework for Access Control: An Informal Description. In *Proceedings of the 13th NIST-NCSC National Computer Security Conference*, pages 135–143, Washington, DC, USA, October 1990.
- [And72] J.P. Anderson. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, ESD/AFSC, Hanscom AFB, Bedford, MA, USA, October 1972.
- [And01] R.J. Anderson. *Security Engineering*. Wiley, Chichester, UK, 2001.
- [Ano01] Anonymous. *Maximum Security*. Sams, Indianapolis, IN, USA, 3rd edition, 2001.
- [ASL01] R. J. Anderson, F. Stajano, and J. Lee. Security Policies. *Advances in Computers*, 55:186–237, 2001.

- [Bak96] D.B. Baker. Fortresses built upon sand. In *Proceedings of the workshop on New security paradigms*, pages 148–153, New York, NY, USA, 1996. ACM.
- [Bis03] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, Boston, MA, USA, 2003.
- [BK85] W. Boebert and R. Kain. A Practical Alternative to Hierarchical Integrity Policies. In *Proceedings of the 8th National Computer Security Conference*, Gaithersburg, MD, USA, October 1985.
- [BL73] D.E. Bell and L.J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report 2547, MITRE Corp., Bedford, MS, USA, 1973.
- [BL76] D.E. Bell and L.J. LaPadula. Secure Computer Systems: Unified Exposition and MULTICS Interpretation. Technical Report 2997, MITRE Corp., Bedford, MS, USA, 1976.
- [BSS⁺95] L. Badger, D.F. Sterne, D.L. Sherman, K.M. Walker, and S.A. Hahighat. Practical Domain and Type Enforcement for UNIX. pages 66–77, Washington, DC, USA, 1995. IEEE Computer Society.
- [Buc04] J. Buchmann. *Einführung in die Kryptographie*. Springer, Berlin, Germany, 3rd edition, 2004.
- [com06] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model, Version 3.1. Technical Report CCMB-2006-09-001, September 2006. <http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R1.pdf>.
- [com07] Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Components. Technical Report CCMB-2007-09-003, September 2007. <http://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R2.pdf>.

- [Dia02] J. Dias. A Guide to Microsoft Active Directory Design. Technical Report UCRL-MA-148650, U.S. Dept. of Energy, May 2002.
- [ea08] D. MacKenzie et al. *GNU Coreutils*. version 6.12, Free Software Foundation, May 2008.
- [Eck08] C. Eckert. *IT-Sicherheit. Konzepte - Verfahren - Protokolle*. Oldenbourg, München, Germany, 5th edition, 2008.
- [FCK95] D.F. Ferraiolo, J.A. Cugini, and D.R. Kuhn. Role-Based Access Control: Features and Motivation. In *Proceedings of the 11th Annual Computer Security Applications Conference*, pages 241–248, New Orleans, LA, USA, December 1995.
- [FGST04] M. Fox, J. Giordano, L. Stotler, and A. Thomas. *SELinux and grsecurity: A Case Study Comparing Linux Security Kernel Enhancements*. University of Virginia, Dept. of Computer Science, Charlottesville, VA, USA, 2004. <http://www.cs.virginia.edu/~jcg8f/GrsecuritySELinuxCaseStudy.pdf>.
- [FK92] D.F. Ferraiolo and D.R. Kuhn. Role-Based Access Control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, USA, October 1992.
- [Gol99] D. Gollmann. *Computer Security*. Wiley, New York, NY, USA, 1999.
- [HN08] J. Heiser and M. Nicolett. Assessing the Security Risks of Cloud Computing. Technical Report G00157782, Gartner, Inc., June 2008.
- [HRS⁺07] B. Hicks, S. Rueda, L. St.Clair, T. Jaeger, and P. McDaniel. A Logical Specification and Analysis for SELinux MLS Policy. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 91–100, New York, NY, USA, 2007. ACM.
- [JSS07] T. Jaeger, R. Sailer, and Y. Sreenivasan. Managing the Risk of Covert Information Flows in Virtual Machine Systems. In *Proceedings of the*

- 12th ACM symposium on Access control models and technologies*, pages 81–90, New York, NY, USA, 2007. ACM.
- [LS01] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42, Berkeley, CA, USA, 2001. USENIX Association.
- [LSM⁺98] P.A. Loscocco, S.D. Smalley, P.A. Muckelbauer, R.C. Taylor, S.J. Turner, and J.F. Farrell. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In *Proceedings of the 21st National Information Systems Security Conference*, pages 303–314, Crystal City, VA, USA, 1998.
- [MMC07] F. Mayer, K. MacMillan, and D. Caplan. *SELinux by Example*. Prentice Hall, Boston, MA, USA, 2007.
- [OFH01] A. Ott and S. Fischer-Hübner. The Rule Set Based Access Control (RSBAC) Framework for Linux. In *Karlstad University Studies*, Karlstad, Sweden, 2001. <http://www.cs.kau.se/~simone/rsbac-framework.pdf>.
- [PHS03] J. Pieprzyk, T. Hardjono, and J. Seberry. *Fundamentals of Computer Security*. Springer, Berlin, Germany, 2003.
- [PP06] C.P. Pfleeger and S.L. Pfleeger. *Security in Computing*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition, 2006.
- [red] *Red Hat Enterprise Linux Documentation*. <http://www.redhat.com/docs/manuals/enterprise/>.
- [RG91] D. Russell and Sr. G. T. Gangemi. *Computer security basics*. O’Reilly & Associates, Sebastopol, CA, USA, 1991.
- [rhe07] Common Criteria Evaluation and Validation Scheme, Validation Report, IBM Red Hat Enterprise Linux Version 5. Technical Report

- CCEVS-VR-07-0037, National Institute of Standards and Technology, National Security Agency, June 2007.
- [RKJ08] S. Rueda, D. King, and T. Jaeger. Verifying Compliance of Trusted Programs. In *Proceedings of the 17th conference on Security symposium*, pages 321–334, Berkeley, CA, USA, 2008. USENIX Association.
- [SGG09] A. Silberschatz, P.B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley, Hoboken, NJ, USA, 7th edition, 2009.
- [Spe08] R. Spenneberg. *SELinux & AppArmor*. Addison-Wesley, München, Germany, 2008.
- [SS⁺99] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The Flask Security Architecture: System Support for Diverse Security Policies. In *Proceedings of the 8th conference on USENIX Security Symposium*, pages 123–139, Berkeley, CA, USA, 1999. USENIX Association.
- [Sta09] W. Stallings. *Operating Systems, Internals and Design Principles*. Prentice Hall, Upper Saddle River, NJ, USA, 6th edition, 2009.
- [Tan08] A.S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, Upper Saddle River, NJ, USA, 3rd edition, 2008.
- [Tec] Tresys Technology. *Security Enhanced Linux Reference Policy*. <http://oss.tresys.com/docs/refpolicy/>.
- [WCS⁺02] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *Proceedings of the 11th USENIX Security Symposium*, pages 17–31, Berkeley, CA, USA, 2002. USENIX Association.
- [WKP05] S. Weber, P.A. Karger, and A. Paradkar. A Software Flaw Taxonomy: Aiming Tools At Security. *SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

- [WSB⁺96] K.M. Walker, D.F. Sterne, M.L. Badger, M.J. Petkac, and D.L. Sherman and K.A. Oostendorp. Confining Root Programs with Domain and Type Enforcement (DTE). In *Proceedings of the 6th conference on USENIX Security Symposium*, Berkeley, CA, USA, 1996. USENIX Association.