Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

A Secure Linux Desktop Environment

Master Thesis

2019
Bc. Zuzana Hromcová

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

A Secure Linux Desktop Environment

Master Thesis

Study programme:	Computer Science
Field of study:	2508 Computer Science
Department:	Department of Computer Science
Supervisor:	RNDr. Jaroslav Janáček, PhD.

Bratislava, 2019
Bc. Zuzana Hromcová

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Zuzana Hromcová
Study programme:	Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science, Informatics
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: A Secure Linux Desktop Environment

Annotation: The thesis deals with secure desktop environment in Linux OS using SELinux security mechanisms. Common PC usage scenarios are analysed, functional and security requirements are defined and a suitable solution is designed to fulfill them. The thesis includes an implementation of the proposed solution.

Aim:
- analyse common PC usage scenarios
- define functional and security requirements
- design and implement a solution
- analyse usability and security of the implemented solution

Supervisor:	RNDr. Jaroslav Janáček, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	13.12.2017

Approved: 13.12.2017 prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

Student

Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Zuzana Hromcová
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: A Secure Linux Desktop Environment
Bezpečné používateľské prostredie v OS Linux

Anotácia: Práca sa venuje problematike bezpečného používateľského prostredia v OS Linux s využitím mechanizmov SELinux.Analyzuje bežné scenáre použitia počítača z hľadiska funkčných a bezpečnostných požiadaviek a navrhuje vhodný spôsob ich naplnenia. Súčasťou práce je implementácia riešenia.

Cieľ:
- analyzovať bežné scenáre použitia PC
- definovať funkčné a bezpečnostné požiadavky
- navrhnúť a implementovať riešenie
- analyzovať použiteľnosť a bezpečnosť implementovaného riešenia

Vedúci: RNDr. Jaroslav Janáček, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 13.12.2017

Dátum schválenia: 13.12.2017 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Acknowledgements:
To my advisor, RNDr. Jaroslav Janáček, PhD., thank you for the opportunity to work on this compelling research topic. I enjoyed working on this thesis, as much as I enjoyed all of your lectures that I could attend in my academic life (and there were quite a lot of them). Thank you for your guidance, encouragement, and for the helpful discussions.

To my parents, thank you for your tremendous help throughout my studies. Thank you for supporting my big life decisions, but also for all those little things you do for me like waking me up on exam days, or picking me up from the train station on the rare occasions when I come to visit you. You have been my role models and I am proud to have followed your footsteps.

To Filip, thank you for introducing me to the world of computer security, and for helping me discover my future career. Thank you for being my source of inspiration and motivation for constant self-development, and thank you for your loving care and devotion that made finishing this thesis possible.

Abstract

The thesis deals with securing a Linux desktop environment using SELinux mechanisms.
The authors address the deficiencies of using SELinux in practice, and suggest an approach to extend the reference policy in a way that improves both the security and usability of a SELinux-protected system.
The authors first analyse common PC usage scenarios and relevant threats, in order to define functional and security requirements on a system. They design and implement a solution to fulfill these requirements, evaluate how the solution reflects these requirements and provide suggestions for future improvements.
The thesis includes manuals and support tools for deployment, administration and maintenance of the composed SELinux policy.

Keywords: SELinux, desktop environments, security

Abstrakt

Práca sa zaoberá bezpečným používateľským prostredím v OS Linux s využitím mechanizmov SELinux. Identifikuje existujúce riešenia tohto problému a uvádza ich nedostatky z hľadiska bezpečnosti a použiteľnosti.
Autori analyzujú bežné scenáre použitia počítača z hľadiska funkčných a bezpečnostných požiadaviek, a navrhujú vhodný spôsob naplnenia týchto požiadaviek, ktorý je zároveň kompatibilný s existujúcimi riešeniami a nástrojmi na podporu SELinuxu.
Súčasťou práce je implementácia riešenia a nástrojov na jeho administráciu, návod na používanie a ďalšie rozšírenie riešenia, rovnako ako ohodnotenie bezpečnosti a použiteľnosti systému zabezpečeného navrhovaným riešením.

Kľúčové slová: SELinux, používateľské prostredie, počítačová bezpečnosť

Contents

List of Figures

List of Tables

Introduction

Information security has always been needed everywhere some data were processed, even in the pre-electronic times. Naturally, its role and importance have evolved alongside the technology advances, and since the data are nowadays being processed electronically on heavily interconnected systems, the focus has shifted from assuring physical security to a wider range of security requirements. Data integrity, authenticity, confidentiality and availability must be enforced in the electronic world.

To achieve these, operating systems, web applications, and other systems protect the data against unauthorized access and modification, using access control mechanisms.

In the context of UNIX operating systems, traditional access control mechanisms only protected file system objects, and were under the discretion of their owner. This design has proved insufficient, as it comes with a number of inherent security weaknesses.

Possibilities to provide stronger security guarantees were in need: being able to protect other objects of the system, besides the filesystem objects; and able to define more fine-grained access control with more complicated policies. These were especially needed in environments with more users in the system, such as in large corporations, but also in single-user environments that were to be protected against malicious applications.

These requirements led to the introduction of a new approach, adding additional security mechanisms on top of the traditional access control systems. With the new mechanisms, all the operations are controlled by a global system policy, that the user has no control over. Several implementations emerged, SELinux being among the most prominent ones.

SELinux is an access control mechanism with a significant security potential. It confines all objects on the system - from the file system objects to network objects, including objects related to databases, inter-process communication or X-window server and clients. The granularity of the controlled operations is much more detailed when compared to the simple security mechanisms. That allows for complex security policies.

An official SELinux policy is provided and regularly updated by Linux distributions.

Due to its complexity, ordinary users are not expected to write their SELinux policies from scratch, so they use the ready-made policy to configure their systems.

This approach works for server setups with system applications and services that have their boundaries clearly defined. But when it comes to desktop environments, the users have varying security and functional requirements on the applications, and so it is a more daunting task to define such a policy that does not limit any of them. Therefore, security compromises are made in the official policy, in favour of usability of the SELinux-protected systems.

As a result, the SELinux policy only restricts system applications and services, while the user applications are virtually unlimited. On a different note, some Linux distributions do not support SELinux policy development in their desktop environments at all, and these systems become unusable with SELinux enabled. Either way, SELinux is not used up to its full potential in desktop environments - systems with SELinux enabled either lack security guarantees, or usability and flexibility.

The goal of this thesis is to address these issues. In this thesis, we extend the reference SELinux policy by further constraining privileges of user applications, in order to protect the system against common threat scenarios.

Instead of writing a detailed policy for a small number of specific applications, while leaving the others unlimited, we propose a more balanced approach. We design policies for a number of more general categories of applications, and provide a way for a security-concerned user for protecting sensitive data and limiting the damages caused by untrusted applications.

We also focus on usability and flexibility of the system with our policy enforced, and so we design our policy in a way that supports most common PC usage use case scenarios without denying legitimate user actions.

The first part of the thesis provides necessary background knowledge and motivation for our thesis. We assume the reader is already familiar with the basic concepts of computer security and access control mechanisms in Linux, and is willing to further explore the principles and possibilities of SELinux.

In Chapters 1 and 2, we introduce the concept of SELinux and briefly review its advantages over other access control mechanisms. We identify the problems of SELinux usability and security in practice. In Chapter 3, we present existing work that attempted to address them, and introduce our fundamentally different approach.

Part II of the thesis is dedicated to the analysis and design of our solution.

In Chapter 4, we identify common threat scenarios and associated security requirements of a Linux desktop system. In Chapter 5, we analyse functional requirements of commonly used applications, and outline the design of our solution.

Part III of the thesis deals with implementation of our SELinux policy.

Chapter 6 is an introduction to SELinux policy development. We explain the mechanisms of the development, and review the structure of the reference policy. In Chapter 7, we describe the implementation of our SELinux policy, we explain the issues we have faced and decisions we have made.

In Part IV, we explain how to use a system protected by our extended reference SELinux policy and how to make changes in the policy.

In Chapter 8, we provide a detailed tutorial on how to enable SELinux on system and how to configure it with our policy. We introduce tools for SELinux administration contributed by the SELinux community, and by the authors of this thesis. We list the most common issues related to SELinux and show how to solve them.

In Chapter 9, we explain how our SELinux policy can be further extended as a sequel to our work. We provide insight into a SELinux policy development, and present useful tools and practices. This chapter is dedicated to more experienced users, and can be skipped by readers who do not wish to further customize the policy.

Finally, Part V of the thesis deals with evaluation of our SELinux policy, in terms of usability and security.

Chapter 10 validates the policy against the usability and security requirements defined in Chapter 4 and Chapter 5, and assesses the overall security of the system protected by our SELinux policy. Finally, in Chapter 11, we identify shortcomings of our work, and suggest further improvements.

Part I

Introduction, motivation and goals

Chapter 1

Linux Access Control Mechanisms

The most prevalent type of access control in today's Linux environments is called *discretionary access control (DAC)*. As a main principle of DAC mechanisms, it is the object's owner who determines who can access the object, and how it can be manipulated. This design, however, comes with inherent security weaknesses[2], which led the security community to develop another type of access control - *mandatory access control (MAC)*. The role of MAC mechanisms is to overcome issues of DAC mechanisms and provide a higher level of security assurance.

SELinux belongs to the family of MAC mechanisms and can provide strong security safeguards, but it requires the users and administrators to deal with some additional complexity in using and managing their systems. To provide the context for understanding why it's worth the inconvenience, in this chapter we demonstrate the strengths and weaknesses of SELinux and other access control mechanisms.

1.1 DAC and capabilities

File system permissions are the primary DAC mechanism in Linux systems. Each file system object is assigned a set of permissions which define how its owner, members of its owners' group and other users can interact with it. Three permissions are specified for each of them - read, write and execute permission for files or list, add and use-in-path permission for directories[3].

The main shortcoming of using file system permissions is the poor granularity of how the users are categorized[1], which led to another mechanism being introduced to supplement them - *access control list (ACL)*. On top of assigning access permissions to the object's owner, group and other users, with ACL one can also define permissions for other specific users and user groups, and default permissions for newly created objects.

[1]Access permissions can only be defined for one particular user (owner), one group and all other users in general. For instance, there is no way to define distinct permissions for three different users.

Both file system permissions and access control lists can be changed by the object's owner and the superuser. This design comes with a number of security weaknesses. For one, the users (owners) can set overly permissive settings on their files. More importantly, DAC mechanisms cannot distinguish between computer users and programs, and so any program executed in the context of a user has the same rights to manipulate the access permissions of the user's objects as the original user. When such a program is faulty, vulnerable or simply malicious, the system security can be compromised.

On a different note, the traditional access rights models only distinguish between privileged and unprivileged processes. Many operations, such as device mounting or network configuration, are only allowed to processes running in the context of the root user. As a result, even if a program needs only a specific part of the superuser privileges, it is granted all of them, and thus it automatically gains an unlimited access to the system. This introduces another security vulnerability.

For example, a backup manager program requires administrator privileges to be able to access all the files on the system. When such a program is executed within the context of a root user, it also gains the privileges to change the configuration of the firewall, even though it does not need them. If there is an exploitable vulnerability in the program, or the intentions of the program authors are not innocent, this trust can be easily misused.

This problem is partially solved by another DAC mechanism called *capabilities*. In this model, the privileges traditionally associated with the superuser account are split into smaller groups, and a program is only assigned those permissions to the privileged operations that it needs to perform, following the least privilege principle.

Privileges (capabilities) of a program can only be changed by processes entitled to that particular privilege, which dramatically decreases the possibilities of rogue applications for misusing their privileges. However, other issues emerging from the nature of DAC mechanisms remain unresolved.

First, it is the users (or superuser) who are responsible for the configuration of the access privileges. Weak configuration can result from the lack of knowledge or interest in protecting the system.

Another issue is that neither DAC mechanisms, nor capabilities provide means to define (and enforce) more complicated security policies. They only protect a small part of the system resources, and do so with a coarse granularity. For all these reasons, these mechanisms can only work in environments where all programs are trustworthy and without faults. This is not applicable in the real-world scenarios where the software is flawed and will be flawed, so we need stronger security mechanisms.

1.2 MAC

Unlike DAC, with MAC mechanisms, access to resources is regulated by a system policy, which cannot be overridden by the users. These mechanisms provide some safeguards that can protect resources even when the users are careless. In Linux, they are brought to life using the Linux Security Module (LSM) framework, which opens the Linux kernel to new security extensions. The extensions can enforce a system policy by hooking various security checks provided by the framework[4].

Before LSM was introduced, security modules extending the functionality of the kernel had to be applied as kernel patches. With LSM, kernel hooks act like an interface that allows external security modules to influence the behaviour of the kernel[5].

The default LSM module built into the kernel is the capabilities module, but other security mechanisms can build their checks on top of the defined capability hooks and traditional access rights checks. LSM modules enforce additional restrictions on security-critical operations, and do not generate exceptions to otherwise denied operations. Thus, any operation is permitted only if all of the mechanisms allow it.

As of now, there are 4 MAC mechanisms using LSM framework, ranging in their user-friendliness and security potential: *AppArmor*, *SELinux*, *Smack* and *TOMOYO*[4]. The mechanisms address varying security goals, and protect access to different sets of system resources.

1.2.1 AppArmor

AppArmor implements a *task centered* policy, which means that access control attributes are bound to programs, rather than users. The rules constraining an application are defined in so-called *task profiles* for each application[6]. These can include capabilities, ability to manipulate specific files, to use network or to mount devices. The acccess control is based on file paths[7].

Programs that do not have a profile defined for them run in an unrestricted state which is equivalent to standard Linux DAC permissions. AppArmor is easy in principle and user-friendly[8]. It is the default LSM module in Ubuntu and OpenSUSE.

Default AppArmor profiles tailored to specific applications are provided by their developers or by the respective Linux distributions.

1.2.2 SELinux

SELinux uses a different paradigm than AppArmor - all operations are denied unless specifically allowed by the policy. AppArmor also applies deny-by-default principle, but only for a set of specified programs, not for all programs as is the case with SELinux.

Further, unlike AppArmor, SELinux is intended to protect the whole system. It extends set of the controlled objects from file system objects to all objects on the system (IPC objects, network, database objects...), with much more fine-grained access permissions.

For example, on top of the traditional read/write/execute file permissions, it can grant permissions to create, open, lock, rename or delete a file, read file's attributes, mount on the file and other.

The SELinux security policy controls access to objects based on their security contexts that include user identity, role, type and level, which allows to implement all main security paradigms in access control theory[8], such as Domain and Type Enforcement, Role-Based Access Control and Multi Level Security[2].

A general reference policy is provided by the SELinux team and adjusted by Linux distributions. Application developers and the security community also contribute to the policy.

SELinux is the default LSM module in Fedora.

1.2.3 TOMOYO

TOMOYO Linux is based on a philosophy different than that of AppArmor and SELinux. Instead of a ready-made security policy provided by the project team, with TOMOYO, the policy must be created from scratch, tailored to the needs of the user.

The behaviour of all applications is first recorded in the test environment, and then a policy is automatically generated for the specific system. In the production environment, all applications are then forced to act within these recorded behaviours[9].

1.2.4 SMACK

SMACK is an acronym for Simplified Mandatory Access Control Kernel, which reflects its main design goal - simplicity. To each object on the system, it attaches one of a small set of labels. MAC rules then define possible interactions between objects, based on these labels. Configuration data are minimal and not strictly required.

SMACK is similar to SELinux in the fact that access control is based on labels. However, SMACK only defines a small number of labels and rules, and only supports four types of access permissions (read, write, append, execute). In fact, it has been criticized for being written as a new LSM module, when a custom SELinux security policy could have provided equivalent functionality[10].

[2]See Sections 2.1.4, 2.1.5 and 2.1.7.

1.3 Summary

The goal of our thesis is to create a secure Linux desktop environment. The weaknesses inherent to the DAC mechanisms make them inappropriate to fulfill our goal, therefore we use MAC mechanisms in the implementation part of the thesis.

From the available MAC mechanisms, we chose SELinux[3] due to its extensive capabilities. Unlike AppArmor, it protects the whole system, not just a part of it[8]. Unlike SMACK, the access permissions are tailored per object type and are much more detailed than having the same four permissions for all objects. Finally, unlike TOMOYO, with SELinux, we are able to define a general policy applicable on a wide range of systems.

SELinux is less user-friendly than the other three, but it is one of the goals of the thesis to reduce the contrast.

[3] Our colleague, Bc. Peter Vašut, is addressing similar goals using AppArmor mechanism, in a parallel Master Thesis[11].

Chapter 2

SELinux overview

In this chapter, we introduce the concept of SELinux and demonstrate the added complexity and pitfalls of using it. We identify the lack of usability and security of SELinux in practice, when configured with the official policy; and analyse problems that prevent SELinux from being widely used to protect Linux desktop environments.

2.1 SELinux principles

SELinux (Security Enhanced Linux) is a mandatory access control mechanism implemented as a security module of the Linux kernel[12]. By default, all operations are prohibited by SELinux, unless they are explicitly allowed.

SELinux is not designed to prevent exploitation of software vulnerabilities (such as memory leaks or buffer over-runs), or to stop malware getting into the system, however, it may limit the damage or leaks they cause[1].

In the further sections, we explain the SELinux concepts relevant in the context of our thesis. For a more comprehensive SELinux reference, we encourage the reader to study *The SELinux Notebook*[1] or *SELinux User's and Administrator's Guide*[13].

2.1.1 Architecture

The SELinux architecture consists of four components, which are strictly separated: *object manager*, *security server*, *access vector cache* and *security policy* (with supporting configuration files).

The *object manager* component carries out accesses to protected objects. In our situation, it corresponds to the Linux kernel or X-Windows server. When a subject (user or process) requests access to an object, the object manager first consults the security server. The *security server* evaluates the authorisation of the subject to perform the operation, according to the security policy.

The *security policy* (or simply a *policy*) is a set of rules that determines which operations are allowed and which are not. The rules are written in a specific language, and then compiled to a binary form and loaded as a kernel module.

There are two types of SELinux policy - *monolithic* and *loadable module* policy. Monolithic policy is compiled from one source file to one fixed binary file. The loadable module infrastructure is more flexible - it allows policy to be managed on a modular basis. It consists of a base policy module that contains all the core components of the policy, and a number of modules that can be loaded and unloaded as required. For example, a policy module usually constrains one program. If the program is not installed on the system, then that module can be unloaded[1].

Further in this thesis, when referring to SELinux policies, we will always speak about loadable module policies, as this is currently the most prevalent SELinux policy type.

The security policy is usually not updated very often, so the same queries to the security server result in the same answers for the object manager. The fourth component of the SELinux architecture, an *access vector cache (AVC)*, helps benefit from this repetition. AVC stores the previous verdicts of the security server, and so in this updated model, the object manager first consults AVC with the query, and proceeds to the security server only if needed.

This mechanism is illustrated in Figure 2.1.

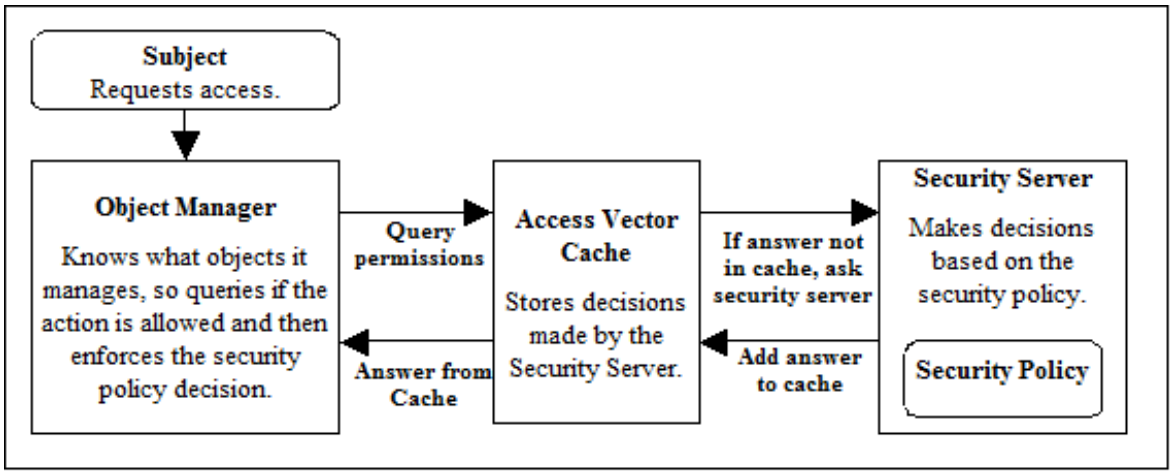

Figure 2.1: *High Level Core SELinux Components*[1]

The SELinux infrastructure can be either enabled or disabled on the system, and the system can either enforce the security policy or not. Based on these factors, we distinguish three major modes of operation[1]:

- **Disabled** - SELinux infrastructure is not enabled, therefore no policy can be loaded.

- **Permissive** - SELinux infrastracture is enabled, SELinux policy is loaded but the policy rules are not enforced. This mode is generally used for testing purposes.
- **Enforcing** - SELinux infrastracture is enabled, SELinux policy is loaded and enforced. This mode is used in production environments.

The most important part of the SELinux architecture from our point of view is the security policy and related configuration files, as these are the only SELinux components customizable by the administrator. Indeed, the access control concepts supported by SELinux that we explain in the remainder of this section, can be implemented precisely by adding specific rules and definitions into the SELinux policy.

2.1.2 Object classes and permissions

When talking about SELinux, we should start by spelling out which categories of system resources it protects. As you may recall from Section 1.1, Linux file system permissions only constrain user accesses to file system objects such as regular files, directories, named pipes, symbolic links and others. Read, write and execute permissions are controlled for each of these resources.

SELinux goes further. It provides means to control access to a broader set of objects, including network objects, IPC objects and database objects. The categories of resources (objects) are referred to as *object classes*, each being assigned a group of access vectors controlled by the policy, referred to as *permissions*.

The object classes and permissions are closely tied to the implementation details of Linux, particularly the kernel. In fact, they are designed to represent as accurately as possible the resources implemented by the system[2]. They are defined in the SELinux policy, but are generally not modified by the policy developers.

Following are some examples of the permissions defined for various object classes in Linux, illustrating the robustness of SELinux model. The meaning of the permissions is self-explanatory.

- **fifo_file (named pipes):**
  append, audit_access, create, execmod, execute, getattr, ioctl, link, lock, mounton, open, quotaon, read, relabelfrom, relabelto, rename, setattr, swapon, unlink, write;
- **netif (network interfaces):**
  tcp_recv, tcp_send, udp_recv, udp_send, rawip_rect, rawip_send, dccp_recv, dccp_send, ingress, egress;

- **msgq (IPC message queues):**
  associate, create, destroy, enqueue, getattr, read, setattr, unix_read, unix_write, write;

- **x_cursor (the cursor on the screen):**
  create, destroy, read, write, getattr, setattr, use.

The full list of the object classes permissions with their descriptions can be found on the SELinux project website[14].

2.1.3 Security contexts

SELinux differentiates between two types of entities on the system - *subjects* (processes) and *objects* (files, sockets, IPC channels, and so on). Subjects manipulate the objects, and the policy constrains the operations that a subject can perform with an object.

The access control is based on object class, and on the subject and object security attributes called *security context* (or *security label*). The security context is a tuple of identifiers assigned to each subject and object. It consists of three mandatory and one optional field - *user*, *role*, *type* and possibly a *level*. The level entry is included in the security context in some types of SELinux policy, as explained in Section 2.1.7.

The usual format of a security context is one of the following:

```
user:role:type, user:role:type:level
```

By convention, identifiers in the security context are appended descriptive suffixes to help better readability of the policy. For instance, `user_u`, `staff_r`, `init_t` would be acceptable user, role and type names, respectively.

All subjects and objects on the system are assigned security contexts. This assignment is computed via the security server, and is influenced by the SELinux policy, kernel and some user settings. When making access control decisions, SELinux security server then uses the subject and object security contexts, rather than other attributes (such as user of file names).

2.1.4 Domain and Type Enforcement (DTE)

The most important part of the security context is the SELinux type. Types of subjects are commonly referred to as *domains*, even though they are de facto a subset of SELinux types.

SELinux supports a type of MAC called *Domain and Type Enforcement (DTE)*, which is its central principle. DTE rules allow a subject to manipulate an object (e.g. to read a file), based on the subject domain and object type and class.

For instance, the policy may allow processes of domain `staff_t` to read files of type `user_home_dir_t` but not of type `shadow_t`.

The domains and types are central to access control decisions in SELinux, so another important part of the policy deals with assigning types (domains) to objects (subjects).

The policy specifies the default types for objects, default domains for users and the first process. These types and domains are generally not changed in the lifetime of the objects or subjects, and the child objects and processes usually inherit the parent's type. However, the policy can specify more complex inheritance rules and enforce a change of domain when a new process is spawned (and therefore a change in granted privileges), or a change of type when a new object is created:

- *domain transition:* when a process executes a program, the child process domain is determined by the parent process domain and type of executable file,
- *type transition:* the type of a new object (e.g. new file) is determined by the type of the parent object (e.g. directory) and domain of the creating process (e.g. `touch` process).

As an example of domain transition, we can look at a use case scenario where an unprivileged user tries to change password. We assume the user runs in an unprivileged `user_t` domain for which it is reasonable not to allow modifying sensitive files such as `/etc/shadow`. This privilege is, however, needed by the `passwd` program that the user executes to change the password. Assume that `passwd_t` is a domain with these privileges, and `passwd_exec_t` is the type of the executable file of passwd program.

We can solve this situation by defining a domain transition rule in the policy. When the user in `user_t` domain executes the `passwd` program with `passwd_exec_t` type, the new process transitions to the `passwd_t` domain that has the necessary privileges. As an effect, the privileges are not granted to the user permanently, only to the `passwd` process until it finishes.

2.1.5 Role-based Access Control (RBAC)

As we mentioned in Section 2.1.4, the access decisions are mostly made based on the subject domain. We also explained that different processes executed in the context of one user can very well run in different domains, each representing a particular set of privileges determined by the type of the underlying executable file. To further control access of users to these domains, SELinux applies another mechanism on the subjects - *Role-based Access Control (RBAC).*

Each Linux user is assigned a single SELinux user. SELinux user names generally correspond to groups or classes of users, rather than to Linux user identities; and are

never changed. For example, system programs could be assigned a SELinux user name of `system_u`, all the standard users `user_u` and administration staff `staff_u`[1].

Furthermore, a SELinux user is assigned a set of roles it can acquire. Only one of the roles is active at each moment, but the user can switch between them. Most importantly, each role is authorized for a set of domains which the processes of that user can enter. This mechanism is illustrated in Figure 2.2.

RBAC allows to restrict access to certain domains to specific roles, for example a user with the administrator role can execute `mount` command and automatically transition to a domain that is allowed to mount devices. But when a user with the unprivileged role executes the same command, the domain transition fails because the unprivileged role is not authorized for the domain.

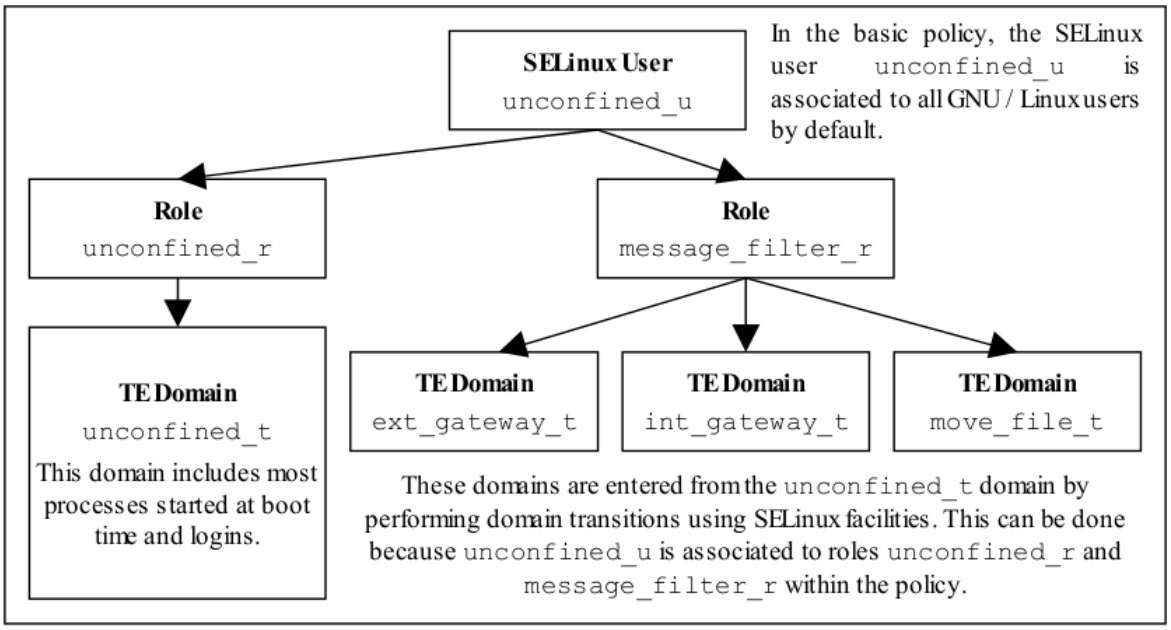

Figure 2.2: *Role Based Access Control in SELinux[1].*

The SELinux roles are of no meaning for objects - as a convention in the SELinux policies, all objects are assigned the same user role `object_r`, that is authorized for all object types.

2.1.6 Constraints

Another mechanism how to impose additional restrictions on DTE rules is called *constraints*. SELinux primarily uses subject domain to control access to objects, but constraints allow to take user or role names into account. For example, we can restrict the ability of a user to read private keys to cases where the user entry of the security context is the same for both the user and file with the private key. This would be possible in the setting with SELinux identities mapped to Linux identities 1:1 (i.e. when

the user identity could be retrieved from the security context), in order to provide user separation.

2.1.7 Multi-Level/Multi-Category Security (MLS/MCS)

The fourth, optional, *level* entry of the security context, is used by SELinux extensions called *Multi-Level Security (MLS)* and *Multi-Category Security (MCS)*.

MLS extension defines sets of hierarchically organized security sensitivities and categories. Both subjects and objects are then labeled with a security level, composed of a sensitivity and a category. The security level entails a subject's clearance or an object's classification[13].

With rules similar to those described in Section 2.1.6, it is then possible to further restrict access based on the classification of subjects and objects. We can implement access control policies to protect confidentiality (or integrity) of data, e.g. based on the Bell-La & Padula[15] and Biba[16] models, that were originally used for enforcing access control in government and military applications.

For example, we can define that a process running at a *confidential* level can read-/write at its current level but only read down levels or write up levels. This ensures confidentiality as the process can copy a file up to the *secret* level, but can never re-read that content unless the process "steps up to that level", also the process cannot write files to the lower levels as confidential information would then drift downwards.[1].

While still used in this way, the *level* entry of the security context is more commonly used for application separation utilising the MCS variant.

MCS is a simplified version of MLS, where all subjects and objects are assigned the same sensitivity level, and non-hierarchical categories. This extension allows to restrict access to cases where both source subject and target object have the same security category[17].

We included MLS/MCS in the overview of concepts for completeness, but we will not specifically focus on them in our policy, so we will not explain them in detail.

2.2 Using SELinux

In Section 2.1, we introduced the main SELinux concepts and mechanisms it uses to restrict operations on the system. As already mentioned, it is a MAC mechanism, and that requires some additional complexity in using and administering the underlying system.

Managing a SELinux-protected system comprises three activities - activating SELinux on a system, adjusting the SELinux policy to the specific needs of the users, and solv-

ing SELinux-related problems. In this section, we expand upon these requirements and explain how SELinux interferes with the activities of system users and administrators.

For more details about SELinux administration and support tools, see Chapter 8.

2.2.1 Activating SELinux

Activating SELinux on a system requires enabling the SELinux infrastracture and loading a SELinux policy[1]. The infrastructure itself denies all operations, and does not define any exceptions. All allowed operations are defined in the policy, which determines the nature of the respective SELinux installation (i.e. how strictly protected the system resources are).

The policy covers aspects of the system in a large detail and its language is not so straightforward, therefore users do not generally write their own SELinux policies. Instead, they use a ready-made generic policy provided by their respective Linux distributions. The distribution policies are adjusted versions of a so-called *reference policy*, which is provided and regularly updated by the SELinux developers.

Once installed, the SELinux policy determines how all subjects and objects of the system are labeled, and controls all accesses to them. The policy is practically immutable, in that the users generally do not change the active policy, except for applying updates and patches. They can make small adjustments in the configuration, but these must all lie within the borders defined by the policy.

2.2.2 Customizing SELinux policy

SELinux policy is rather static, but it leaves some flexibility for the user, as long as the decisions are in compliance with the policy. The user can do the following changes:

1. *Activate/deactivate modules.* The policy modules generally correspond one-to-one to software applications that they confine, and so the users usually load and unload the policy modules when they install or uninstall the respective applications.

2. *Switch a specific SELinux domain to permissive mode.* In this mode, SELinux does not block any accesses of the specified domain but logs all performed operations that are not allowed by the policy.

3. *Change some decisions of the policy.* SELinux policy can have conditional statements built-in, which are evaluated based on the current value of boolean flags controlled by users. Booleans allow parts of SELinux policy to be changed at

[1]See Section 8.1 for a detailed guide on how to activate SELinux

runtime, without any knowledge of SELinux policy writing. This allows changes without reloading or recompiling SELinux policy[13].

4. *Toggle between active roles of the SELinux user.* The active role of the user determines the set of domains that the user is allowed to enter. The role change must be allowed in the policy.

5. *Manage security labels.* Users can temporarily or persistently change file and directory labels and thus override default security labels defined in the policy. It is required for incorrectly labeled or unlabeled applications (such as newly-installed or compiled software), or in cases when the users want to change the privileges of the program. The change must be allowed in the policy.

It is easy to perform these operations using a wide range of available SELinux support tools[18]. The difficult part of these decisions is that they require understanding of the policy structure and philosophy - the available modules, domains, roles, booleans and appropriate security contexts, respectively.

2.2.3 Solving SELinux-related problems

Another aspect of using a SELinux-protected system is that the users must be able to respond to situations when SELinux denies a desired operation. The denials are logged as error messages[2], that can generally have four primary reasons[19]:

1. *Labeling problems.* Access control decisions in SELinux are based on security contexts of subjects and objects. If they are incorrect (i.e. inconsistent with the policy), SELinux will not function properly. Mislabeling can occur when a file is copied from a different location or medium, or when a new software is installed and is yet unlabeled. Solution to this problem is to restore the security contexts to those defined in the policy, or to define a more appropriate context assignment for the respective object.

2. *Inappropriate policy configuration.* Even if the security contexts are in compliance with the policy, the policy configuration might not be in accordance with the user's requirements. For example, SELinux policy for `httpd` might not allow it to send mail, in order to prevent a compromised website from becoming a spam box. However, we might want our `httpd` to send mail legitimately. The solution in this case could require changing a value of the appropriate boolean flag, and in other situations switching to a different role or otherwise customizing the policy, as described in Section 2.2.2.

[2]For more details, see Section 8.3.

3. *Bugs in policy or applications.* The problems might be caused by bugs in the application, or in the policy confining the application. These must be solved on a case-to-case basis.

4. *The machine has been compromised.* Inevitably, some SELinux error messages might be created because an application is compromised and tries to do something it is not allowed to do. In this case, SELinux can partially act as an intrusion detection system, and it can help the user identify an ongoing attack on the system.

In general, facing SELinux error messages usually requires analysing SELinux logs, relabeling files, changing SELinux configuration or disabling some parts (e.g. putting a domain into permissive mode). These solutions are supported by SELinux tools and require understanding of the active policy and SELinux logs. More advanced approach is to modify the SELinux policy to allow more operations, which requires both understanding of the active policy and SELinux policy language. Alternatively, the policy can be generated automatically by SELinux tools[3], but this must be used carefully, as permitting further operations could pose a security threat.

2.3 SELinux development and support

In this section, we examine how SELinux is maintained and used in practice.

SELinux originally started as the Flux Advanced Security Kernel (FLASK) development by the Utah university Flux team[20] and the US Department of Defence. The development was enhanced by the NSA[21] and released as open source software.

To date, part of its development is still carried by the NSA[21], while the majority is done by the developers community. Several projects are active, e.g. to maintain SELinux integration in userspace[22] or Linux kernel[23].

2.3.1 SELinux reference policy

At first, a so-called *strict policy* was developed, where, by default, all operations on the system were denied and additional privileges could be granted to the processes with specific rules. This, however, turned out to be too complex and difficult to maintain.

To address this issue, another philosophy was introduced, and a development of a *targeted policy* has started. In this policy, the majority of the processes are executed in a special *unconfined* domain where all meaningful operations are explicitly allowed. Only system processes, and those with the network access or working with security-critical data are treated individually (all operations are denied by default, unless allowed by

[3]The `audit2allow` tool can generate policy `allow` rules from logs of denied operations.

policy rules). The users and user applications are usually unlimited, i.e. running in the *unconfined* domain.

A set of basic policy modules combining the strict and targeted approach is maintained as an upstream *reference policy*[24], which is further adjusted for specifics of Linux distributions by its developers.

The support, however, varies across the distributions[25]. We have reviewed four major Linux distributions: Debian, Ubuntu, OpenSUSE and Fedora.

Ubuntu

Ubuntu uses AppArmor as a default security mechanism[7], which is incompatible with SELinux.

It is possible to replace AppArmor with SELinux in Ubuntu, but SELinux policy packages are not being maintained since 2009. Instead, the Ubuntu developers recommend using the Debian reference policy[29].

OpenSUSE

Similarly to Ubuntu, the default security module in OpenSUSE is AppArmor, but SELinux can be installed and enabled instead. The reference policy is regularly updated and some major applications are covered.

Debian

The Debian packaged Linux kernels have SELinux support compiled in, but disabled by default. Packages with a policy and basic tools can be installed to enable SELinux[26].

Tools for the policy maintenance and extension are provided, both text-based and graphical.

The default SELinux policy[27] (derived from the reference policy) is being maintained and tested by the developers, however, this is mostly for server installations. Desktop versions of Debian are not heavily tested with SELinux[28], and even the developers expect issues to arise.

Indeed, our tests showed that the default Debian SELinux policy is flawed in all of the latest Debian versions (7-9). With SELinux enabled, they are not able to function without major issues, as serious as issues with booting the system or with X-windows subsystem.

Fedora

SELinux is fully integrated into the Fedora OS and no further installation is needed. The reference policy is regularly updated and tested, and over 200 modules for desktop

applications are provided. There are plenty additional tools for the policy maintenance.

SELinux is given by far the best support by the Fedora developers, who participate actively in the reference policy development. Both of their policies, however, concentrate on the server environments.

2.3.2 Reference policy problems

Regardless of the support of SELinux reference policy in Linux distributions, we have some security concerns about the reference policy design for desktop environments.

The downside of the reference policy is that, while the system applications and services are covered thoroughly by the policy, this is not the case for the user applications. Some major applications have a fully working policy module available, but it is definitely not a general rule.

Since there are a lot of applications without an available SELinux module, there is a natural need to ensure they work even when no specific module was written for them. This is done by completely ommiting the security in favour of the usability. These applications are executed in the *unconfined* domain where virtually all the possible operations are explicitly allowed.

As a result, the SELinux policy in practice is quite unbalanced. Some applications are covered in detail, operating with a restricted set of operations, while others are not covered at all, free to perform any operation, without any security safeguards[4]. SELinux is thus not used up to its full potential, as it does not limit consequences of vulnerable user applications being exploited.

Even worse, SELinux reference policy has evolved to such an extent that it is not easy to understand, customize and configure - to illustrate, there are approximately 4 thousand types and 100 thousand rules defined in over 400 modules. A lot of problems arise when regular users attempt to use SELinux. The updates of the reference policy, Linux kernel and applications are not synchronized, so often some operations the users wish to perform are not permitted by the policy, which might cause frustration and, subsequently, disabling SELinux as a security module in Linux, and refusing the protection it offers.

2.4 Summary

SELinux is an access control mechanism far more complex than other DAC and MAC mechanisms, with a potential to protect the operating system significantly. But when it comes to Linux desktop environments, this potential is not fully fulfilled.

[4]Of course, the operations are still limited by traditional DAC mechanisms, but these are insecure by design, as we described in Chapter 1.

In some major Linux distributions, SELinux infrastructure is supported but the reference policy is currently not compatible with desktop environments, due to its complexity and difficult maintenance. But even in the distributions that have support of SELinux as their priority, the used policy is unbalanced. Majority of user applications are not limited, unless there is a specific policy module written for them.

As demonstrated, customizing and administering a system with SELinux reference policy installed requires knowledge of this complex policy. In practice, this often discourages security-concerned users from further exploring its potential, and ordinary users from using SELinux at all.

In Chapter 3, we review existing solutions that try to solve problems with SELinux usability and security in practice, and introduce our approach and goals of this thesis.

Chapter 3

Improving SELinux coverage and usability

The objective of our thesis is to extend the reference SELinux policy to overcome its problems, as described in Chapter 2. In the following chapter, we review existing attempts to solve the problem of SELinux poor coverage and usability in desktop environments. We identify deficiencies in these solutions, and introduce our, fundamentally different, approach.

3.1 Previous work

The issue of insufficient usability and/or security of SELinux-enabled desktop systems has already been studied by other researchers. We have observed several attempts to increase the scope of SELinux reference policy, in expanding the number of applications confined by policy modules.

In the Bachelor theses by Jan Jiřinec[25] and Michael Sládek[30], the authors have decided to extend the reference policy by adding custom policy modules for the applications of their choice. The result of each of the theses was a new policy module covering one desktop application - Evince in GNOME environment and BackupPC in Fedora environment, respectively.

Their contribution thus was adding new modules to the reference policy, in the same fashion that the SELinux community gradually expands the policy.

The Master thesis by Ondřej Vadinský[17] analysed the issue from a higher perspective. He viewed the desktop environment as a set of four logical layers: *userspace*, *desktop environment*, *services* and *applications*. To cover the security requirements of the environment, he created rules for each of the layers, either by creating a new policy module, or by modifying an existing module (such as *gnome* and *kde* modules for the desktop environment and *userdomain* module for the userspace environment).

Ultimately, the result of this thesis was a series of patches to existing reference policy modules, and a set of new SELinux policy modules for five specific applications - the most frequently used applications in Fedora desktop environment KDE (KMail, KAddressBook, Konqueror, KWallet and Akonadi).

The reasoning in this thesis is more general than in the previous two cases, but the choice of the rules for the respective policies was not justified properly with the respect on the security of the system, and the security of the policy was not tested. The authors used a *polgengui* tool to generate a set of basic rules for the applications, rather than allowing permissions based on the assessment of the requirements of that respective application.

We see these as major problems as the solutions are both incomplete and lack explicit justification behind the security guarantees they provide. Moreover, they did not change the fact that user applications are unlimited by default.

This was partially addressed by a policy by Dominick Grifith[31]. This policy is intended for more advanced users, and its goal is to confine the user space by default. It is configured in a *semi-strict* fashion where the unconfined logins are prohibited by default, but the unconfined domain can be entered manually.

The author adds new modules for some user applications. The policy also adds SELinux user isolation, in that various Linux users are mapped to various SELinux user identities, and users thus cannot interfere with each other[31].

We were not able to test this policy as the repository is unreachable, and is most probably outdated anyway (the last update was made in 2010). Our policy will share some ideas with this policy, such as confining the user applications by default, but will go further than adding modules for a few specific applications.

3.2 Our solution

In this section, we explain our approach how to improve security and usability of SELinux in practice, and we introduce the goals we set for our SELinux policy.

3.2.1 Target distribution

We will focus on the Debian distribution, as it supports SELinux infrastructure by default (unlike Ubuntu), but does not currently provide a working policy for desktop environments (unlike Fedora). One of our goals will be to fix the problems of reference policy, that currently make a SELinux-enabled Debian system unusable.

Since Fedora already provides a working SELinux policy and active support from their development team; and also all the aforementioned theses have focused on Fedora,

we believe our thesis will thus have a greater impact for the Linux community if we concentrate our efforts towards a different Linux distribution[1].

3.2.2 Classifying applications

The existing solutions of extending reference SELinux policy all concentrate on adding new policy modules for specific applications. We aim to generalize the task and cover more applications at once.

With reference policy, if there is no policy module confining a particular application, the user can either leave it unconfined, or write a specific module for that application from the scratch. Writing a set of rules for each application separately can be a challenging and time-consuming task. In an ideal world, it would be the application authors' responsibility to provide a SELinux module for the application since they are the ones who know the best how the application works, but this is often not the case. If another developer or security enthusiast tries to do it instead, some marginal use case scenarios of the application might stay uncovered, which may result in the application not working properly.

Our extended policy will provide a series of modules for more general categories of applications, rather than specific applications. This will provide a SELinux-aware and security-concerned user a third option - to limit the application with a more general set of rules.

The policy will contain modules for groups of applications, rather than for every single application independently. As a result, a compromise between a detailed application-oriented policy and a general policy allowing all operations will be accomplished, with a certain level of security guaranteed.

3.2.3 Analytical approach

In the previous work, we have observed a rather practical approach to covering the desktop applications with a SELinux policy. Our vision of an ideal solution is more analytical - we believe it must be properly justified, and that our decisions to grant privileges to applications must be supported by arguments. Moreover, our solution will cover all applications equally, rather than concentrating on a specific set that is then fully covered while the others stay unnoticed. Our SELinux policy will achieve this by abstracting from the unnecessary details of the respective applications and focusing on the features that they have in common.

There are two main guidelines which we will follow. First, the policy must be easy to install and must not limit the users in their regular (non-malicious) activity.

[1]Nevertheless, it will not be difficult to port our solution to other Linux distributions supporting SELinux.

Second, all the applications must be restricted to some extent, i.e. the system must be protected against the most common attack scenarios.

We will back our decisions with analyses of common PC usage scenarios and common attack scenarios, that induce a set of security and functional requirements of the system, against which we evaluate our policy.

3.2.4 Compatibility and extensibility

We build our SELinux policy upon the reference policy, patch where necessary and extend it with new modules.

The reason behind this decision is that the reference policy mainly covers system applications and services, which is not the focus of our thesis. The reference policy has undergone years of research and development by the community; and we wish to benefit from these efforts, and further extend the qualities of the policy, rather than writing our policy from the scratch.

Also, our goal is to design our SELinux policy with the extensibility in our minds. Should a user conclude that a new, yet uncategorized set of applications deserves a special treatment, it should not be very challenging to extend our policy to meet more demanding requirements. It will be possible to add a custom, more detailed policy module for each application.

For example, a set of office applications can share one policy module but it will be possible to add another module for each of the respective applications. This module can be more benevolent and allow more operations if needed. Similarly, if we decide to further restrict one of the applications, we can assign it to another group of applications, either existing or newly-created, with more strict rules.

3.2.5 Target audience

In our model, we assume a system with one or multiple security-aware users. The protected system is a personal computer used by a small number of users, typically in a home environment. Since the reference SELinux policy already confines system applications and services in a great detail, we focus on user applications (both CLI and GUI).

We assume that the administrator of the system is trusted - otherwise he/she could disable the SELinux policy and circumvent all our security measures. Other (non-privileged) users may or may not be trusted.

The main goal for this thesis is to limit any damage caused on a system by rogue applications. This requires the users to be *security-concerned* (and SELinux-aware), because they have to actively work with the SELinux policy components, in order to

fully use its potential. In practice, this means to protect the user privacy, to ensure the confidentiality and integrity of user data, and to prevent the system from being misused or corrupted.

As described in Section 2.2, this requires labeling newly installed applications or incorrectly copied files, and doing the basic diagnostics if something goes wrong (e.g. when a file is assigned an incorrect context or the policy prevents a desired action).

Security-concerned users recognize the value of SELinux security mechanisms, and want to use them to secure their desktop environments. Our policy provides these users with security guarantees, without the need of studying the reference policy in detail or writing their own policies, but they must understand the main SELinux concepts and the philosophy of our policy.

This model assumes trusted users with untrusted software, and we assume two situations that the users should be able to handle:

- After installing a new application (such as web browser), the user should be able to classify the application into one of our policy modules, which correspond to categories of applications. The classification should be realized by permanently setting the correct security contexts to the application itself and associated files. By setting the security contexts, the application is restricted from undesired operations, determined by its category.

- The user should be able to execute an application in a different domain, without the overhead of setting the correct security contexts. This can include executing a web browser in a more restricted domain when browsing on untrusted websites, or in a more trusted domain, when sensitive data are being processed.

In case of *ordinary users*, we can assume they will generally not seek to further constrain their applications. Instead, they will be more concerned about the usability of the system.

These users should be able to use their systems with our SELinux policy installed, without a need for further configuration and maintenance, as is the current situation with the reference policy on Fedora systems.

Even in this scenario, our policy will guarantee a certain level of security for all programs. Since the ordinary users will likely use the default settings, the user applications cannot have unlimited privileges by default (as is the case in the reference policy with the default *unconfined* domain). In our policy, the default level of application privileges will be slightly restricted. Should the users be limited by this decision, they will have the option to switch each application to a less restricted (possibly even unrestricted) domain, but this will have to be done manually after interactive authentication.

The users will ultimately benefit from these settings - even if they incorrectly forget to assign an untrusted program to one of the categories, it will not gain unlimited privileges - for example, it will not be able to read users' most sensitive documents such as private keys.

3.3 Summary

To summarize, the target distribution of our work will be Debian (GNOME environment), and the target audience will be security-aware users.

We set the following goals for our SELinux policy:

- The *default domain* of applications should be restricted to some extent, i.e. user applications should not be unconfined by default. The default level of privileges can be partially customizable by the user.
- The (security-concerned) user should be able to, temporarily or permanently, *further restrict* any application by assigning it a security context associated with permissions more strict than those at the default level. The policy should cover practically all applications, i.e. the user should be able to choose such a security context that reflects the requirements of the application. This should be possible even at the cost of the policy being less detailed and less strict.
- The (ordinary) user should be able to *grant more privileges* to any application, in case none of the designed categories represents a specific application accurately. This mechanism should be present as a fallback solution, in order not to disrupt usability of the applications/system.
- Our policy should be thoroughly tested, not only from the usability perspective, but also against the security requirements. The policy design and implementation should be based on analysis of common PC usage use case scenarios, and most prevalent threats.
- Our policy should be compatible with the reference policy and easily extensible.

As a result, our approach is to build our policy upon the reference policy, and add new modules, while using the existing structure of the reference policy and the modules for system applications and services.

For user applications, we will implement a small number of modules representing *categories* of applications, according to their functional and security requirements. We further define a default constrained domain, and a fallback unconfined domain.

Part II

Analysis and design

Chapter 4

Security requirements

There are various scenarios of how a Linux system can be compromised, and various techniques that malicious software leverages on the infected system. In this chapter, we analyse the threat landscape and common adversary techniques, and determine how these should be reflected in our policy.

This will help us understand which resources we need to protect, and which restrictions we have to put on the applications.

As a reference in our analysis, we use an industry-recognized MITRE ATT&CK Framework[32].

4.1 MITRE ATT&CK Framework

Adversarial Tactics Techniques and Common Knowledge (ATT&CK) framework is a knowledge base of tactics and techniques used by attackers when compromising IT systems. The framework is provided by a non-profit security company MITRE[32], and it is globally-accessible.

The framework is based on real-world observations, and updated by security community. It is widely regarded today as the most comprehensive catalog of attacker techniques and tactics[33].

The catalog contains descriptions of over 200 *techniques* that adversaries may use over the course of an attack. Real-world examples of these techniques are also provided. The techniques are classified into groups called *tactics*, which are further divided into the following groups, based on the attack phase and the target platform:

- *PRE-ATT&CK* tactics and techniques describe activities of the malicious actors prior the actual attack, and include planning, reconnaissance or development of malicious tools.
- *Enterprise* tactics and techniques concentrate on the activities during the attack.

They are further divided into groups of techniques relevant for Linux, macOS and Windows platforms.

- *Mobile* tactics and techniques describe activities of the adversaries during the attack on mobile platforms.

4.2 Scope of the analysis

In this analysis, we mainly focus on *Enterprise* tactics and techniques relevant to Linux OS[34], but we also decided to include two techniques of the *Effects*[35] tactic. This tactic is only presented for mobile platforms in the Framework, but we consider them relevant for Linux OS as well, as they describe techniques of ransomware, which would otherwise not be represented in the analysis[1].

In total, we focus on techniques classified into the following 12 tactics: *Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Effects, Exfiltration* and *Command and Control.*

We could achieve the maximal security of a system by mitigating all of these techniques, but there are three reasons why it is unrealistic in our scenario.

First, the techniques are directed against various layers of system protection including physical security, firewall, or even security awareness of users. Some of them are beyond the control of SELinux, as they rely on software vulnerabilities or human mistakes. We exclude such techniques from our analysis.

Second, mitigating some of the techniques by SELinux could have undesired impact on usability of the system. These include techniques that are also employed by legitimate programs, such as the use of hidden files and directories (which can be misused for *Defense Evasion*), or listing running processes on the system (which can be misused for *Discovery*). Strictly blocking these techniques would contradict our goal of simplicity and usability.

Finally, some techniques require specific privileges, such as ability to execute application deployment software (for *Lateral Movement*). SELinux reference policy does not define a special category for such a software, nor do we plan to include it in our extension, and so mitigation of these techniques would be too complicated for our more general, high-level policy.

Nevertheless, even if we only concentrate on a subset of the malicious techniques, we can successfully fulfill the five main security goals of our thesis - to protect the

[1]This tactic is not included in MITRE ATT&CK Linux OS tactics, because these are more focused on targeted attacks, rather than mass-spreading malware. In this analysis, we consider both.

system from being misused or corrupted, and to protect user privacy, integrity and confidentiality of user data.

Techniques to achieve initial access, discovery or persistence are by nature not malicious, they only assist the malicious actors in performing their ultimate goals. The main focus of the malicious actors is usually on other techniques, such as techniques for collection of sensitive data, exfiltration or communication with the C&C server, which directly interfere with our security goals.

Our highest priority will therefore be to mitigate the techniques associated with the actual malicious activities, and less focus will be given to those that only play an assisting role. The techniques with lower priority can be reflected in our policy as conditional rules.

In the analysis, we determine the priority with which our policy will mitigate each respective tactic, and evaluate whether SELinux is able to mitigate the associated techniques, without breaching our usability goals.

4.3 Threat analysis

In this section, we review the relevant adversary tactics and techniques. We analyse which of them can be blocked with the use of SELinux, and whether their mitigation should be included in our security requirements.

We do not explain the individual techniques in detail. In case mitigation of the respective technique is beyond the scope of SELinux, we only mention the name of the technique and a short explanation why we do not consider it. For techniques that are relevant for defining the security requirements for our policy, we may include a short description, unless the name of the technique is self-explanatory.

The reader is advised to find the full descriptions on the project website[32], using the names of the techniques, which we are referring to using *italic* font, and technique ID, as defined by the MITRE ATT&CK Framework (such as *T1189*).

Some techniques can be used to achieve several tactics, with different possibilities of mitigation using SELinux, therefore, we sometimes list the same technique several times, but in different contexts.

4.3.1 Initial Access

The initial access tactic represents the attack vectors adversaries use to gain the first entry to the system[32].

According to the recent Internet Security Threat Report by Symantec[36], the most common infection vectors in targeted attacks are Spear-Phishing Emails (71.4%),

Watering-hole websites (23.6%), Trojanized Software Updates (5.7%) and Web Server Exploits (2.9%).

This distribution indicates that applications behind the most common attack vectors are web browsers and mail clients, and that they should be treated with a special caution. By limiting the privileges of these two types of applications, we can affect the privileges of majority of the malware introduced to the system.

MITRE ATT&CK Techniques

There are 9 initial access techniques in the Framework, none of which can be mitigated by a SELinux policy.

We exclude from the analysis the initial access techniques that rely on exploiting software vulnerabilities or bugs, which SELinux cannot prevent: *Drive-by Compromise (T1189)*, *Exploit Public-Facing Application (T1190)* and *Supply Chain Compromise (T1195)*.

Further, we exclude initial vectors that take advantage of human mistakes, such as falling for social engineering or browsing rogue websites, which SELinux also cannot prevent. The said techniques are *Spearphishing Attachment (T1193)*, *Spearphishing Link (T1192)* and *Spearphishing via Service (T1194)*.

SELinux also cannot prevent infection through legitimate channels, as is the case in the techniques *Valid Accounts (T1078)* and *Trusted Relationship (T1199)*, where the attacker gains access to the system using stolen credentials of a legitimate user, or by impersonating a trusted third-party.

Finally, the *Hardware additions* technique requires adding new hardware to the system, such as devices for network tapping, keystroke injection or kernel memory reading via DMA. This attack vector cannot be prevented from the OS level, let alone by SELinux.

In general, SELinux cannot prevent any of these initial access techniques, and so their mitigation is beyond the scope of this thesis. Our policy will disregard completely the individual scenarios of how a malicious code has been executed on a system, and focus on the actual malicious behaviour and other adversarial techniques.

Security requirements

Web browsers and mail clients should be treated with special caution, as these are behind the most prevalent intial attack vectors.

4.3.2 Execution

The execution tactic represents techniques that result in execution of adversary-controlled code on a local or remote system[32]. The execution techniques are not directly malicious, and have their legitimate use. They can be controlled by SELinux, but we should consider their mitigation by an optional policy.

MITRE ATT&CK Techniques

MITRE ATT&CK for Linux describes 10 execution tactics. We exclude *Exploitation for Client Execution (T1203)*, as it relies on software vulnerabilities, and *Trap (T1154)*, as it is a shell built-in that cannot be disabled by SELinux.

The other techniques can be controlled by SELinux, but it should be up to user whether a strict or more benevolent approach should be employed.

Standard techniques of (malware) execution, which require execute permissions, can be mitigated by blocking execution of specific files. These techniques include *User Execution (T1204)*, execution using *Command-Line Interface (T1059)* or *Graphical User Interface (T1061)*, execution using *Third-party Software (T1072)*, execution by scheduling a task or a periodic background job (*Local Job Scheduling (T1168)*), and execution of a shell (*Scripting (T1064)*).

The execution using *Source* command does not require the file to be executable, as it loads and executes the file in the current context. This technique can be mitigated by SELinux by restricting the ability of the `source` command to read untrusted files.

Security requirements

Our policy should provide an optional feature of application whitelisting[2], i.e. a mode in which the execution of untrusted software and/or scheduling it as a task is completely blocked. In this mode, all software with the default security context is considered untrusted and can be executed only after the user has changed its security context to a more privileged domain. As this feature can severely impact the usability of the system, it should be optional for the users.

If possible, the policy should limit the privileges of any process to execute a shell, and the ability of the `source` command to read untrusted software, in order to prevent its execution.

4.3.3 Persistence

Persistence is any access, action, or configuration change to a system that gives an adversary a persistent presence on that system[32].

[2]Similar to AppLocker[37] in Windows OS, not so prevalent in Linux[38].

Persistence techniques do not directly interfere with our security goals, and because of their legitimate use, the mitigation of some of them should be optional.

MITRE ATT&CK Techniques

MITRE ATT&CK Framework describes 13 persistence techniques on Linux systems.

Our policy should block the use of *Bootkit (T1067)* and *Kernel Modules and Extensions (T1215)* for all user applications, as there is no legitimate scenario when they should interfere with the boot sectors or kernel modules. It is one of our priorities to protect the integrity of the system.

Ensuring persistence by modifying *.bash-profile and .bashsrc (T1156)*, by scheduling a task (*Local Job Scheduling (T1168)*), and by ability to *Create Account (T1136)* can be both legitimate and rogue. SELinux can provide means to prevent these techniques, but that could clash with some legitimate use-case scenarios. Mitigation of these techniques should be optional in our policy.

As for the *Web Shell* technique, SELinux can mitigate the scenario where malware runs a web server on the client machine, allowing access to the infected machine using a command-line interface. The situation when malware exploits an existing web server, cannot be prevented by SELinux.

Following are the other persistence techniques, all of which we exclude from our analysis:

The *Setuid and Setgid (T1166)* technique is excluded, because it relies on a software vulnerability in an application with the setuid or setgid flag set. *Redundant Access (T1108)* because it is a strategy, rather than a tangible technique, so in general, it cannot be prevented by SELinux.

The `trap` command (*Trap (T1154)*) is built into the shell and can register code to be executed upon receiving specific interrupt signals (e.g. Ctrl+C keyboard interrupt). This mechanism cannot be blocked by SELinux.

Browser Extensions (T1176), *Hidden Files and Directories (T1158)* and *Valid Accounts (T1078)*, because they are based on misusing legitimate functionality, and controlling them could have a negative impact on usability.

SELinux does not have the ability to detect or prevent *Port Knocking (T1205)* technique. SELinux can block network access to ports, network interfaces and even at the packet level, but cannot analyse patterns in the network communication.

Security requirements

All user applications should be denied access to the boot sector of hard drives, and the ability to load/unload kernel modules.

Our SELinux policy should optionally limit the ability to modify `.bash-profile` and `.bashsrc` scripts; to schedule a task using `at`, `cron` or `launchd`; to create user accounts; and to run a web server.

4.3.4 Privilege Escalation

Privilege escalation is the result of actions that allows an adversary to obtain a higher level of permissions on a system or network[32].

In general, the reference SELinux policy is more strict than the traditional MAC mechanisms, and SELinux restrictions are in place even after a malicious applications has gained administrator privileges. Thus, mitigating these techniques will not be our main focus.

MITRE ATT&CK Techniques

Process Injection (T1055), especially to an elevated process, should be denied to all user applications.

SELinux cannot prevent misusing poor configuration of the `sudoers` file (*Sudo (T1169), Sudo Caching (T1206)*) but it can prevent the applications from weakening a good configuration, by controlling write access to the `sudoers` file.

SELinux cannot prevent *Web Shell (T1100)*, *Exploitation for Privilege Escalation (T1068)* nor *Setuid and Setgid (T1166)*, as they rely on exploiting software vulnerabilities. Neither can it mitigate the *Valid Accounts (T1078)* technique, where the user gains access to the system using stolen credentials.

Security requirements

Our policy should block the ability of an application to inject into another process, and the ability to manipulate the `sudoers` file.

4.3.5 Defense Evasion

Defense evasion consists of techniques an adversary may use to evade detection or avoid other defenses[32]. We will mostly be concerned with the techniques that can evade/disable SELinux, the other techniques will be treated with lower priority.

MITRE ATT&CK Techniques

First and foremost, our policy should control access to disabling SELinux, so as not to make its protections ineffective (*Disabling Security Tools (T1089)*). Low-level access to system should be carefully controlled, to mitigate *Rootkits (T1014)*.

Further, the policy should limit the ability to *Install Root Certificate (T1130)*, to inject into another process (*Process Injection (T1055)*), to *Clear Command History (T1146)* and manipulate *HISTCONTROL (T1148)*.

It is possible but not necessary to mitigate *File Permissions Modification (T1222)* and *Indicator Removal on Host (T1070)*, as other features of SELinux can naturally assume these roles (DTE and SELinux logging).

We do not consider techniques *Masquerading (T1036)*, *Obfuscated Files or Information (T1027)*, *Port Knocking (T1205)*, *Web Service (T1102)*, *Hidden Files and Directories (T1158)*, *File Deletion (T1107)*, *Binary Padding (T1009)*, *Scripting (T1064)*, *Timestomp (T1099)*, *Redundant Access (T1108)* nor *Indicator Removal from Tools (T1066)*, as they are aimed at other layers of security protections, and legitimate from the SELinux perspective.

SELinux also cannot prevent software vulnerabilities that facilitate *Exploitation for Defense Evasion (T1211)* or using leaked user credentials for logging into *Valid Accounts (T1078)*.

Security requirements

Our policy should control access to disabling SELinux and changing SELinux configuration; and should prevent installation of rootkits.

It should also limit the ability to manipulate installed certificates, to inject into another process, to clear command history and logs, and to manipulate certain file permissions.

4.3.6 Credential Access

Credential access represents techniques resulting in access to or control over system, domain, or service credentials[32]. These techniques are malicious by nature, and are often deployed by keyloggers, banking trojans or other types of spyware, so it is one of our priorities to mitigate them.

MITRE ATT&CK Techniques

Our policy should mitigate *Input Capture (T1056)*, *Network Sniffing (T1040)* and *Two-Factor Authentication Interception*, as long as the second authentication factor is a hardware token and the credentials are intercepted with the use of a keylogger. The policy cannot prevent interception of out-of-band communications (SMS).

It is important that the policy restricts access to *Credentials in Files (T1081)* and *Private Keys (T1145)*.

If possible, the policy should also limit the possibility to dump credentials from the *Bash History (T1139)* and by reading information from the `/proc` filesystem (*Credential Dumping (T1003)*).

SELinux cannot prevent *Brute Force (T1110)* techniques and *Exploitation for Credential Access (T1212).*

Security requirements

Our policy should prevent capturing pressed keystrokes and unauthorized network sniffing. If possible, it should limit the possibility to dump credentials from the bash history and and by reading information from the `/proc` filesystem.

The policy should provide special protection to files with credentials and private keys.

4.3.7 Discovery

Discovery consists of techniques that allow the adversary to gain knowledge about the system and internal network[32]. All of the discovery techniques could be blocked by SELinux, but since they do not directly interfere with our security goals, and due to their otherwise legitimate use, we will not particularly concentrate on their mitigation.

MITRE ATT&CK Techniques

Most discovery techniques do not leak sensitive data of the user, so we will prefer usability to security in this case. This applies to techniques *Account Discovery (T1087), Browser Bookmark Discovery (T1217), Network Service Scanning (T1046), Password Policy Discovery (T1201), Permission Groups Discovery (T1069), Process Discovery (T1057), Remote System Discovery (T1018), System Information Discovery (T1082), System Network Configuration Discovery (T1016), System Network Connections Discovery (T1049)* and *System Owner/User Discovery (T1033).* Some applications may not be allowed these accesses in our policy, but it will not be our main focus to control them.

Our policy should be focused on techniques that can threaten confidentiality of user data, and should limit the ability of user processes to perform *File and Directory Discovery (T1083)* and *Network Sniffing (T1040).*

Security requirements

Our policy should prevent unauthorized processes from performing network sniffing; and should limit the ability to read user files (sensitive and/or non-sensitive).

4.3.8 Lateral Movement

Lateral movement consists of techniques that enable an adversary to access and control remote systems on a network. These techniques are examples of spreading mechanisms within the infected network, and are commonly used by worms.

We should limit the lateral movement techniques, as they pose a threat to security of other systems within the network.

Techniques

Remote File Copy (T1105) can be mitigated by SELinux. Our policy should limit the ability of user applications to use certain network protocols, such as SMB or FTP, or to use network in general, unless necessary.

We can also mitigate *SSH Hijacking (T1184)* by controlling access to credential stores.

We will not consider mitigation of *Exploitation of Remote Service (T1210), Third-party Software (T1072), Application Deployment Software (T1017)* and *Remote Services (T1021)*, as they rely on software vulnerabilities or leaked user credentials, which are beyond the control of SELinux.

Security requirements

Our policy should limit network access of user applications, and should limit the range of network protocols they can use for the communication. The policy should protect access to sensitive user information, such as credential stores.

4.3.9 Collection

Collection consists of techniques used to identify and gather information, such as sensitive files, from a target network prior to exfiltration[32]. Data collection is the main goal of spyware, and is directed against the user privacy and confidentiality of user data. Therefore, it is one of our primary goals to mitigate collection techniques.

MITRE ATT&CK Techniques

Our policy should mitigate *Audio Capture (T1123), Screen Capture (T1113), Input Capture (T1056)* and collection of *Clipboard Data (T1115).* To a reasonable degree, it should protect *Data from Local System (T1005), Data from Network Shared Drive (T1039), Data from Information Repositories (T1213)* and *Data from Removable Media (T1025).*

We do not consider mitigation of *Data Staged (T1074)* and *Automated Collection (T1119)* techniques, as these refer to coding practices of the malware, rather than techniques with impact on data confidentiality.

Security requirements

Our policy should protect access to user data on all drives - fixed, removable or remote. Unless necessary, the user applications should not be allowed to read, write or otherwise modify the data.

The ability to record audio and video using microphone and webcam device, to record keystrokes and to capture screenshots, should only be allowed when absolutely necessary.

4.3.10 Effects

Effects on the system are actions that interfere with our security goal of preventing the system from being corrupted, and protection of user data integrity. Those techniques are commonly used by ransomware, disk wipers and other malware types whose main goal is blackmailing, sabotage or destruction.

MITRE ATT&CK Techniques

Our policy should block the ability to *Encrypt Files for Ransom (T1417)* (used by ransomware), in order to protect the integrity of user data.

We should also mitigate the *Wipe Device Data (T1447)* technique, in order to protect the integrity of the system.

Security requirements

Whenever applicable, our policy should block user applications from modifying user and system data.

4.3.11 Exfiltration

Exfiltration refers to techniques and attributes that result or aid in the adversary removing files and information from a target network[32]. Exfiltration techniques are crucial for the successful operation of spyware, and we should mitigate them with the highest priority, as they threaten the confidentiality of user data.

MITRE ATT&CK Techniques

SELinux cannot prevent the techniques of *Automated Exfiltration (T1020), Data Compressed (T1002), Data Encrypted (T1022), Data Transfer Size Limits (T1030)* nor *Scheduled Transfer (T1029)*, as these appear legitimate to SELinux.

The policy also cannot block *Exfiltration Over Command and Control Channel (T1041)*, once such a channel has been established. Possibilities of how to prevent establishing such a channel are analysed in Section 4.3.12.

On the other hand, the policy can prevent *Exfiltration Over Alternative Protocol (T1048)*, different from the C&C protocol. FTP, SMTP or DNS are a few examples. The policy should also prevent *Exfiltration Over Physical Medium (T1052)* and *Exfiltration Over Other Network Medium (T1011)* such as WiFi connection or Bluetooth.

Security requirements

Our policy should limit the ability of the applications to create and maintain network connections (using various protocols), and the access to removable media.

4.3.12 Command and Control

The command and control tactic represents how adversaries communicate with systems under their control within a target network[32]. It is crucial to mitigate the command and control techniques, as they are used by the majority of the malware. Command and control communication helps the malware exfiltrate collected infiltration (spyware), receive further commands (backdoors), download new modules and so on.

MITRE ATT&CK Techniques

Whenever possible, our policy should block the ability of user applications to communicate over network, either using a *Standard Non-Application Layer Protocol (T1095), Standard Application Layer Protocol (T1071)* or *Custom Command and Control Protocol (T1094)*, *Commonly Used Port (T1043)* or *Uncommonly Used Port (T1065)*.

The possibility of *Communication Through Removable Media (T1092)*, *Remote File Copy (T1105)* and using legitimate *Remote Access Tools (T1219)* should be given only when required.

We do not consider techniques *Custom Cryptographic Protocol (T1024), Standard Cryptographic Protocol (T1032), Multilayer Encryption (T1079), Data Encoding (T1132), Data Obfuscation (T1001), Multiband Communication (T1026), Port Knocking (T1205), Web Service (T1102), Multi-Stage Channels (T1104), Connection Proxy (T1090), Multi-hop Proxy (T1188), Domain Fronting (T1172)* and *Fallback Channels*

(T1008), as they are aimed at other layers of security protections, and legitimate from the SELinux perspective.

Security requirements

Our policy should control the ability of user applications to communicate over network, using various network protocols and ports.

The policy should limit the possibility to execute remote access tools. If possible, access to removable and remote drives should be limited.

4.4 Summary

Our extension of SELinux reference policy should mostly mitigate the adversary techniques associated with credential access, effects, collection, exfiltration, command and control and some techniques of lateral movement.

Some of the MITRE ATT&CK techniques are beyond the control of SELinux and must be mitigated with the use of other security protections. We do not focus on these techniques.

Mitigation of other techniques could potentially have a negative impact on the usability of the system, and should thus be reflected into a conditional policy, or omitted.

In this chapter, we defined a set of security requirements for our policy. In the following chapter, we will use them to suggest a design of the policy.

Chapter 5

Functional requirements

The goal of this thesis is to provide a SELinux policy that covers the whole Debian system. The reference policy already confines system applications and services in a great detail - it defines policies for the bootloader, init, for SSH service and many others. We assume that these policies are compliant with the functional and security requirements of these programs.

This thesis extends the reference policy by covering the other part of the system - user applications. This chapter is dedicated to analysing functional requirements of these programs. We identify the most widely used user desktop applications, and sort them by their common features.

Finally, we confront the requirements of legitimate applications with the security requirements defined in Chapter 4, and create a high-level classification of user applications to be reflected in our policy.

5.1 Classification of user applications

In order to define functional requirements of user desktop applications, we first started with analysing user-composed lists of popular Linux applications[39] [40] [41] [42] [43]. We sorted these applications by their functionality and created the following categories:

- Web Browsers (e.g. Firefox, Vivaldi)
- Mail clients (e.g. Thunderbird, Evolution)
- Basic utilities (e.g. Calculator, Calendar, Notes)
- Office software/text editors (e.g. LibreOffice, Sublime)
- Code editors (e.g. Atom)
- Image/photo editors (e.g. GIMP, Inkscape, Pinta)

- Image/photo galleries, PDF readers (e.g. Eog, Evince, Okular)
- Electronic book/magazine readers (e.g. Calibre)
- Video/audio players (e.g. VLC, Totem)
- Streaming applications (e.g. Spotify, Tomahawk)
- Video/audio editors (e.g. Audacity, Openshot)
- Audio/video/screen recorders (e.g. Handbrake)
- File management tools - archivers, converters (e.g. winFF)
- Cloud storage services (e.g DropBox, pCloud)
- Downloading tools, torrent clients (e.g. Transmission, FileZilla)
- Messaging applications (e.g. Skype)
- CD/DVD burning/disk formatting applications (e.g. WoeUSB)
- Basic local games (e.g. Mines)
- Gaming platforms (e.g. Steam)
- Applications for digital signature creation/verification (GPG)
- Applications for encryption/decryption (OpenSSL, Kleopatra)
- Key/password managers (e.g. KeePass)

Some other applications, such as backup utilities, WINE or Tor, are already covered by the reference policy - there are specific modules confining them. We will therefore not consider them in the further analysis.

5.2 Design of our policy

Our original plan was to create a SELinux policy module to cover each type of applications defined in the previous sections. To illustrate, instead of the traditional approach where each specific video player would be confined by a separate, detailed policy module, in this approach, one module would cover all video players.

This approach has, however, proven insufficient. Some of the types of applications have overlapping functionality and can be easily merged into one category, to decrease the complexity of the policy. For example, image/photo galleries, PDF readers and video/audio players all require read-only access to user data. Video/audio players also

need a permission to play sounds, but this permission itself is not considered security-sensitive and does not threaten integrity nor confidentiality of user data. Therefore, we can create one general category of file viewers with these permissions, which will cover all these types of applications.

More importantly, there are two classes of applications that require special treatment. As demonstrated in Section 4.3.1, mail clients and web browsers are associated with the most common attack vectors, and thus their privileges should be strictly limited. Moreover, both of them are a (frequently used) large piece of software, and there can be vulnerabilities which can be exploited when processing malicious data from the Internet. From the security point of view, all these reasons suggest the web browsers and mail clients should only have minimal privileges.

On the other hand, both of these categories represent a complex piece of software with a wide range of features, which require severe privileges.

Naturally, web browsers require network access to communicate over web protocols (HTTP, HTTPS), but there are other requirements. For example, users often wish to upload any local file, or download a file using the browser and store it in any local directory. Further, using online messaging and teleconferencing services may require access to webcam or microphone. Finally, to be able to use a private key for authentication for some web services, the browsers require read access to sensitive user data. In order to satisfy all these requirements, a full access to user data (including sensitive data) and these devices would have to be granted to web browsers.

The situation is similar with mail clients. Apart from network communication over mail protocols, the users require the ability to attach files to email messages, and to save attachments locally, for which the mail clients need read/write access to user data. To provide active hyperlinks in the email messages, the mail clients need to be able to execute web browsers. To correctly display images in emails from external sources, they need to communicate over other than mail protocols. Finally, to be able to digitally sign or encrypt emails, the mail client requires read access to sensitive user data.

Our solution to this problem will be to create several SELinux modules to confine web browsers, and several modules to confine mail clients. These modules will vary in the range of privileges, and we will enable a security-concerned user to decide which mode will be used as the default and when to switch to another (more or less privileged) mode.

In conclusion, our extension of reference SELinux policy will comprise a number of additional modules for various groups of desktop applications. These groups of applications will loosely correspond to classes defined in Section 5.1, but it will not be a one-to-one assignment. Some modules are intended for several classes of applications (since they have common security and functional requirements), and some applications

are covered by several modules with different privileges (web browsers and mail clients).

5.3 Requirements for our policy modules

In this section, we present the list of modules that will be implemented by our policy, supplemented with the functional requirements, example applications and typical use case scenarios. The applications are assigned the least privileges needed for their flawless operation, while limiting potential damages if they are exploited.

When referring to user data in this analysis, we distinguish between sensitive and non-sensitive (general) user data. Sensitive data include password databases, encryption keys and other data the user identifies as such.

In particular, we will recognize download/upload data, which, as a rule, refers to the Downloads folder; and configuration data, which refers to locations in the user home directory where applications store user configuration and temporary data. All these data are considered non-sensitive, but sometimes we will use this more fine-grained separation.

When referring to network access, we will distinguish between network access over specific application-layer protocols, general network access over standard protocols, and unlimited network access (including access to raw sockets etc).

When referring to devices, we will distinguish between default, non-privileged access to devices (access to terminals, speakers etc), and privileged access to devices, which can include access to security-sensitive devices (such as webcam or microphone) and raw access to devices (such as disks).

5.3.1 Simple local applications

Simple local applications such as games (*gnome-mines, gnome-chess*) and basic GUI utilities (*gnome-calculator, gnome-todo*). These applications only process their configuration data (user settings, game score statistics) and do not need any access to user data or network.

5.3.2 File viewers

Local applications for reading documents (*evince*), displaying images or photos (*eog*); video and audio players (*vlc, totem*). These applications only require read access to user data, there is no need to modify them, except for their configuration data. Access to sensitive user data or network access is not required.

This mode can also be used to display untrusted documents downloaded from the internet, or email attachments from unreliable sources. PDF viewers or office software

can contain security vulnerabilities that can be exploited when processing a malicious file from a rogue source. In this scenario, these applications can be executed in a more restricted, read-only domain, in order to limit damages caused by the exploited application.

5.3.3 File editors

General local applications processing non-sensitive user data. Examples include office suites (*libreoffice*), code editors (*atom*), image/video/audio editors (*gimp*), paint applications (*pinta*), photography applications or file archivers. Full access to non-sensitive user data is needed. These applications do not require network access nor access to sensitive user data.

5.3.4 Trusted file viewers

Applications for creating digital signatures, data encryption and decryption. They require read access to sensitive user data (such as private keys) and full access to non-sensitive user data (for example, to read a document being signed, and create a signature for that document). Network access is not required.

File viewers should generally be executed in the default, non-trusted mode (see *File viewers*), but this trusted mode can be used to read sensitive user data. For example, the user can execute an office software in this trusted mode to be able to read sensitive documents such as private keys or passwords.

5.3.5 Trusted file editors

Local applications with full access to user data, including sensitive data such as private and public keys, and passwords. Examples include password and key managers (*seahorse, keepass*).

This mode can also be used for executing standard text editors, for example to edit sensitive user data.

5.3.6 Device recorders

Local applications for recording audio/video and screen recording using microphone/webcam devices (*gnome-sound-recorder, cheese*). These applications need full access to non-sensitive user data and to webcam and microphone devices.

5.3.7 Web browsers

This class of applications is dedicated solely to web browsers, such as *firefox-esr*, *chromium* or *konqueror*. As web browsers have gradually become more and more complex, a wide range of access permissions is needed to support their functionality. We want to enable a security-concerned user to limit its privileges, and thus reduce potential damage, and so our policy offers four modes of using a web browser. It is up to the user to decide which mode will be used as the default and when to switch to another mode.

In all of the following modes, web browsers can communicate over web protocols.

General web browsers

General web browsers have full access to all non-sensitive user data. They do not have access to webcam or microphone devices.

Restricted web browsers

Restricted web browsers can only access a small subset of user data (typically a Downloads folder, but the user can change this configuration). They do not have access to most of the user data, nor to webcam or microphone devices. This mode can be used for browsing untrusted websites.

Trusted web browsers

Trusted web browsers have full access to all non-sensitive user data, and they can also read sensitive user data. They do not have access to webcam or microphone devices. This mode can be used when the user needs to use a locally stored private key to authenticate for a remote server, or to digitally sign an email (when using webmail).

Unlimited web browsers

Unlimited web browsers are granted all permissions needed to support the browser functionality, including full access to non-sensitive user data, to webcam and microphone devices.

5.3.8 Mail clients

This class of applications is dedicated solely to mail clients, such as *thunderbird* or *evolution*. For the basic functionality, mail clients require network access to communicate over email protocols (SMTP, POP3...). Similarly to the web browsers, there are three

modes with different privileges to be used with mail clients. It is up to the user to choose the default mode and the proper mode for each situation.

In all of the following modes, mail clients can communicate over mail protocols. The mail clients are never allowed access to webcam or microphone devices.

General mail clients

General mail clients have full access to all non-sensitive user data. They can communicate over web protocols (e.g. to download resources from external servers, to correctly display images included in email message) and execute a web browser upon clicking on a hyperlink.

Restricted mail clients

Restricted mail clients can only access a small subset of user data (typically a Downloads folder, but the user can change this configuration). They do not have access to most of the user data, nor can they communicate over other than mail protocols or execute web browsers.

This is a paranoid mode that can, for example, mitigate some forms of phishing.

Trusted mail clients

Trusted mail clients have full access to all non-sensitive user data, and they can also read sensitive user data. This mode can be used when the user needs to use a locally stored private key to digitally sign, encrypt or decrypt an email message.

5.3.9 General network applications

Applications that require network access, other than over only mail and web protocols. Examples include electronic book/magazine readers (*calibre*), streaming applications (*spotify*), cloud storage services (*dropbox*) or torrent clients (*transmission-gtk*). These applications can communicate over various network protocols and have full access to non-sensitive user data.

5.3.10 Teleconferencing applications

Messaging applications (*skype, yakyak*) and other applications that require ability to communicate over various network protocols, to read and manipulate non-sensitive user data (for file sharing), and, most notably, to provide screen sharing and to access webcam and microphone devices.

Category/User data type	Config	Downloads	Sensitive	General
Simple local applications	Full	None	None	None
File viewers	Full	Read-only	None	Read-only
File editors	Full	Full	None	Full
Trusted file viewers	Full	Full	Read-only	Full
Trusted file editors	Full	Full	Full	Full
Device recorders	Full	Full	None	Full
General web browsers	Full	Full	None	Full
Restricted web browsers	Full	Full	None	None
Trusted web browsers	Full	Full	Read-only	Full
Unlimited web browsers	Full	Full	None	Full
General mail clients	Full	Full	None	Full
Restricted mail clients	Full	Full	None	None
Trusted mail clients	Full	Full	Read-only	Full
General network applications	Full	Full	None	Full
Teleconferencing applications	Full	Full	None	Full

Table 5.1: *Access to user data by application categories.*

5.3.11 Unlimited applications

Some applications may require specific privileges, and might not fall into any of the previous categories. Since we still want to ensure a smooth execution of such applications, we will provide a fallback mechanism - the `unconfined` domain from the reference policy[1]. However, it should be used rarely and wisely, since it does not provide any security guarantees.

5.4 Summary

In total, our policy will define 15 modules for user applications, which loosely correspond to categories of applications, when sorted by functionality.

The main differences between these categories are in the level of access to user data, network resources and devices, and are summarized in Table 5.1 and 5.2.

Of course, we could define other categories with different combinations of privileges. We chose this set of categories because we believe they represent the reality the most accurately.

In case that individual requirements of a user are different from our vision, the user

[1]Remember that in the original reference policy, user applications are executed in this domain by default. In our policy, we only use it when no other option is available.

Category/Protocols, devices	Web	Mail	TCP/UDP	Raw	Devices
Simple local applications	No	No	No	No	Default
File viewers	No	No	No	No	Default
File editors	No	No	No	No	Default
Trusted file viewers	No	No	No	No	Default
Trusted file editors	No	No	No	No	Default
Device recorders	No	No	No	No	Default
General web browsers	Yes	No	No	No	Default
Restricted web browsers	Yes	No	No	No	Default
Trusted web browsers	Yes	No	No	No	Default
Unlimited web browsers	Yes	No	No	No	Full
General mail clients	Yes	Yes	No	No	Default
Restricted mail clients	No	Yes	No	No	Default
Trusted mail clients	Yes	Yes	No	No	Default
General network applications	Yes	Yes	Yes	No	Default
Teleconferencing applications	Yes	Yes	Yes	No	Full

Table 5.2: *Access to network and devices by application categories.*

is advised to add a similar module to our policy, with a different, more appropriate combination of privileges. The mechanism is not difficult and is explained in Chapter 9.

Part III

Policy development

Chapter 6

SELinux policy development

In this chapter, we explain the mechanisms of a SELinux policy development, and the main parts of a SELinux policy. As stated earlier, we only consider the modular policy architecture, and so we will not provide details about monolithic policy in this chapter.

6.1 SELinux policy languages

A SELinux policy source code defines a set of policy modules. Some of them form the base policy module, the other are optional and can be loaded or unloaded as required. These module source files are compiled into *policy packages*, and combined into a single binary policy, which is then loaded into the kernel.

The SELinux policy further defines mapping between the Linux users and SELinux identities, a set of roles that the SELinux user is authorized for, the default role for each user, the default type for each role and the default contexts for some objects (such as X-server objects or DBUS objects).

Currently, there are three separate policy languages in common usage[44], which we introduce in the following sections:

- *Kernel policy language* - The original, low-level policy language.
- *Reference policy language* - Language created in M4 macros over the kernel policy language, which includes interfaces and templates.
- *CIL language* - A new language, semantically equivalent to kernel policy language that solves some of the kernel policy language problems and adds new features.

In this thesis, we will actively use the kernel and reference policy languages. We only need a passive knowledge of the CIL language, as it can be used for troubleshooting during the policy development.

6.1.1 Kernel policy language

In this section, we review the policy language statements[44] and explain which of them we will use in our policy to accomplish our goals. We will not provide the syntax of the rules. For a comprehensive reference, we advise the reader to study *SELinux By Example*[2]

Policy building blocks

Statements `class, common, inherits` are used to define object classes and permissions. We will preserve these definitions from the reference policy, as they are closely tied to the Linux kernel objects, and are not supposed to be modified by policy developers.

We will use the statements `user, role, type, typealias` to define users, roles and types, and statements `attribute, attribute_role` and `typeattribute, roleattribute` to define groups of types and groups of roles.

RBAC rules

We will use the `roles` and `types` statements to define a set of types allowed for a role, and a set of roles allowed for a user. We will use the `role_transition` statement to define possible role transitions (role changes).

DTE rules

The DTE rules constitute the majority of the SELinux policy, and so naturally will be the most frequently used rules in our policy as well.

The `allow` rules specify access allowed between two types.

For example, the following rule would allow a process running in the `unconfined_t` domain to execute a file of type `bin_t`:

```
allow unconfined_t bin_t : file execute;
```

The `neverallow` rules specify permissions that may never be granted by any allow rule. This statement is especially useful when the SELinux policy consists of several modules, possibly provided by independent creators, and when it is hard to keep track of all allowed permissions.

For example, when the following rule is used, no domain except for the `kernel_t` domain can be given the `sys_module` capability (to load kernel modules):

```
neverallow ~kernel_t self:capability sys_module;
```

Labeling rules

Statements `default_user`, `default_role`, `default_type`, `fs_use_task`, `fs_use_trans`, `fs_use_xattr`, `genfscon`, `netifcon`, `nodecon`, `portcon`, `sid` are used to define default security contexts or parts of security contexts to objects (kernel, filesystems, ports...). We will preserve these definitions from the reference policy.

In our policy, we will more often use the `type_transition` statement, which allows (with combination of other statements) to define type and domain transition rules. These rules determine the types/domains of newly created objects (e.g. a new file in a directory) or processes.

For example, the following combination of rules specifies that whenever a process running in the `init_t` domain executes a file with the `apache_exec_t` type, the new process will be executed in the `apache_t` domain:

```
allow init_t apache_exec_t:file execute;
allow init_t apache_t:process transition;
allow apache_t apache_exec_t:file entrypoint;
```

Constraint statements

The `constrain` statement with the `r1`, `r2`, `t1`, `t2`, `u1`, `u2` keywords and `==`, `!=`, `eq`, `dom`, `domby`, `incomp` operators are used to define constraints. We do not define any new constraints in our policy, we only preserve the constraints defined by the reference policy.

Conditional policy statements

We will use the `bool`[1], `if` and `else` to define conditional policies, for example:

```
if (bool_allow_execmem && bool_allow_execstack) {
 # Allow making the stack executable via mprotect.
 allow $1 self:process execstack;
}
```

Audit rules

We will use audit statements to define rules specifying which events should be logged. By default, all denied accesses are logged but it is possible to suppress logging of an event that is commonly denied or security non-sensitive using `dontaudit` rule; or log allowed operations that are security-sensitive using an `auditallow` rule.

[1] We will actually use the gen_bool macro of the reference policy language to define a boolean flag, rather than the `bool` kernel policy statement.

MLS/MCS statements

Our policy does not use the optional MLS/MCS part of the SELinux policy, so we will not use the related statements such as `category, sensitivity, level, low, level, low, low_high, low_high, default_range, dominance` or others like `mlsconstrain` and `mlsvalidatetrans`...

6.1.2 Reference policy language

The reference policy relies heavily on the M4 macro processor, as the policy interfaces, templates and other supporting services are created as M4 macros over the kernel policy language.

In this section, we will provide a few examples of the support M4 macros. More information can be found in Section 5.7 of *The SELinux Notebook*[1], or in the reference policy documentation.

Loadable module macros provide support to loadable module infrastructure - for example, to define modules (`policy_module`), or optional policies that are enabled or disabled based on whether the corresponding module is loaded or not (`optional_policy`).

Other macros allow to define boolean flags (`gen_bool`), user-role assignments (`gen_user`) or contexts assignments (`gen_context`).

Especially useful are the macros that define groups of related object classes and permissions, and common access patterns. A few examples follow.

When used in the policy `allow` rules, the `read_file_perms` macro is expanded to a set of permissions required for read access to the file:

```
define(‘read_file_perms’,‘{ getattr open read lock ioctl }’)
```

The `socket_class_set` is expanded to a set of all socket object classes:

```
# All socket classes
define(‘socket_class_set’, ‘{ tcp_socket udp_socket rawip_socket
    netlink_socket packet_socket unix_stream_socket unix_dgram_socket
    appletalk_socket netlink_route_socket netlink_firewall_socket
    netlink_tcpdiag_socket netlink_nflog_socket netlink_xfrm_socket
    netlink_selinux_socket netlink_audit_socket netlink_ip6fw_socket
    netlink_dnrt_socket netlink_kobject_uevent_socket tun_socket
    netlink_iscsi_socket netlink_fib_lookup_socket
    netlink_connector_socket netlink_netfilter_socket
    netlink_generic_socket netlink_scsitransport_socket
    netlink_rdma_socket netlink_crypto_socket }’)
```

The `create_files_pattern` expands to rules necessary to allow the specified domain (`$1`) to create a file with the specified type (`$3`) in a directory of the specified type (`$2`):

```
define(`create_files_pattern',`
 allow $1 $2:dir add_entry_dir_perms;
 allow $1 $3:file create_file_perms;
')
```

We will use these and other similar macros frequently in our policy.

6.1.3 CIL (Common Intermediate Language)

There is currently a project underway called the Common Intermediate Language (CIL) project that defines a new policy definition language[1]. CIL is meant to enable several features that are currently difficult or impossible to achieve with the current policy languages and tools, and to fix some problems of the kernel policy language, such as inconsistency of naming conventions, order dependence and dependence on M4 macros.

In general, CIL preserves the current kernel policy almost unchanged (just with a different syntax) and layers on features from the existing languages, and novel features (such as inheritance). It is designed to be an intermediate language between the low-level kernel policy representation and a high level policy language. It is possible to write a SELinux policy solely in CIL, but the future goal is to build more domain-specific high-level languages over CIL.

In this thesis, we will not use CIL language for policy development. The reference policy is implemented using the policy language and M4 macros over this language, and since our goal is to extend it, we naturally choose to preserve the coding conventions[2].

Being aware of CIL language can, however, be useful for a more experienced reader of this thesis, who will seek to customize or further extend our policy. CIL uses a LISP-inspired syntax for easy parsing, and is suitable for policy analysis. It can be used during the policy development, for identifying bugs in the policy code. We used this opportunity frequently, and provide more information in Chapter 9.

6.2 Reference policy structure

The source code and documentation of reference policy can be retrieved by installing the `selinux-policy-src` and `selinux-policy-doc` packages. The former package contains the policy definitions and rules, and `Makefile` for compilation and installation.

In this section, we provide a brief overview of the important parts of the reference policy, upon which we build our policy. More details can be found in Section 5.2 of *The SELinux Notebook*[1].

[2]The reader is, of course, welcome to study the comprehensive CIL language reference on the project website[45].

The most important parts of the policy source tree are:

- `config` directory - application configuration files (e.g. default security contexts);
- `doc` directory - documentation, templates and example files;
- `policy` directory - policy modules and configuration files;
- `support` directory - M4 support macros;
- `Makefile, Rules.modular, Rules.monolithic` - rules for building the policy;
- `users` - definition of SELinux users and role assignments;
- `constraints` - constrain rules.

Other files are generated during the compilation using definitions in the other source files, for example `booleans.conf` with all the boolean flags (name, description, default value), `modules.conf` with all the modules (whether it is a base or loadable module, whether it should be enabled or disabled).

The policy modules in the `policy` directory are logically structured into 7 layers (subdirectories). The *admin, apps, kernel, roles, services* and *system* layers contain the core policy modules, and the *contrib* layer consists of modules for third-party/optional applications.

Each of the policy module is implemented in the following three files[46]:

The `.te` file contains the SELinux policy code (type enforcement rules). The `.if` file contains functions that can be called by other modules, and that allow them to use the internal types defined in the respective module (interfaces). The `.fc` file contains mappings between file paths and security contexts, generally those defined in the `.te` file.

Some of the modules form the base policy module, the other modules are optional and can be loaded or unloaded as required. The type of the module is determined by the preamble of the interface file. In this example, `filesystem` would be compiled into the base module and `mozilla` would be a loadable module:

```
filesystem.if
## <summary>Policy for filesystems.</summary>
## <required val="true">
## Contains the initial SID for the filesystems.
## </required>
```

```
mozilla.if
## <summary>Policy for Mozilla and related web browsers.</summary>
```

The configuration files in the `config` directory are presented in three subdirectories - `appconfig-mcs`, `appconfig-mls` and `appconfig-standard`. Which set of configuration files is used, is determined by the policy type (standard/with MCS/with MLS). The reference policy uses MCS[3], so we will use the `appconfig-mcs` directory, although we do not add any special rules using the MCS mechanism.

6.3 Summary

Because we are extending a reference SELinux policy, we will naturally be using the same languages and coding conventions. We will use the statements of a kernel policy language and reference policy language, which builds upon the kernel policy language with the use of M4 macros.

CIL language will be useful for troubleshooting during the development.

In our extension of the reference policy, we define a new layer of modules (*app-groups*), implement a number of modules in this layer and define interactions between the existing and the new modules. We will build on the reference policy, follow the established conventions and use the provided `Makefile`.

[3]Multi-Category Security, see Section 2.1.7.

Chapter 7

Implementation

This chapter is dedicated to the implementation of our SELinux policy. We describe the newly added modules and other changes made in the reference policy.

7.1 Fixing the reference policy for Debian

The SELinux policy for a Debian distribution, GNOME environment, has been bugged at least since version Debian 7[28]. We tested the reference policy with Debian versions 7, 8 and 9, and encountered severe problems with booting or using the X-server. Therefore, in the first part of our implementation, we had to make changes in the existing modules of the reference policy, to fix these bugs.

7.1.1 Missing privileges

The most important issue was missing privileges of some applications. We added privileges to the following modules:

- `init` module and its interaction with `udev`, `dbus`, `cron`, `xserver`, `gnome` and `logging` modules,
- `selinuxutils` module,
- `unconfined` module.

It is important to note that our patches are generic, i.e. we added some small sets of privileges to make the system usable, without a deeper understanding why these privileges are required. But due to the nature of the affected modules (system or already highly-privileged applications), we are confident these changes do not have any impact on the security of the user applications, which are the main focus of this thesis.

7.1.2 Broken mechanism for default contexts

The second issue we encountered was a problem with setting default security contexts for users upon log on, as defined in the `config` directory of the policy source tree. Although a default role for a user is defined, as well as a default domain for a role, the latter setting is not effectively used. Instead, a so-called *failsafe* context is used for the user, which is normally used in case the context for some subject or object cannot be determined by any other settings.

We are not aware of any reason why this mechanism doesn't work. Nevertheless, we can still determine the default context of the user by changing the value of the failsafe context (in our case to `confined_r:confined_t:s0`).

The drawback of this solution is that the same role and domain will be used for all SELinux users, even though we would prefer for the common users to log into a nonprivileged role and the `root` user to a privileged role, or further distinguish between various security contexts for the user based on how he/she logged in. In this solution, all users are logged into the unprivileged role but can switch to the privileged role manually[1].

7.2 Design of the reference policy extension

In our implementation, we rely on the reference policy that already confines system data and related resources. User applications also require some level of interaction with system resources, but they do not require any special permissions such as reading `/etc/passwd` file or loading kernel modules. Therefore, we assign only a basic set of system-related privileges to all user applications, such as read access to configuration files, access to temporary directory, or ability to use terminals.

We add a new layer of modules to the reference policy that we call `appgroups`, which refers to the nature of our modules that confine groups of applications, rather than specific applications.

The layer contains modules for the categories of applications, as defined in Chapter 5, and a `helperfnc` module. The latter module defines an interface over the functions provided by the reference policy, which we then use in the other modules to define the required privileges.

The user is by default logged in with an unprivileged `confined_r` role and `confined_t` domain, but is also authorized for the privileged `unconfined_r` role and `unconfined_t` domain.

The unprivileged domain ensures that the default privileges of the user and applications are limited, even if they are not specifically covered by the SELinux policy.

[1] See Section 8.2.4 for more details on how to switch between roles.

The privileged domain allows the user to change SELinux policy configuration and do other changes in the system.

7.3 User roles and domains

The unprivileged domain determines the permissions of the user, and of any application that is executed without a domain transition (e.g. generic applications in `/usr/bin` or user home directory). Those are generally applications that are not classified into any of our categories, nor confined by any of the existing reference policy modules.

We implemented the unprivileged role (`confined_r`) using the template for a non-privileged user role provided by the reference policy (`user_r`), but we made some minor changes. The unprivileged domain is authorized for basic access to system resources (such as to read configuration files or to send logs to `syslog`) and for access to non-sensitive user data, but - as a contrary to the `user_t` domain - it is restricted from using network resources, accessing private user data, executing untrusted files and adjusting SELinux configuration.

These precautions ensure that generic applications have only limited permissions, unless assigned to a category of applications with the appropriate set of privileges. In order to manipulate the sensitive user data or access network, either the user must execute an application within the respective domain, or must switch to a privileged role.

It is expected that the user will commonly switch between the privileged and unprivileged role, in the same way that he/she uses `sudo`. As a rule of thumb, all privileged operations that require the use of a sudo command will now have to be carried out in the privileged domain. To prevent frequent switching between the two roles, the user can execute one shell in the privileged domain, another shell in an unprivileged domain, and use them as required.

7.4 Helper functions for new modules

To simplify implementation of modules for application groups, we created a module called `helperfnc`, in which we implemented helper functions for the other modules.

7.4.1 Interface over the reference policy

The helper functions mostly constitute a simplified interface over the reference policy.

The reason behind this step was that the functions provided by the modules of the reference policy are often too detailed, for example, in order to grant the read privileges to the objects in a `temp` directory, one may need to call separate functions to allow

the access to directories, to files, to symbolic links, sometimes even named sockets and named pipe of the `tmp_t` type.

We grouped such functions regarding the same type and access vector, and created a number of simplified functions. We provide a few examples, but the user is advised to study the documentation of the module for more details[2].

Example 1

Read access to `/var` directory.

```
# Read generic files in /var, /var/lib
# Types: var_t, var_lib_t
interface(`helper_read_var',`
 files_list_var($1)
 files_read_var_files($1)
 files_read_var_symlinks($1)
 files_list_var_lib($1)
 files_read_var_lib_files($1)
 files_read_var_lib_symlinks($1)
')
```

Example 2

Privileges to use network ports and send packets associated with web protocols.

```
interface(`helper_network_web',`
 corenet_sendrecv_http_client_packets($1)
 corenet_tcp_connect_http_port($1)
 corenet_tcp_sendrecv_http_port($1)
 corenet_sendrecv_http_cache_client_packets($1)
 corenet_tcp_connect_http_cache_port($1)
 corenet_tcp_sendrecv_http_cache_port($1)
')
```

Example 3

Example of a helper function that does not use the existing functions of the reference policy modules. This part of the function allows the basic IPC communication between the processes of the same domain.

```
# Basic interaction (~ safe for all modules)
interface(`helper_self_permissions',`
 # IPC objects (unlimited access to objects of the same type)
 allow $1 self:shm create_shm_perms;
 allow $1 self:sem create_sem_perms;
 allow $1 self:msgq create_msgq_perms;
 allow $1 self:msg { send receive };
 ...
')
```

[2]See Appendix B.

Type name	Description
user_home_t	Default type for objects in the user home directory.
user_home_config_t	Type for objects in hidden directories in the user home directory (e.g. `~/.config/` directory).
downloads_t	Type for objects in the download/upload directories, which are the only directories accessible to a restricted web browser and a mail client (e.g. `~/Downloads/`).
private_t	Type for private user data (e.g. in the `~/Private` directory).

Table 7.1: *Security types for files in the user home directory.*

7.4.2 Classification of user data

The reference policy by default defines a single type for filesystem objects located in the user home directory - `user_home_t`. However, we need a more fine-grained control over the files in this directory.

For example, the category of simple local applications (`local_restricted_t`) does not require access to user data, but does require access to user configuration directories such as `~/.local/share`, `~/.config` or `~/.cache`. In the reference policy, all of these files have the same type so we could not fulfill both of these requirements at the same time.

We solved this problem by creating new labels for objects in the user home directory, as listed in Table 7.1.

To preserve compatibility with the other modules of the reference policy, we defined an attribute `user_home_generic_content_type` and replaced all occurrences of `user_home_t` with this attribute.

We further defined new types for bash configuration files and bash history file, in order to put restrictions on access to these files, and thus to comply with our security requirements.

All of these changes apply to the `userdomain` module.

7.5 Added modules

We added 15 new modules into the policy, that implement the requirements of the categories of applications (*appgroups*), as defined in Chapter 5. These modules rely on the interface of the `helperfnc` module.

In this section, we describe the groups of privileges that we granted to these modules (in various combinations, based on their functionality requirements). We also describe

possible interactions between our newly-defined domains and types.

7.5.1 Basic access

All of our domains can be executed by both privileged (`unconfined_t`) and non-privileged (`confined_t`) domain. They all have privileges to use IPC objects, manipulate files within their domain or send signals to processes of the same domain.

All of our domains are given the basic access to system files, which includes ability to read generic files in `/usr`, `/etc`, `/tmp`, `/var`, `/sysfs`, `/proc`, retrieve filesystem atrtibutes, read sysctls, manage temporary files and `tmpfs` files, manage locks and user configuration data, send messages to `syslog` and execute generic binary files without domain transition.

7.5.2 Access to user data

Our policy largely focuses on protecting the user data, so our domains are given a varying level of privileges with regards to the data in the user home directory and on removable drives.

In the restricted level, the domains only have access to user configuration data. Some domains can only access files in the download/upload folders. Most of the domains have a full or readonly access to non-sensitive user data, and only trusted domains can access or even modify sensitive user data.

7.5.3 Access to network

Unless necessary, our domains have no access to network. Web browser domains only have access to web protocols, mail clients only to mail protocols, and sometimes to web protocols.

The domain for general network applications has full access to network via TCP/UDP protocols, but only unconfined domain can use raw sockets.

7.5.4 Access to devices

Basic access to devices given to all domains consists of access to terminals, ability to use a printer device and speakers.

When necessary, the domains are also given access to the webcam and microphone devices. Only the unconfined domain has full access to all devices (such as raw disks).

7.5.5 Interaction with contrib modules

Whenever necessary, the helper functions define possible interactions with modules from the `contrib` layer, for example, the trusted applications are allowed to interact with the `gpg` module, which also deals with private user data.

7.5.6 Interaction between new modules

The existing modules of the reference policy often define a number of types that are used by the respective application. For example, the `samba` module defines type `smbd_exec_t` for the daemon executable file, `smbd_t` for the corresponding application domain, and a couple of other types that are used to label private configuration files, log files etc (`samba_log_t`, `samba_var_t`, `samba_var_run_t`, `samba_etc_t`, `samba_secrets_t`, `samba_unit_t`...)

Even though this approach allows to define complex policies, it may cause confusion to the user, who will likely have a hard time understanding which type should be used in which situation.

We decided for a simplified solution where we use only two or three types for each module - a type for the application domain (with which all the privileges are associated); a type for an executable file that determines the entrypoint to the domain; and a type for all other associated files. Sometimes, we even use the same type for the application domain and file type. All types that we defined for our new modules, are listed in Table 7.2.

As for the interaction between the modules, the application domains are not authorized to modify files of types defined by the other modules. However, in favour of usability, we added an option to execute the files defined by the other modules, but always in the less-privileged of the two domains.

For example, when a web browser (`browser_general_t`) executes a calculator (`local_restricted_exec_t`), domain transition to `local_restricted_t` domain is enforced. This is to prevent a non-trusted, non-privileged application from being executed in a domain with higher privileges (in this example, the ability to read and modify user data and the access to the network).

On the other hand, should the calculator attempt to execute the web browser, the policy will not allow the domain transition (even if requested), and the web browser will be executed in the context of the less-privileged calculator. This is to prevent a non-trusted, non-privileged application from executing highly-privileged applications, possibly with malicious inputs.

The graph in Figure 7.1 illustrates the relationships between the domains, showing which of each pair is more privileged.

Executable file type	Other files type	Application domain
browser_general_exec_t	browser_t	browser_general_t
browser_restricted_exec_t	browser_t	browser_restricted_t
browser_trusted_exec_t	browser_t	browser_trusted_t
browser_with_devices_exec_t	browser_t	browser_with_devices_t
local_general_exec_t	local_general_t	local_general_t
local_readonly_exec_t	local_readonly_t	local_readonly_t
local_restricted_exec_t	local_restricted_t	local_restricted_t
local_trusted_readonly_exec_t	local_trusted_readonly_t	local_trusted_readonly_t
local_trusted_rw_exec_t	local_trusted_rw_t	local_trusted_rw_t
mail_general_exec_t	mail_t	mail_general_t
mail_restricted_exec_t	mail_t	mail_restricted_t
mail_trusted_exec_t	mail_t	mail_trusted_t
network_general_exec_t	network_general_t	network_general_t

Table 7.2: *Types added to the reference policy. Each line in the table represents types used by one of the modules. The name of the corresponding modules are obvious from the naming convention. Some modules can share access to a single file type (web browser and mail client modules).*

Further, one of our requirements was to be able to temporarily execute a file in a different domain. For example, when the user executes a file with the `browser_general_exec_t` type, the new process will automatically transition to the domain `browser_general_t`.

But using a `runcon` command[3], we can temporarily execute the file in a different context, as long as it is allowed in the policy. An `entrypoint` permission between the file type and the desired temporary domain is required.

In our policy, we defined these permissions for the groups of domains that can be used for the same types of applications. For example, a text editor can be temporarily executed in read-only mode to process potentially malicious data from a mail attachment, or a web browser can be temporarily executed in a trusted mode to be able to use the webcam and microphone for teleconferencing. All the possible combination for temporary domain change are listed in Figure 7.2.

7.6 Optional policy parts

We implemented some parts of the policy as conditional statements, in order to allow the user to determine the level of strictness of the policy - often, it means to decide

[3]See Chapter 8.

between a usability and security.

Our optional policies deal with application whitelisting (which types of files can be executed), memory corruption prevention (ability to execute code on stack or heap) and restricting access to network resources.

For example, our policy by default does not allow execution of files located in the download/upload folders, in order to prevent accidental execution of untrusted files downloaded from the internet. Furthermore, the policy does not allow any network access to the default domain of the user. Both of these settings can, however, be changed by the policy boolean flags.

All added boolean flags with their default values and meanings are listed in Table 7.3.

7.7 Summary

We extended the reference SELinux policy by a new layer of 16 modules - a helper module serving as an interface over the existing policy, and 15 application group modules bulding upon the interface.

Since the reference policy is currently not working on a Debian desktop system, we also added a few patches necessary to fix its issues.

In our policy, a user always logs in into an unprivileged role and domain, and can then switch to a privileged role and domain when necessary. This separation ensures that even if an application is not assigned to one of our application group domains, it will not be executed with unlimited privileges (as is the case in the original reference policy).

The privileged role and domain can be used for changing SELinux settings, or general system configuration, or any other operation in which the user does not wish to be limited by SELinux. The role change can, however, only be made after an authentication, which prevents malicious code running in the context of the user to switch to a more privileged role.

Boolean name	Default	Description
appgroups_allow_execmem	True	Ability of the new domains to execute memory mappings.
appgroups_allow_execstack	True	Ability of the new domains to execute code on stack.
appgroups_allow_execheap	True	Ability of the new domains to execute code on heap.
apprgroups_exec_shell	True	Ability of the new domains to execute a shell.
appgroups_exec_downloads	False	Ability of all domains to execute files in the download/upload directory.
appgroups_exec_all_readable_files	False	Ability of the new domains to execute all files to which they have read access (e.g. temporary files).
appgroups_exec_all_executable_files	False	Ability of the new domains to execute all executable files (from the perspective of SELinux).
appgroups_read_app_private_files	False	Ability of the new domains to read private files defined by specific categories, e.g. not only generic files in user home directory (`user_home_t`), but also files in the user home directory owned by mozilla (`mozilla_home_t`).
appgroups_network_for_default_role	False	Allow network access to the unprivileged domain (TCP/UDP, not raw).
appgroups_servers_for_network_general	True	Allow network applications to create and run servers, apart from creating connections with remote servers.
appgroups_allow_gpg_for_default_domain	False	Allow access to gpg utility and related files to the default domain.

Table 7.3: *Boolean flags added to the policy, determining optional sets of privileges for the application group domains.*

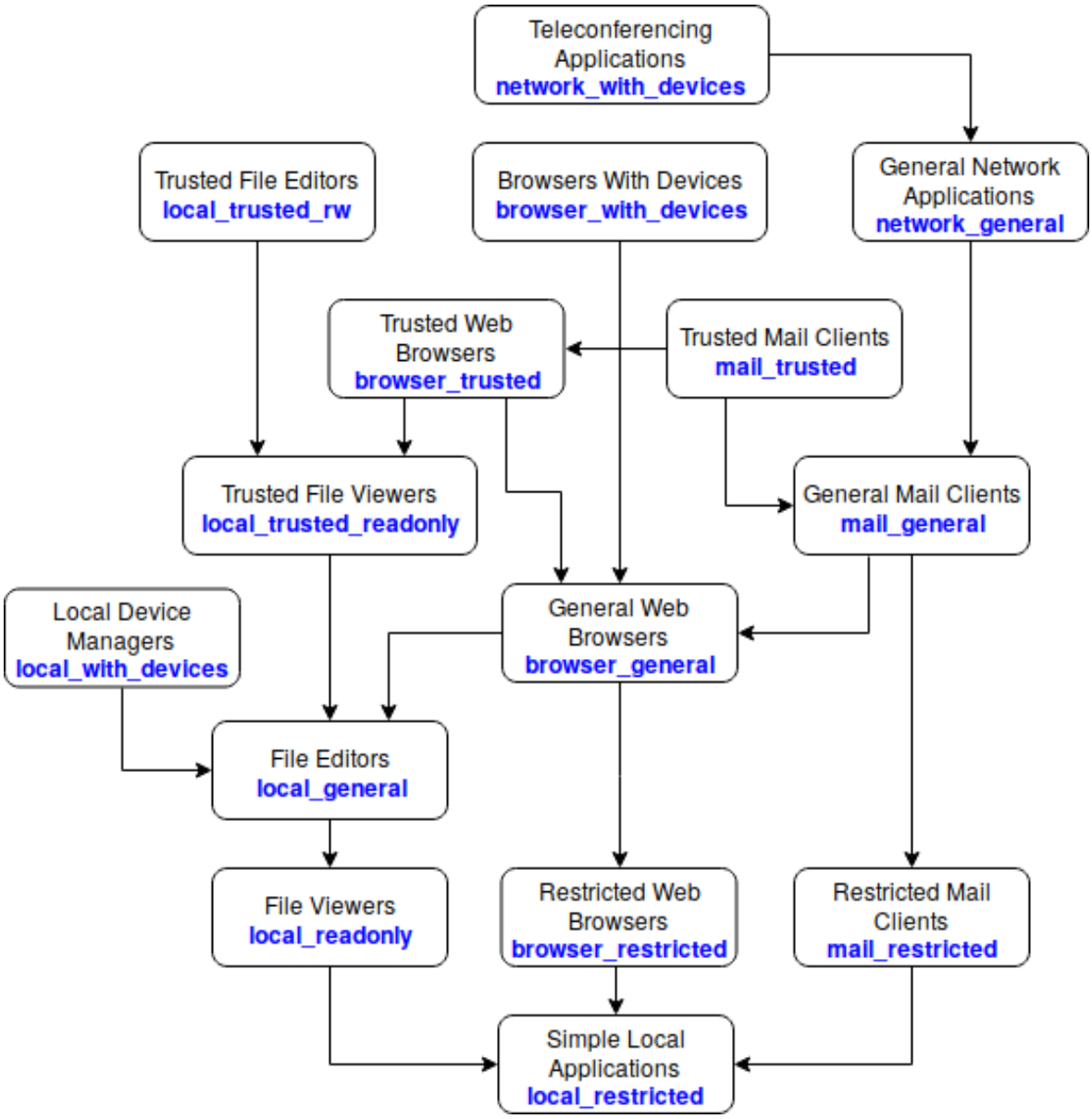

Figure 7.1: *Topological ordering of the newly-created modules, based on their privileges. There is a directed path in the graph from module A to module B precisely when the privileges of module A are a superset of privileges of module B.*

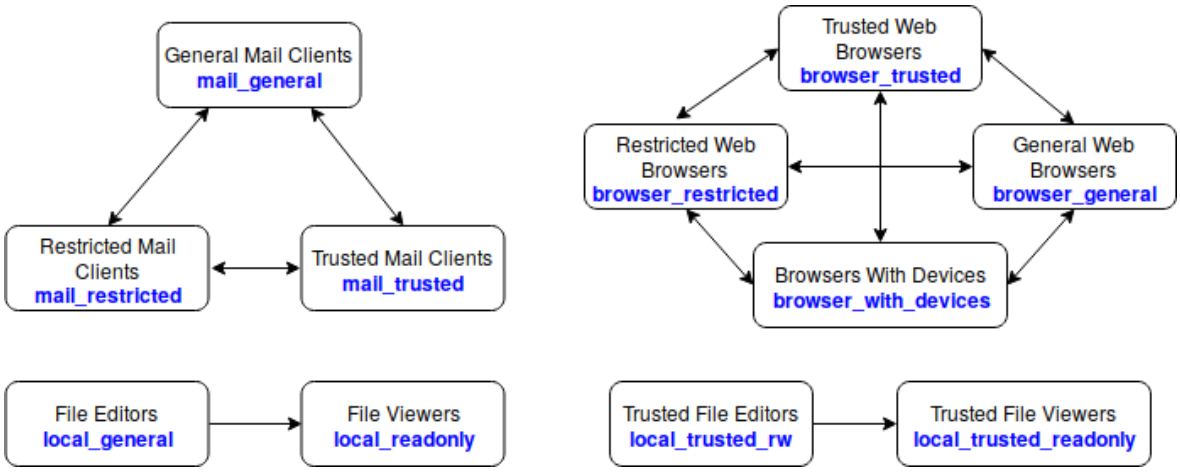

Figure 7.2: *Pairs of file types, which can be interchangeably executed in all the underlying domains, when using a runcon command. There is a directed edge between two modules when the former module is allowed to be executed in the domain of the latter module.*

Part IV

Deployment and maintenance

Chapter 8

SELinux setup and administration

In Section 2.1, we introduced the main SELinux concepts and mechanisms that are used to restrict operations on the system. As already mentioned, it is a MAC mechanism, and that requires some additional complexity in using and administering the underlying system.

In this chapter, we provide a detailed tutorial on how to enable SELinux on a system, how to configure it with our policy, and introduce tools for handling common problematic situations.

We tested the policy, and this tutorial, on a Debian 9.8 system with GNOME desktop environment[1]. For other distributions and desktop environments, there might be differences in tool/package names; and it must be checked whether the system is using a SELinux capable kernel and filesystem.

In this section, we use the following convention in the listings - `$` marks a start of a command to be executed in the user shell, `#` in a root shell, `//` indicates a comment and the other lines contain command outputs.

8.1 Enabling SELinux

In this section, we describe how to install and enable SELinux with our policy[47]. Administrator privileges are required.

The installation consists of the following steps:

1. Install the basic set of SELinux utilities.
2. Obtain the compiled SELinux policy.
3. Activate SELinux - configure GRUB and PAM and schedule a file system relabeling.

[1]Release: 4.9.0.8-amd64, kernel version: #1 SMP Debian 4.9.144-3.1 (2019-02-19), architecture: x86_64

4. Reboot the system. Automatic relabeling will occur, and then the system will be rebooted again. This step might take quite some time.
5. Check whether the installation was successful.
6. Customize the installation.
7. Switch SELinux to enforcing mode.

8.1.1 Installing SELinux utilities

The basic selinux tools are included in the `selinux-basics` package. SELinux further uses `auditd` for logging. It is not required to install this package but it is recommended, as it is necessary for effective troubleshooting.

```
# apt-get install selinux-basics auditd selinux-policy-default-
   selinux-policy-dev-
```

Note: We do not install the package `selinux-policy-default` because it contains the reference SELinux policy. We will install the policy manually in the next step. We do not install the package `selinux-policy-dev` since it comprises tools for SELinux policy development, which are not necessary for an ordinary SELinux user.

Other packages with SELinux utilities should be installed automatically with this command as recommended packages. If not, the user should also install the packages `selinux-utils`, `setools`, `setools-gui`, `python3-setools`, `policycore-utils` and `policycoreutils-python-utils`.

It is essential to install the `newrole` package, to be able to switch between SELinux roles. With our policy, a user is by default logged in with an unprivileged role, which is not authorized for doing any SELinux-related changes. Without the ability to switch into a privileged role, the user can easily lock himself/herself out from the system:

```
# apt-get install newrole
```

8.1.2 Obtaining SELinux policy

The reference SELinux policy can be obtained from the official sources, by installing one of the packages `selinux-policy-default` or `selinux-policy-mls` on Debian. Doing so installs the policy and configuration files to `/etc/selinux/default` or `/etc/selinux/mls`, respectively.

This is not our case, however, as we want to use our own policy. The policy can be found in Appendix A.

To install the compiled policy to the configuration directory (`/etc/selinux/`) and the policy packages to the policy store (`/usr/share/selinux` or alternatively `/var/lib/selinux`), run the following script:

```
# %APPENDIX%/thesis-release/INSTALL
```

As an alternative, the policy can be compiled from the source code. This option can be used when installation of the compiled version goes wrong (for any reason), or where there is a new version of the reference policy and so our policy should be updated accordingly. In that case, the reader can apply a patch to the reference policy with the changes we made. The full source code and the patch against the reference policy can be found in Appendix B, and more information about how to compile the policy from the source code, in Section 9.3.

8.1.3 Activating SELinux

Activate SELinux using the following command:

```
# selinux-activate
```

This command makes GRUB to be aware of SELinux. It adds kernel boot parameters to make SELinux the active LSM module.

The command also creates an empty `/.autorelabel` file, which triggers relabeling of the system on the next reboot. The next step is, therefore, to reboot the system.

```
# reboot
```

The system is then automatically relabeled, i.e. all the objects are assigned security contexts, as defined in the active SELinux policy. This step might take a while.

After the relabeling is completed, the system reboots automatically.

8.1.4 Verifying SELinux installation

To verify the installation was successful on Fedora or Ubuntu, we would run the following command:

```
# check-selinux-installation
```

The command checks that everything has been setup correctly and catches common SELinux problems[47].

However, on Debian systems using `systemd` (not init scripts), this command fails before doing any relevant tests[48]. Instead, we recommend to do some checks manually. SELinux is by default configured in permissive mode, which means it does not block any operations but it logs information about attempted operations that would be denied if SELinux was configured in enforcing mode.

First, disable any `dontaudit` rules to force SELinux to log all denied operations. This is done by `semanage` command from `policycoreutils-python-utils` package:

```
# semanage dontaudit off
```

Try to execute some applications, log out and back in, reboot the system. Then, check the logs in `/var/log/audit.log` or `/var/log/audit/audit.log` to see if any of the legitimate operations would be denied. The logs are quite descriptive, but we recommend to study reference materials[49] to understand them.

As an alternative, the logs can be read in more user-friendly form when `ausearch` command from `auditd` package is used. See Section 8.3 for more details.

The `dontaudit` rules should then be turned back on, in order to prevent overloading the logs with security-insignificant entries:

```
# semanage dontaudit on
```

8.1.5 Customizing the installation

To change some user-specific settings, run the post-installation script included in Appendix A:

```
# %APPENDIX%/thesis-release/POSTINSTALL
```

The script sets contexts of some files, as per the instructions of the user. It asks the user for the location of download/upload directory (`downloads_t`) and directory with private user data (`private_t`). It also sets the default values of built-in boolean flags.

8.1.6 Switching to enforcing mode

Finally, switch SELinux to enforcing mode. This can be done temporarily (until the next reboot), or permanently (by changing the corresponding kernel boot parameter). Without SELinux being switched to enforcing mode permanently, the security protections provided by SELinux are not active, therefore it is an important step.

```
// temporary change
# setenforce 1
// permanent change
# selinux-config-enforcing 1
```

8.2 Using a system with SELinux

In order to use a SELinux-enabled system, the users have to be aware of the security contexts and be able to view and modify them. SELinux is based on DTE, and so it is crucial that objects of the system are assigned correct type.

After SELinux has been enabled, filesystem objects are labeled with security contexts, as defined in the policy. Without any further changes after the installation, the

policy constrains operations of a predefined set of applications, mostly system services and applications and some user programs, while the majority of the user programs runs in the default mode.

Using a SELinux-enabled system requires assigning correct types to the newly-installed applications, and to files that require extra protection (such as private keys). Sometimes, the users have to switch between user roles, to be able to access more privileged domains.

The administrators will typically need to adjust policy configuration, and to solve most common SELinux-related problems.

In this section, we will review how these activities can be accomplished by some Linux commands and SELinux utilities. For more details, we advise the readers to study the SELinux project website[18] and the manpages of the respective commands.

Beware that some of the SELinux-related administration tasks are only allowed in the privileged user domain (`unconfined_t`), and so the user must first switch to the privileged role (`unconfined_r`) to be able to use them.

8.2.1 Using basic Linux utilities

Many Linux commands accept the `-Z` or `-context` argument to view, create, and modify the security contexts. It is very important to use these arguments, in order to avoid inconsistencies in file labeling.

We review the most useful commands with examples[50] and explain how to use them. All of these commands can be used in the unprivileged domain `confined_t`.

Viewing security contexts

Command `ls -Z` lists directory contents with the security context of each file.

```
$ ls -lZ
-rw-r--r--. 1 test test unconfined_u:object_r:user_private_t:s0 1822
   Jan 01 12:30 passwords.kdbx
```

Command `id -Z` prints the security context of the user.

```
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0
```

With the `-Z` argument, the `ps` command prints the security contexts of the running processes.

```
$ ps -aZ
LABEL                          PID TTY  TIME     CMD
system_u:system_r:init_t:s0  550 tty1 00:00:00 gnome-shell
unconfined_u:unconfined_r:browser_general_t:s0 1213 tty2 00:00:08
   firefox-esr
```

```
unconfined_u:unconfined_r:local_restricted_t:s0 1544 tty2 00:00:00
    gnome-calculator
...
```

Modifying security contexts

Incorrectly modifying the security contexts when creating/moving/copying files or directories is a common mistake of SELinux users, which leads to the most common SELinux problems[51]. It is therefore important to use the SELinux-specific arguments when using these commands.

With the `-Z` argument (or without any arguments), the `mkdir` command sets the security context of the newly created directory to the default context. The default context is determined by the context of the parent directory, e.g. `user_home_t` in the user home directory.

With the `--context` argument, the specified security context is set. In our conditions, this command is useful when the user wants to create a new directory accessible for a restricted web browser (`downloads_t`), or a new directory with sensitive data (`private_t`).

```
$ pwd
/home/test
$ mkdir dir1
$ ls -lZ dir1
drwxr-xr-x. 2 test test unconfined_u:object_r:user_home_t:s0 4096 Jan
    01 12:31 dir1
$ mkdir --context=unconfined_u:object_r:downloads_t:s0 dir2
$ ls -lZ dir2
drwxr-xr-x. 2 test test unconfined_u:object_r:downloads_t:s0 4096 Jan
    01 12:32 dir2
```

Without any arguments, the `mv` command will attempt to maintain the security context of a file that is moved to a different directory. This can lead to undesired behaviour, when the source and destination folders contain files with different security contexts.

As an example with our SELinux policy, this can lead to problems if we use the web browser in a restricted mode. Let's assume the user wants to upload a photo to a website, using the web browser. The photo is located in the `/home/user/Pictures` directory and labeled with the `user_home_t` type. The user first moves the file to the directory to `/home/user/Downloads` directory, which contains files with the `downloads_t` type.

If the user moves the file using the `mv` command without any argument, the file retains its security type of `user_home_t`. The web browser will not be able to read

this file, because it is not allowed to read generic files with in the user directory (with `user_home_t` type), only the files with the `downloads_t` type.

The correct handling of this situation is to use the `-Z` argument when moving the image file. With this argument, the `mv` command sets SELinux security context of destination file to the default type in the destination folder.

For brevity, we only display the *type* portion of the security context in the following example.

```
$ ls -lZ /home/test/Downloads
-rw-r--r--. 1 test test downloads_t     15 Jan 6:00 image
$ ls -lZ /home/test/Private
-rw-r--r--. 1 test test user_private_t 182 Jan 7:00 passwords1
-rw-r--r--. 1 test test user_private_t 154 Jan 8:00 passwords2

$ mv /home/test/Private/passwords1 /home/test/Downloads
$ mv -Z /user/test/Private/passwords2 /home/test/Downloads

$ ls -lZ /home/test/Downloads
-rw-r--r--. 1 test test downloads_t     15 Jan 6:00 image
-rw-r--r--. 1 test test user_private_t 182 Jan 7:00 passwords1
-rw-r--r--. 1 test test downloads_t    154 Jan 8:00 passwords2
```

The `cp` command is similar to `mv` and `mkdir`. With `-Z` argument, the security context of the destination file is set to the default type in the destination folder. With `-context` argument, the security context is set to the specified value.

The behaviour without any modifiers is slightly different to that of `mv` command. If the destination file already exists, the new file will maintain the context of the previous file[51]. Otherwise, the security context of the source file will be preserved, just like with the `mv` command.

8.2.2 Changing policy configuration

Our SELinux policy has builtin a number of if/then/else rules called *booleans* that allow users to tweak the predefined rules to allow different access.

For example, in our policy, the user can choose whether the user applications will be able to execute code on stack. From the security point of view, this is not recommended as it can facilitate memory corruption exploitation, but a lot of legitimate applications are implemented poorly and require these privileges. By setting the `appgroups_-allow_execstack` flag, the user can choose between the security and usability. All boolean flags defined in our policy are described in Section 7.6.

To manage the values of the boolean flags, SELinux comes with a number of tools. All booleans with their values can be listed using the following commands:

```
// list all booleans with their current value
```

```
$ getsebool -a
appgroups_allow_execstack --> off
...
// list descriptions of all booleans
$ semanage boolean -l
SELinux boolean  State  Default Description
allow_execstack  (off  ,   off) Allow appgroup executables to make
   their stack executable. This should never, ever be necessary.
   Probably indicates a badly coded executable, but could indicate an
    attack. This executable should be reported in bugzilla.
...
```

To permanently or temporarily change boolean values in SELinux policy, `setsebool` command can be used:

```
// temporary change, <value> corresponds to 0 (disable) or 1 (enable)
$ setsebool <boolean_name> <value>
// permanent change, <value> corresponds to 0 (disable) or 1 (enable)
$ setsebool -P <boolean_name> <value>
```

Commands `getsebool` and `setsebool` are included in the `policycoreutils` package, and command `semanage boolean` in the `policycoreutils-python-utils` package.

The `getsebool` command can be used in the unprivileged user domain, but the `setsebool` command can only be used in the privileged domain.

8.2.3 Setup for new applications

The SELinux policy defines security contexts for system objects, such as kernel, IPC objects, files (configuration files, log files, libraries...), filesystems or network ports. Policy modules also define contexts for specific applications, and their related objects (configuration files, user data...).

These definitions are included in the `.fc` policy source files and are based on file paths, for example:

```
/usr/bin/tor -- gen_context(system_u:object_r:tor_exec_t,s0)
```

Naturally, the list of applications with specific, non-generic hardcoded types in the policy can not be exhaustive. The security contexts for all the other applications (e.g. commercial applications, new software or software not included in the policy) must be set manually by the user. Without the change of the security contexts, the applications are executed in the default `confined_t` domain, and the associated objects are assigned default types.

The default domain is restricted and, for security reasons, does not allow all operations. To enable the whole functionality of such an application, we must assign it

to one of our application categories. The application and the associated files must be assigned a meaningful type - the one entitled to such a set of privileges, that is in accordance with the application functional and security requirements. Failure to do so might lead to SELinux blocking desired operations, or not providing the promised security guarantees.

In order to help the user choose the correct type for a newly-installed application, we created a tool `set-appgroup-contexts`. The tool is installed on a system by the post-installation script, and is only accessible from the privileged domain.

It interactively guides the user to identify the suitable application category by a series of simple questions. It is important that the user correctly fills in the path to the program executable file and directory with the associated files (for example a folder in `/opt` or `/usr/share`). In case the application does not fall into any of the supported categories, there is always the option to fallback to the unconfined domain - so as not to make the system unusable.

It is important to note that the tool is based on SELinux utilities `semanage fcontext` and `restorecon`, so it is required to have packages `policycoreutils-python-utils` and `policycoreutils` installed on the system.

Also, since relabeling is a security-sensitive operation, it is not allowed to execute the tool in the default unprivileged role. In order to use the tool, the user first must switch to the privileged role, which requires authentication, as explained in Section 8.2.4. This ensures that even if a rogue application tries to change its security context by executing the tool, it would not be successful without the consent of the user[2].

8.2.4 Temporary change of a security context

Domain change

The design of our policy allows the users to flexibly change security contexts of the applications. For example, the user can execute a text editor in a read-only domain, when processing a document downloaded from an untrusted website. In another scenario, the user can execute a mail client in a trusted domain, in order to use private key for digitally signing an email message.

This mechanism is possible using the `runcon` utility, which executes the application, command or a script in the specified domain, without permanently changing the security context of the executable file.

In the following example, the mail client `evolution` is permanently assigned a type `mail_general_exec_t`. When executed normally, the new process transi-

[2]We assume that security-concerned users are paranoid enough to not to type their passwords on-demand, without further investigation.

tions automatically to the mail_general_t domain. However, when executed with the following runcon command, the process transitions to the mail_trusted_t domain:

```
$ runcon -t mail_trusted_t evolution
```

It is important to note that the runcon command can only be used for combinations of types that are explicitly allowed by the policy. All admissible combinations for the modules of our policy are stated in Section 7.5.6.

The runcon utility is a core Linux utility, included in the coreutils package. It is available both for privileged and unprivileged user domain.

Role change

Another scenario when a user needs to temporarily change a part of his/her security context is when switching between user roles is required. Each SELinux user is assigned a set of allowed roles, and each role is authorized for a number of security domains. However, only one user role can be active at the time. The user can change the active role to have access to a different set of domains. In our policy, the user is logged into an unprivileged confined_r role, but is also eligible for a privileged unconfined_r role, which is authorized for the unconfined_t domain. In order to change the configuration of the SELinux policy, or the system in general, the user first has to switch to the privileged role.

The newrole command executes a new shell for the user, with the specified role, as long as the role change is allowed by the policy. For security reasons, running the command requires authentication of the user.

```
$ id -Z
unconfined_u:confined_r:confined_t:s0
$ newrole -r unconfined_r
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0
```

The command is included in the newrole package.

Although there are several roles defined in our policy (for backwards compatibility), only three of them are effectively used - system_r for the system processes, and unconfined_r and confined_r for the users. Our policy allows switching from unconfined_r to confined_r role, and vice versa.

Type change

Finally, the user will sometimes need to temporarily change a security context of a file. In our policy, this can happen when the user wants to upload a file from the home directory using a restricted web browser, which only has access to files of downloads_t

type. The user can either copy the file into the `Downloads` folder, or temporarily change the context of the file without copying it, only to restore the original context after the file has been uploaded.

In the latter case, when we do not wish to define a permanent rule in the policy, we use the `chcon` command to add a temporary labeling rule, and a `restorecon` command to actually change the security context (using this rule).

```
// Temporarily change the type of a file
# chcon -t downloads_t /home/test/Pictures/IMG89.png
// Change the context with the reference label (the same context as
   another file)
# chcon --reference /home/test/Downloads/sample.png /home/test/
   Pictures/IMG89.png
// Restore the context and view the changes (for both cases)
# restorecon -v /home/test/Pictures/IMG89.png
```

The command `chcon` is included in the `coreutils` package, `restorecon` in the `restorecon` package. In order to temporarily change a security context of a file, the user must first switch into the `unconfined_r` role.

This change is only temporary and does not survive file system relabeling, unlike the changes made using the `semanage fcontext` command introduced in the next section.

8.2.5 Permanent change of a security context

In another scenario, the users may wish to change the security contexts permanently. For example, the policy defines type and `private_t` for directories and files with sensitive data. Only trusted applications (i.e. executed in the trusted domains) can access and manipulate these data. By default, the post-installation script creates a `Private` folder in the user home directory, and labels it with the `private_t` type.

The user can, however, set the type of any other file or folder to `private_t`, and thus decide which data should be regarded as sensitive.

The commands in the following example would assign type `private_t` to all objects matching the specified regular expression - first, to a single file, and then to a directory with all of its content. The `semanage fcontext` tool adds this entry to the active policy, and `restorecon` uses the updated policy to recover correct labels in the specified directory.

```
# semanage fcontext -a -t "private_t" "/home/test/passwords.kdbx"
# restorecon -v /home/test/passwords.kdbx
# semanage fcontext -a -t "private_t" "/home/test/Passwords(/.*)?"
# restorecon -R -v /home/test/Passwords
```

The `semanage fcontext` command is included in the `policycoreutils-python-utils` package and the `restorecon` command in the `policycoreutils` package. The other useful modifiers are `-d` to delete the assignment, and `-m` to change the existing assigment.

It is important to note that relabeling of the object from the source type to a target type must be allowed by the security policy, i.e. the user cannot arbitrarily change security contexts of the programs. In our policy, relabeling is not allowed in the unpriviliged user domain `confined_t`, so the user must switch to the privileged domain by changing a role to `unconfined_r`.

8.3 Solving SELinux-related problems

Using a SELinux-enabled system will almost certainly be accompanied by occasional access denials, i.e. situations when SELinux blocks some desired operation of the user or a program. These problems can be mostly avoided or fixed by closely following the recommendations from Section 8.2, but there can be other reasons behind these problems.

In this section, we explain how to examine SELinux error messages, what are the primary causes for them, and how to address them[19].

8.3.1 Identifying the reason behind the error message

SELinux error messages originate from access denials, and are logged in `/var/log/audit.log` or `/var/log/audit/audit.log` files. The logs can be viewed directly, or using `ausearch` command. Another way how to examine the logs is by using the `audit2why` tool. This tool translates SELinux audit messages into a description of why the access was denied.

We demonstrate these tools on an example scenario where a user tries to upload a photo from the user home directory using a restricted web browser, which can only access files in download/upload directories, and so the operation is denied.

```
// Displaying recent AVC audit logs
# ausearch -m avc --start recent
----
time->Fri Apr 26 12:15:39 2019
type=PROCTITLE msg=audit(1556273739.294:806): proctitle="firefox-esr"
type=SYSCALL msg=audit(1556273739.294:806): arch=c000003e syscall=4
   success=no exit=-13 a0=7f56d916b0c8 a1=7f56d729c118 a2=7
   f56d729c118 a3=16 items=0 ppid=1547 pid=1971 auid=1000 uid=1000
   gid=1000 euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid
   =1000 tty=pts0 ses=3 comm=53747265616D5472616E7320233231 exe="/usr
```

```
    /lib/firefox-esr/firefox-esr" subj=unconfined_u:unconfined_r:
    browser_restricted_t:s0 key=(null)
type=AVC msg=audit(1556273739.294:806): avc:  denied  { getattr } for
     pid=1971 comm=53747265616D5472616E7320233231
    path="/home/test/IMG863.png" dev="vda1" ino=393672 scontext=
    unconfined_u:unconfined_r:browser_restricted_t:s0 tcontext=
    system_u:object_r:user_home_t:s0 tclass=file permissive=0
```

We marked the important parts of the entry as bold. As we can see, the process `firefox-esr` of `browser_restricted_t` domain was denied to retrieve attributes of the file `/home/test/IMG863.png` of `user_home_t` type.

The `audit2why` tools attempts to explain the meaning of such logs. Indeed, in this example, it identifies that the problem was caused by a missing type enforcement rule:

```
// Interpreting the audit logs with since the last policy reload
# cat /var/log/audit/audit.log | audit2why -l
type=AVC msg=audit(1556273732.695:783): avc:  denied  { getattr } for
     pid=1971 comm="pool" path="/home/test/seclin/README.md" dev="
    vda1" ino=393671 scontext=unconfined_u:unconfined_r:
    browser_restricted_t:s0 tcontext=system_u:object_r:user_home_t:s0
    tclass=file permissive=0

Was caused by:
Missing type enforcement (TE) allow rule.

You can use audit2allow to generate a loadable module to allow this
    access.
```

The `audit2allow` tool can even generate the necessary rules and compile a new module to fix the issues. Beware that enabling new privileges in the policy without justification is dangerous.

Following is the output of this tool in the aforementioned situation:

```
# audit2allow -b

#============= browser_restricted_t ==============
allow browser_restricted_t bash_config_t:file read;
allow browser_restricted_t bash_history_t:file read;
allow browser_restricted_t gpg_secret_t:dir { getattr read };
allow browser_restricted_t ssh_home_t:dir { getattr read };
allow browser_restricted_t user_home_t:dir write;
allow browser_restricted_t user_home_t:file read;
```

Please note that this example only shows one log entry but in reality, there will likely be an overwhelming amount of them, and it is important to learn how to identify the important parts. Chapter 2.16 of *The SELinux Notebook*[1] and other reference

materials can help as a starting point in understanding the meaning of the logs, but mastering it requires a lot of experimenting.

Labeling problems

As already mentioned, it is crucial that the security contexts of files and other objects are correctly assigned, otherwise SELinux will not function as expected.

Solution to the problem of mislabeling is to permanently or temporarily change the security context of the affected files[3].

Inappropriate application policy

Another reason for SELinux errors is the case when an application policy is not in accordance with its requirements. In our situation, a solution might require assigning the application to a more suitable domain or changing a value of a policy boolean[4].

Bugs in policy or applications

The problems occurring when using SELinux might be caused by bugs in the application, or in SELinux policy. These must be solved on a case-to-case basis.

The machine has been compromised

Sometimes, AVC logs are created because an application is compromised and tries to do something it is not allowed to do. Such an attempt would trigger a lot of AVC denial messages, that can help the user identify an ongoing attack on the system.

8.3.2 Addressing SELinux-related problems

The desired approach to facing SELinux error messages is to identify the reason for the error, and then relabel the concerned application, or to change the policy configuration.

However, there are other approaches as well, when the troubleshooting is not possible or when there is a bug in the application or in the policy.

The first approach is to modify the SELinux policy, to allow the blocked operation. An experienced user can modify the source code of the policy, following the manual in Chapter 9. A less experienced user can use automatized tool `audit2allow` which can generate policy allow rules from logs of denied operations. However, this must be used carefully (and we do not recommend doing so), as permitting further operations without justification can pose a security threat.

[3] See Section 8.2.5 and Section 8.2.4 for more details.
[4] See Section 8.2.3 and Section 8.2.2 for more details.

As the last resort, the user can switch a specific domain, or the whole SELinux into permissive mode. In this mode, the policy decisions will be logged but not enforced. This solution discards the security guarantees provided by the SELinux policy, but allows the user to further investigate the problem, while having access to the system.

```
// Prints SELinux mode (permissive/enforcing/disabled) and other
   information
# sestatus
// Temporarily switches SELinux to permissive/enforcing mode
# setenforce 0/1
// Permanently switches SELinux to permissive/enforcing mode
# selinux-config-enforcing 0/1
// Switches the specified domain to permissive mode
# semanage permissive -a <domain>
// Switches the specified domain to enforcing mode
# semanage permissive -d <domain>
```

8.4 Summary

Configuring SELinux on a system requires installing the SELinux infrastructure, enabling the SELinux policy and further adjust its configuration to the specifics of the respective user.

In order to use a SELinux-enabled system, the user must get used to using SELinux utilities, especially to correctly set security contexts of programs and other files. SELinux project provides a variety of tools to assist these activities.

Sometimes, errors in SELinux policy or policy configuration can lead to preventing the user from performing some operations. In this case, SELinux error messages and support tools can help reveal the reason behind this error, and fix it. If there is a substantial error, some parts of the SELinux policy can be deactivated, so as to make the system usable. Experienced users can solve the problems by adjusting the SELinux policy source code, which is explained in the next chapter.

Chapter 9

Policy maintenance

In this chapter, we provide some details about how we approached the SELinux policy development. Since it is not so common to write SELinux policies, there are fewer up-to-date manuals available. Thus, we believe our tips could serve as a starting point to any experienced reader of this thesis who will decide to further extend our work.

9.1 Developer tools

For a policy development and testing, one should install packages `policycoreutils-dev`, `selinux-policy-dev`, `setools`, `setools-gui` and all packages mentioned in Chapter 8.

As for the development environment, we used the *Sublime-text* editor with a customized syntax highligher[1], but any decent text editor would be more than enough, due to the nature of how SELinux is compiled and tested. It is useful if the text editor is capable of tracking references to functions, highlighting syntax and doing some basic refactoring.

We found an older project by Tresys called *SLIDE* [52] (SELinux Policy IDE), which might also be worth a try. It is an *Eclipse* plugin that comes with some additional features such as auto-completion of interface names or compile assistant, but we did not test this tool, so we cannot confirm whether it is still up-to-date.

9.2 Important files

The SELinux configuration files are located at several places on the system.

Global configuration files are located in the `/etc/selinux/` directory. The file `/etc/selinux/config` contains several entries, most importantly SELinux mode

[1] See Appendix B.

(enforcing/permissive/disabled) and SELinux policy name. There can be several policies installed on the system - the active policy will be determined by the policy name in this file. If this file is missing or corrupt, no policy will be loaded[1].

The directory `/etc/selinux/<policy_name>` contains a subdirectory `contexts` with the default contexts and a subdirectory `policy` with the actual compiled binary policy.

Other configuration files are in a so-called *policy store*, whose location can be either of these: `/etc/selinux/<policy_name>/modules` or `/var/lib/selinux/<policy_name>/modules`.

Beside other files, it contains the policy packages for policy modules, which are used for rebuilding the binary policy in the `/etc/selinux/<policy_name>/policy` directory.

9.3 Compilation

The SELinux policy is compiled with the `checkmodule` and `checkpolicy` tools, but since the reference policy `Makefile` builds on these tools, we did not use them directly.

The most important targets of the `Makefile` are the following:

- `clean` - deletes all temporary files and compiled policies;
- `bare` - does the `clean` target, and also removes configuration files and documentation, especially the list of modules to be compiled and `policy/modules/kernel/corenetwork.te` and `policy/modules/kernel/corenetwork.if` files;
- `conf` - generates configuration files in the source directory;
- `make` - compiles policy modules and policy base to policy package files (`.pp`) in the source directory; and copies the `.pp` files into policy store;
- `install, install-headers` - copies contents of the `config` directory to `/etc/selinux/<policyname>/contexts`;
- `load` - creates a binary policy file in `/etc/selinux/<policyname>/policy`, and loads the policy into the kernel.

To compile the whole policy from the source code, it is recommended to follow the tutorial in Section 8.1, but the steps in Section 8.1.2 should be replaced by compiling the policy using the following commands:

```
# make conf
# make load
```

When making changes to already compiled policy, this is not always the best approach, as the policy compilation can easily take several minutes. When we only make changes to a single module and when the changes do not influence the other modules (i.e. the changes are not made in the module interface), it is more handy to recompile and reload only this module:

```
# make <modulename>.pp
# semodule -i <absolutepath>/<modulename>.pp
```

If the affected module is a required module, then it is the base policy that must be recompiled, i.e. `<modulename> = base`. Otherwise, `<modulename>` is the name of the module, e.g. `mozilla`.

The `semodule` tool allows to load (`-i`), unload (`-r`) or list (`-l`) all loaded modules.

Finally, the third scenario which requires a slightly different compilation is adding or removing a module. The list of modules to be compiled is in the `policy/modules.conf` file, which is generated by `make conf` target but not deleted by `make clean` target, nor re-generated by `make load` target. Therefore, if we add (or remove) a policy module, the following sequence of commands should be used:

```
# make bare
# make conf
# make load
```

The same approach should be used when modifying the `corenetwork` module. In contrast with the other modules, the source files of this module are generated by M4 preprocessor from files `corenetwork.if.in`, `corenetwork.if.m4`, `corenetwork.te.in` and `corenetwork.te.m4`. This module contains policy and context assignments for network objects, for example for specific ports, and since there is a large amount of similar rules, it is reasonable to generate them using macros.

When recompiling the policy, it is sometimes useful to switch SELinux to permissive mode (`setenforce 0`), especially in cases when bigger changes have been made in the policy. Otherwise, if there is a bug in the policy, the system could stop responding and it might be necessary to boot in the safe mode and switch SELinux to permissive mode. It is therefore better to first check logs[2] after recompilation, and only then switch SELinux back to enforcing mode (`setenforce 1`).

[2]See Section 8.3.

9.4 Troubleshooting

When developing or modifying a SELinux policy, one can encounter problems during the policy compilation and loading the policy. But even if those succeed, the policy may not comply with the access control requirements, i.e. it may deny some desired operations, or otherwise behave unexpectedly. For these cases, the SELinux project provides a number of useful tools.

9.4.1 Syntactical errors

When error is encountered during policy compilation, the error output specifies the name of the file and the line with the problem. Beware that source of the problem can be hidden in a call to a function from another module interface, in which case the compiler does not nest into the function and marks the call to the function as faulty.

When error is encountered during loading the policy, the error output again specifies the number of line with the problem, but in this case, it is not the line in the source file, since the policy loader deals with the compiled policy packages. The line number is specified with regards to the `.cil` file that was generated in the process. This is when knowledge of the CIL syntax comes in handy.

To investigate the source of the problem, we first have to convert a binary policy package file (`.pp`) into a `.cil` file using a conversion utility located at `/usr/libexec/selinux/hll/pp`

```
/usr/lib/selinux/hll/pp <modulename>.pp > <modulename>.cil
```

The utility must be used manually, as demonstrated. Note that there is no manual for it, other than the following usage description:

```
Usage: pp [OPTIONS] [IN_FILE [OUT_FILE]]
Read an SELinux policy package (.pp) and output the equivilent CIL.
If IN_FILE is not provided or is -, read SELinux policy package from
standard input. If OUT_FILE is not provided or is -, output CIL to
standard output.
Options:
-h, --help print this message and exit
```

When the policy package has been converted into a file with CIL syntax, the user can search the faulty line and find the cause of the error. An example follows.

A problem occurred when trying to load a policy module:

```
# semodule -i /path/to/source/code/browser_general.pp
Re-declaration of typeattribute file_type
Failed to create node
Bad typeattribute declaration at /var/lib/selinux/default/tmp/modules
    /400/browser_general/cil:5
```

```
semodule:  Failed!
```

We use pp tool to locate the problem:

```
$ /usr/lib/selinux/hll/pp browser_general.pp > browser_general.cil
$ cat -n browser_general.cil
1 (type browser_general_t)
2 (roletype object_r browser_general_t)
3 (type browser_general_exec_t)
4 (roletype object_r browser_general_exec_t)
5 (typeattribute file_type)
6 (typeattributeset file_type (browser_general_t
   browser_general_exec_t ))
7 (roleattributeset cil_gen_require system_r)
...
```

The problem in this case was that our module redefined a type attribute `file_type` that had already been defined elsewhere in the policy.

9.4.2 Semantical errors

When the loaded SELinux policy does not comply with the expected requirements, one might need to examine the rules defined in the loaded policy.

The `seinfo` utility can be used to query information about the policy, such as number of users, user-role assignments, list of booleans etc. For example, the following command would list all types assigned to the `cert_type` attribute:

```
$ seinfo -x -a=cert_type
Type Attributes: 1
attribute cert_type;
cert_t
dovecot_cert_t
slapd_cert_t
```

The `sesearch` utility allows to query particular rules of the policy, based on the source and destination types, object classes and permissions and a rule type. For example, the following command would query all allow rules with `browser_restricted_t` source type and `downloads_t` destination type.

```
$ sesearch -A -s browser_restricted_t -t downloads_t
allow browser_restricted_t downloads_t:dir { search write create
   rename open reparent link remove_name rmdir read ioctl getattr
   setattr lock add_name unlink };
allow browser_restricted_t downloads_t:file { write create append
   rename open link read ioctl getattr setattr lock unlink };
allow browser_restricted_t downloads_t:lnk_file { write create rename
    link read ioctl getattr setattr lock unlink };
```

```
allow browser_restricted_t user_home_generic_content_type:dir {
    search open read ioctl getattr lock };
...
```

The `apol` tool is a graphical extension of the above tools, which also allows to analyze the policy (rules, users, constraints...). The tool takes the binary policy as an input file.

```
apol /etc/selinux/<policyname>/policy/policy.<ver>
```

9.5 Adding a new module

In order to further extend our work by defining a different set of privileges for a certain group of applications, one should add a new module for this group, for example a module named `mygroup`. First, three files must be created in the `policy/modules/appgroups` directory: `mygroup.te`, `mygroup.if` and `mygroup.fc`.

Template files from Appendix B can be used as a starter.

The `.te` file should define any new types and attributes. Most probably, a type `mygroup_t` will be appropriate for the files associated with the group of applications, and `mygroup_exec_t` for an executable file launching the application. To specify that by executing a file of `mygroup_exec_t` type, the created process will be executed in the `mygroup_t` domain, a domain transition interface should be added to the `.if` file. This function should then be called from other modules, with the parameter set to all domains that should have the privileges to execute the file. In our scenario, it will most probably be `confined_t` or even `unconfined_t` domain, but some other domains may also need these privileges.

The `.te` file should further contain calls to the appropriate functions from the simplified interface over the reference policy modules (`helperfnc.if`). Should the module require any extra privileges, of course it can also use functions from other modules' interfaces or define more specific rules.

Finally, the `.fc` file should define files that should be labeled with the types defined in this module.

9.6 Summary

We welcome all experienced readers of this thesis to further extend and customize our SELinux policy. We provide a documentation of our part of the policy and a template for adding new modules.

The policy can be compiled using the a `Makefile`. When problems occur, the `pp`, `seinfo`, `sesearch` and `apol` tools can be useful for troubleshooting.

Part V

Evaluation

Chapter 10

Evaluation

In this chapter, we assess the overall security and usability of a Debian system protected with our SELinux policy, and demonstrate that our policy fulfills the functional and security requirements that we defined in Chapters 4 and 5.

10.1 Security validation

We start the security validation by evaluating the overall security of the mechanism. It is important to note that once installed, the SELinux policy cannot be disabled or modified without the user's knowledge. All policy configuration operations, such as file relabeling, module loading, or a change of a role, are only allowed when the user switches to a privileged role and domain. The role change always requires the user authentication, and thus cannot be done covertly by a malicious application running in the context of the user. This claim is based on an assumption that the privileged role will be used with caution and only for limited amount of time, and the assumption that the user does not type a user password whenever prompted.

These assumptions are in accordance with our requirements on target users - trusted and security-aware.

Another argument for why this setting comes with some security guarantees is that the applications that are assigned one of the newly-defined domains, have limited privileges. They cannot make substantial changes to the system and user data, nor can they covertly transition to a more privileged domain.

Finally, the applications that are not assigned any of the new domains, are executed in the unprivileged `confined_t` domain, which limits their abilities.

As for the actual malicious techniques, we reviewed our policy from the perspective of the security requirements, defined in Chapter 4. We analysed whether our policy provides a user with options to mitigate the relevant techniques, and mention them. We present our observations on how our policy mitigates the MITRE ATT&CK techniques

in Tables 10.1-10.12. In the tables, we review all the relevant techniques and explain whether it is possible to mitigate them using SELinux (or why it is not) and whether we mitigated them in our policy (and how).

We refer to three types of domains in the tables:

- unprivileged user domain refers to the `confined_t` domain, which is the default domain for users and generic applications,
- privileged user domain refers to the `unconfined_t` domain,
- application domains refer to the newly-defined domains for the categories of applications in our policy.

In conclusion, our policy provides possibilities to fulfill our main security goals. We protect the system from being corrupted by prohibiting the application domains from making changes in system configuration. We protect the system from being misused, and protect confidentiality and integrity of user data by controlling access to sensitive and non-sensitive data, to network resources in general and to specific network protocols. Finally, we protect the privacy of the user by controlling access to webcam and microphone devices.

Limitations of our solution

In favour of simplicity and usability, all application domains have practically unlimited access to X-server objects - we preserved the `xserver` module from the original reference policy. However, the downside of this decision is that our policy does not focus on mitigating some of the collection techniques - the ability to capture pressed keystrokes, make screenshots and steal clipboard data. For example, these techniques can be leveraged by banking trojans to obtain credit card information. Mitigation of these techniques would therefore contribute to the security of the system.

Future extensions of our work should deal with this issue.

10.2 Functionality validation

We tested our policy on a Debian 9.8 system with GNOME environment, using popular user applications, both from Debian distribution and third-party applications. We tested each application with one or more suitable domains, and we report our findings in Tables 10.13-10.21.

In general, we were able to categorize popular user applications to suitable categories, and thus limit their privileges. In some cases, it was necessary to switch the policy into a less strict mode (by changing a value of a boolean flag) in favour of usability, or to sacrifice some non-essential functionality in favour of security.

Adversary Technique	Priority	Our mitigation / Note
Drive-by Compromise	-	The mitigation is beyond the capabilities of SELinux, as these techniques rely on exploiting software vulnerabilities or bugs.
Exploit Public-Facing Application	-	
Supply Chain Compromise	-	
Hardware Additions	-	The mitigation is beyond the control of SELinux, as this technique relies on physical access.
Spearphishing Attachment	-	The mitigation is beyond the control of SELinux, as these techniques take advantage of human mistakes.
Spearphishing Link	-	
Spearphishing via Service	-	
Trusted Relationship	-	These are otherwise legitimate operations, and so should not be prevented by SELinux.
Valid Accounts	-	

Table 10.1: *Mitigation of Initial Access techniques.*

Adversary Technique	Priority	Our mitigation / Note
Command Line Interface	Low	Ability to execute a shell is controlled by a global boolean flag (`appgroups_allow_exec_shell`).
Scripting	Low	
Graphical User Interface	Low	Ability to execute specific groups of files is controlled by boolean flags as a form of application whitelisting: by default, it is not allowed to execute files in download/upload folders (`appgroups_execute_downloads`) and executable files in general, unless specified otherwise (`appgroups_execute_all_executable_files`, `appgroups_execute_all_readable_files`).
Third-party Software	Low	
User Execution	Low	
Local Job Scheduling	Low	
Source	Low	The `source` command is built into the shell, and so constraining the read privileges of the command would require constraining reading privileges of the shell domain, which is usually identical to the user domain. For usability reasons, we decided not to mitigate this technique, as we do not wish to restrict the user from reading files and scripts.
Exploitation for Client Execution	-	The mitigation is beyond the capabilities of SELinux, as the technique relies on exploiting software vulnerabilities or bugs.
Trap	-	The `trap` command is built into the shell and cannot be disabled nor controlled by SELinux.
Space after Filename	-	This is not a valid technique in Linux (it refers to OSX).

Table 10.2: *Mitigation of Execution techniques.*

Adversary Technique	Priority	Our mitigation / Note
Bootkit	High	Access to raw disk is not allowed to the default user domain nor user application domains.
Kernel Modules and Extensions	High	Neither default user domain, nor application domains are allowed to manipulate kernel modules.
.bash-profile and .bashsrc	Low	Application domains are not allowed to modify files bash configuration files (`~/.bashrc`, `~/.bash_profile`, `~/.profile`, `~/.bash_logout`). The default user domain is allowed this access.
Create Account	Low	Neither default user domain, nor application domains are allowed to create user accounts.
Local Job Scheduling	Low	Neither default user domain, nor application domains are allowed to execute `at` and `cron` commands.
Web Shell	Low	Ability of running a webserver is controlled by a global boolean flag (`appgroups_servers_for_network_general`), and only allowed to network applications. Default domain can also have this privilege, if `appgroups_network_for_default_role` is also enabled.
Setuid and Setgid	-	The mitigation is beyond the capabilities of SELinux, as the technique relies on exploiting software vulnerabilities.
Browser Extensions	-	These are otherwise legitimate operations, and so should not be prevented by SELinux.
Valid Accounts	-	
Hidden Files and Directories	-	
Trap	-	The `trap` command is built into the shell and cannot be disabled nor controlled by SELinux.
Port Knocking	-	These techniques are indistinguishable from legitimate operations from the perspective of SELinux, and thus cannot be mitigated by it.
Redundant Access	-	

Table 10.3: *Mitigation of Persistence techniques.*

Adversary Technique	Priority	Our mitigation / Note
Process Injection	High	We do not allow the default user domain, nor the application domains, to `ptrace` other processes.
Sudo Caching	Low	SELinux cannot prevent misusing poor configuration of `sudo`, but we prevent malicious applications from weakening the configuration. Neither application domains, nor unprivileged user domain can edit the `/etc/sudoers` file (or any other file in `/etc`).
Sudo	Low	
Exploitation for Privilege Escalation	-	These techniques cannot be mitigated by SELinux, as they rely on exploiting software vulnerabilities.
Setuid and Setgid	-	
Web Shell	-	
Valid Accounts	-	This is an otherwise legitimate technique, that should not be prevented by SELinux.

Table 10.4: *Mitigation of Privilege Escalation techniques.*

On several occasions, the tested applications could be assigned to several of our categories. For example, `vlc` can be executed as a *File viewer* application, and the essential functionality (of playing local video and audio files) will be supported. To be able to play videos from the internet, it should be executed as a *General network application*; and if the user wishes to use it for recording videos using the webcam and microphone device, it should be executed as a *Device recorder*. Whenever such a situation occurred during in our tests, we chose to assign the application to the least privileged category supporting the basic functionality, and we stated the limitations of the application in this mode in our analysis. Nevertheless, it is up to the user to decide which of the categories are the most suitable for fulfilling his/her security and functionality expectations from that application.

Limitations of our solution

We had problems with executing `cron` and `at` with SELinux enabled, which may cause some discomfort for the user. Also, new files in the `/var/run/Network-Manager` folder are occasionally mislabeled due to a missing type transition rule in the reference policy, and the user is then required to restore the correct contexts (`restorecon` command).

Further, we encountered problems when testing booting into a recovery mode, when, due to the broken reference policy mechanism of assigning default security contexts, the root user is logged in into the unprivileged domain. This domain does not have the privileges to do the necessary setup (for example, it would require the ability to load kernel modules, which we don't want to grant to applications running in the default domain), and so the user cannot switch to the more privileged role and domain. This

Adversary Technique	**Priority**	**Our mitigation / Note**
Disabling Security Tools	High	Only the privileged `unconfined_t` domain is eligible to change SELinux configuration.
Rootkit	High	Neither application domains, nor unprivileged user domain are authorized for doing system changes.
Install Root Certificate	High	Only trusted application domains have access to modifying certificates.
Process Injection	High	We do not allow the default user domain, nor the application domains, to `ptrace` other processes.
Clear Command History	Low	SELinux cannot prevent the applications from setting the HISTCONTROL and HISTFILE environment variables. However, we control access to the `~/.bash_history` file - the default user domain is allowed to modify it but application domains are not.
HISTCONTROL	Low	
File Permissions Modification	Low	We control the ability of application domains to read and modify file permissions (on a case-to-case basis).
Indicator Removal on Host	Low	Application domains are not authorized for deleting log files.
Exploitation for Defense Evasion	-	SELinux cannot mitigate this technique, as it relies on exploiting software vulnerabilities.
Valid Accounts	-	This is an otherwise legitimate technique, and so it should not be prevented by SELinux.
Space After Filename	-	This is not a valid technique in Linux.
Binary Padding	-	Mitigation of these techniques is beyond the capabilities of SELinux, because they are all aimed at other layers of security protections, and appear legitimate from the SELinux perspective.
File Deletion	-	
Hidden Files and Directories	-	
Indicator Removal from Tools	-	
Masquerading	-	
Obfuscated Files or Information	-	
Port Knocking	-	
Redundant Access	-	
Scripting	-	
Timestomp	-	
Web Service	-	

Table 10.5: *Mitigation of Defense Evasion techniques.*

Adversary Technique	Priority	Our mitigation / Note
Bash History	High	Application domains are not authorized to read the `~/.bash_history` file. For usability reasons, both privileged and unprivileged domain is allowed this access.
Credential Dumping	High	Entries in `/proc/<pid>` directory have the security contexts determined by the domain of the running process, for example entries of a trusted file editor application will be of `local_trusted_rw_t` type. Therefore, it is not possible to dump a process memory using this mechanism by another application, unless it is authorized for reading files of this type. As Figure 7.1 depicts, more privileged domains can read files of less privileged types, but not vice versa. If used properly, this should eliminate the possibility to dump credentials being processed by more trusted applications.
Credentials in Files	High	Only trusted application domains are allowed to manipulate files with credentials/keys.
Private Keys	High	
Network Sniffing	High	Only applications with unlimited access to network can use raw network devices.
Input Capture	High	Mitigation of these techniques is possible with SELinux but remains an open problem for future extensions of this thesis.
Two-Factor Authentication Interception	High	
Brute Force	-	This technique appears legitimate from the perspective of SELinux.
Exploitation for Credential Access	-	SELinux cannot prevent exploitation of software vulnerabilities.

Table 10.6: *Mitigation of Credential Access techniques.*

Adversary Technique	Priority	Our mitigation / Note
Network Sniffing	High	Raw access to network objects is only allowed to unlimited network applications.
File and Directory Discovery	High	Only trusted application domains can list sensitive files. For usability reasons, all domains are allowed to list (not read) non-sensitive files.
Account Discovery	Low	All of these techniques have otherwise legitimate use. In order not to disrupt normal activity of a user, we do not employ any active measures to mitigate these techniques.
Browser Bookmark Discovery	Low	
Network Service Scanning	Low	
Password Policy Discovery	Low	
Permission Groups Discovery	Low	
Process Discovery	Low	
Remote System Discovery	Low	
System Information Discovery	Low	
System Network Configuration Discovery	Low	
System Network Connections Discovery	Low	
System Owner/User Discovery	Low	

Table 10.7: *Mitigation of Discovery techniques.*

Adversary Technique	Priority	Our mitigation / Note
Remote File Copy	High	Only network application domains are allowed to use network, but even those do not have unlimited access to all protocols (e.g. web browsers and mail clients can only use web and mail protocols, not SMB or FTP).
SSH Hijacking	High	Only trusted application domains are allowed to read data from credential stores.
Application Deployment Software	-	SELinux cannot mitigate these techniques, as the rely on software vulnerabilities or leaked user credentials.
Exploitation of Remote Service	-	
Remote Services	-	
Third-party Software	-	

Table 10.8: *Mitigation of Lateral Movement techniques.*

Adversary Technique	Priority	Our mitigation / Note
Audio Capture	High	We limit access to microphone device.
Data from Information Repositories	High	We limit read access to user data on local and removable drives.
Data from Local System	High	
Data from Removable Media	High	
Data from Network Shared Drive	High	Only domains for network applications are authorized for using network, and by extension for reading data on network shared drives.
Input Capture	High	Mitigation of these techniques is possible with SELinux but remains an open problem for future extensions of this thesis.
Clipboard Data	High	
Screen Capture	High	
Automated Collection	-	These techniques refer to coding practices, that generally cannot be detected nor mitigated by SELinux.
Data Staged	-	

Table 10.9: *Mitigation of Collection techniques.*

Adversary Technique	Priority	Our mitigation / Note
Encrypt Files for Ransom	High	We control read/write access to user data.
Wipe Device Data	High	We control delete access to system and user data.

Table 10.10: *Mitigation of Effects techniques.*

Adversary Technique	Priority	Our mitigation / Note
Exfiltration Over Alternative Protocol	High	Our policy limits the ability of user domains to use standard and non-standard network protocols.
Exfiltration Over Command and Control Channel	High	
Exfiltration Over Other Network Medium	High	
Exfiltration Over Physical Medium	High	We limit the privileges of application domains to use removable media and other devices.
Automated Exfiltration	-	These techniques refer to coding practices of the malware authors, and generally cannot be detected nor mitigated by SELinux.
Data Compressed	-	
Data Encrypted	-	
Data Transfer Size Limits	-	
Scheduled Transfer	-	

Table 10.11: *Mitigation of Exfiltration techniques.*

Adversary Technique	Priority	Our mitigation / Note
Commonly Used Port	High	Our policy limits the ability of user domains to use standard and non-standard network ports and protocols.
Custom Command and Control Protocol	High	
Remote File Copy	High	
Standard Application Layer Protocol	High	
Standard Non-Application Layer Protocol	High	
Uncommonly Used Port	High	
Remote Access Tools	High	Ability to use legitimate remote access tools by our application domains would require specifically allowing domain transition to a more privileged RAT domain. We do not allow such a transition in our policy.
Communication Through Removable Media	High	We limit access to removable media.
Connection Proxy	-	These techniques refer to coding practices of the malware authors, and generally cannot be detected nor mitigated by SELinux.
Custom Cryptographic Protocol	-	
Data Encoding	-	
Data Obfuscation	-	
Domain Fronting	-	
Fallback Channels	-	
Multi-Stage Channels	-	
Multi-hop Proxy	-	
Multiband Communication	-	
Multilayer Encryption	-	
Port Knocking	-	
Standard Cryptographic Protocol	-	
Web Service	-	

Table 10.12: *Mitigation of Command and Control techniques.*

Application	Functionality	Note
gnome-calculator	Calculator	Financial mode is not working (it cannot retrieve currency rates).
gnome-calendar	Calendar	No problems with the functionality.
gnome-todo	TODO list manager	
/usr/games/*	Simple local games	

Table 10.13: *Tested simple local applications.*

Application	Functionality	Note
totem	Video/audio player	The `appgroups_allow_execmem` flag must be enabled. In this mode, it is not possible to play audio/video from the internet.
vlc		In this mode, it is not possible to record audio/video nor play audio/video from the internet.
eog	Image viewer	It is possible to view and print images - editing is not possible in this mode.
evince	PDF viewers	In this mode, it is possible to view and print documents. To be able to send documents via email or save a copy of the document, more privileged categories must be used.
okular		
xpdf		The `appgroups_exec_shell` flag must be enabled.

Table 10.14: *Tested file viewers.*

issue needs to be addressed by fixing the broken mechanism in the reference policy (or, more likely, in Debian).

Finally, the reference policy does not concentrate on graphical applications, and therefore some of the existing modules may be missing support for their graphical versions. For example, the `ssh-keygen` tool for generating SSH keys works without problems in the text mode, but to be able to use this tool in a graphical version (with `seahorse` password/key manager), we had to add some privileges to the tool.

In conclusion, we are confident that our modules define privileges that are in accordance with functional requirements of the associated application groups, but we cannot guarantee the same for reference policy modules, as testing the existing 200+ reference policy modules is beyond the scope of this thesis.

10.3 Summary

Our policy successfully fulfills our security goals and mitigates most of the relevant MITRE ATT&CK techniques. In contrast with the reference SELinux policy, our

Application	Functionality	Note
pinta	Paint application	The `appgroups_allow_execmem` and `appgroups_exec_shell` flags must be enabled.
inkscape	Paint applications	It is not required to enable any boolean flags.
gimp		
sublime-text	Document editors	The `appgroups_exec_shell` flag must be enabled.
libreoffice		
texstudio	Document generator	It can only be used as a local application, as long as all necessary packages are already installed. Otherwise, it must be executed as a general network application.

Table 10.15: *Tested file editors. The same applications could also be used as trusted file editors, file viewers and trusted file viewers, as required.*

Application	Functionality	Note
keepass2	Password manager	The `appgroups_allow_execmem` and `appgroups_exec_shell` flags must be enabled. Synchronization is not possible in this mode. The password databases should be stored in the Private directory, to maximize their confidentiality.
seahorse	Credential store	It is possible to manage passwords and SSH keys.
openssl	Utility for encryption/decryption, digital signature creation/verification, key/certificate generation	No boolean flags are required. It is recommended to store the associated files in the Private directory.

Table 10.16: *Tested trusted file viewers and editors.*

Application	Functionality	Note
gnome-sound-recorder	Sound recorder	The `appgroups_allow_execmem` flag must be enabled.
vlc	Video/audio recorders	No boolean flags have to be enabled.
cheese		
kazam		

Table 10.17: *Tested device recorders.*

Application	Note
firefox-esr, iceweasel	It is necessary to disable `mozilla` module from reference policy. The `appgroups_allow_execmem` flag must be enabled.
konqueror	The `appgroups_allow_execmem` flag must be enabled.
chromium	It is necessary to disable `mozilla` module from reference policy. The `appgroups_allow_execmem` flag must be enabled. It only works with the `-no-sandbox` option, since the Chrome sandbox requires severe privileges, such as `chown_dac_override`, `net_raw` or `sys_admin` capabilities.
epiphany-browser	The `appgroups_allow_execmem` flag must be enabled.
opera	Cannot be executed in our browser domains at all, since it requires `sys_admin` capability.
vivaldi	The `appgroups_allow_execmem` and `appgroups_exec_shell` flags must be enabled. It only works with the `-no-sandbox` option, since our modules do not allow `sys_admin` capability.

Table 10.18: *Tested web browsers.*

Application	Note
evolution	Configuration of an email account requires access to web protocols, and therefore cannot be done in the restricted mode. After account configuration, the restricted mode is a suitable option.
thunderbird	The `appgroups_allow_execmem` and `appgroups_exec_shell` flags must be enabled. Similarly to above, configuration cannot be done in the restricted mode.

Table 10.19: *Tested mail clients.*

Application	Functionality	Note
calibre	E-book reader	The `appgroups_exec_shell` flag must be enabled.
rhythmbox	Streaming application	No boolean flag must be enabled.
filezilla	File download/upload clients	The `appgroups_servers_for_network_general` flag must be enabled (because of the nature of the FTP protocol).
transmission-gtk		
atom	Code editor	The `appgroups_allow_execmem` and `appgroups_exec_shell` flags must be enabled. It cannot be executed as a local application, because it requires network access for synchronization with code repositories (and exits with error without this access).
texstudio	Document generator	Can also be executed as a local application, as long as all the required packages are installed.

Table 10.20: *Tested general network applications.*

Application	Functionality	Note
skypeforlinux	Messaging application	Messaging, video and audio calls work in this mode. However, setup of the application and logging in should be done in the unconfined domain, as it requires making system changes (installing ceritificates, ensuring persistence).

Table 10.21: *Tested teleconferencing applications.*

policy limits the privileges of user applications by default, instead of assigning them all into the unconfined domain. We provide means for a security-concerned user to determine privileges for the applications, following the least privilege principle.

We contributed to usability of the SELinux policy from the perspective of the user and administrator in that the policy modules are more transparent when compared to the reference policy modules. The user is not required to study the reference policy in detail in order to correctly use our extension. At the same time, the privileges assigned to newly-defined application domains correspond to the usability requirements defined for application groups, so our policy mostly does not limit legitimate activities of the user.

Some of our security and usability requirements remained unmet, and we list these deficiencies as a basis for future work and improvements.

Chapter 11

Discussion and remarks

In this chapter, we discuss limitations of our solution and suggest possible improvements.

11.1 Our decisions and remarks

While working on this thesis, we encountered several problems that required decisions which affected the security and usability of the solution. In this section, we explain our motivation behind these choices, and possible other solutions.

11.1.1 Extending the reference policy

In the early phase of our research, we considered a possibility to implement a SELinux policy from the scratch, rather than by extending the existing reference policy.

Creating an independent policy would allow us to further simplify the policy and make it more transparent to the other developers. We could reduce the number of types and rules defined in the policy.

However, this approach would require defining rules for the system applications and services, which was not the focus of our thesis. Extending the reference policy allowed us to build upon the work of the SELinux project team, which has focused on these categories of applications, and which has been developing and testing the policy for years.

The compatibility with the reference policy also allows a wider use in the security community.

We improved the transparency of the reference policy by defining a simplified interface over its modules. We believe this will allow our fellow developers to define new modules in a simpler way.

11.1.2 Reference policy functions

We observed that the functions already defined in the reference policy were sometimes inconsistent. For example, for some directories, access to files, symbolic links, named pipes and sockets of the directory type was defined. For other directories, only access to files was defined, as if the policy developers assumed these directories will not contain any filesystem objects other than common files. In general, the reference policy does not always define functions for accessing all combinations of objects and access vectors.

When creating an interface over the reference policy, we considered changing the reference policy functions radically, to create uniform interfaces, but we decided not to. When necessary, we added new functions to the reference policy but in general, we wanted to keep the number of changes in the reference policy in a reasonable level, so that our patch to the policy can be easily applied even after a reference policy update.

We believe that this should not have a negative impact on the security or usability of the system, as we believe the reference policy already defines all important combinations of accesses to objects and we trust it has been well tested.

11.1.3 Simplifications

We simplified rules for the added modules to access some subsystems, which we did not consider essential for the security or usability of the system.

For example, we allowed unlimited access to the dbus system and some IPC objects, as we believe that a faulty treatment of these objects could have a big impact on usability of the system. On the other hand, we believe that allowing these accesses does not have a fundamental impact on the system security.

Similarly, we did not specifically treat access to database objects, as in our scenario, none of the common user applications requires this access. We assume that services that require these access are already covered in the reference policy (for example in `postgresql` or `apache` modules).

A possible sequel to our work could define more fine-grained access control to these objects.

On a different note, we also made some simplifications when designing the optional policy decisions. The values of boolean flags defined in our policy affect all application domains. It could be worth to add more flexibility for this mechanism, i.e. to define the same set of boolean flags for each of the application domains, and thus be able to vary the settings for different applications.

11.1.4 Security and usability compromises

When implementing the rules for the groups of applications in our policy, we had to settle for several decisions that were in conflict with the ideal privileges required by an application.

For example, we made a compromise in favour of security by denying access to the network for a calculator and other local applications. They could normally require this access, for example to load the currency exchange rates, or more generally for checking updates or loading help pages.

These functions are, however, not essential for functionality of the local applications, and can be omitted. As a result, should such an application be compromised (for example by injecting malicious code into a running process), the privileges of the malicious code would be significantly limited.

A different compromise, this time made in favour of a usability, was to allow execution of memory for user applications. This is a very dangerous privilege indeed, as it facilitates memory corruption exploitation, which can lead to introducing malicious applications to the system. However, as we found out during the testing phase, a lot of legitimate applications require these privileges due to bad coding practices of their developers. These applications are not able to function properly without these privileges, and so we allowed them for the user applications.

However, we implemented an option to disable this access. We expect a security-concerned user to disable this option, even if that means that he/she will not be able to use some applications.

A more elegant solution would be to allow this access per-category, or per-application, rather than by a global decision. But since we provided a comprehensive manual on how to add a new module to our policy, we believe a security-concerned user can use it to create a new module with the appropriate combination of privileges.

11.1.5 Target audience

The target audience of our thesis (and by extension, of our policy) are security-concerned, trusted users.

Our policy defines a default, non-privileged role, in which the user processes have limited privileges. But we also define a privileged role in which the user is virtually unrestricted, and which can be used to customize the policy, or for a routine administration of the system. A user is authorized for both of these roles, and can switch between them as required.

In a case of a non-trusted user, these privileges could be easily misused - the user could always log into the privileged domain, and could even disable SELinux protec-

tions.

It is possible to define a role for untrusted users, for example for systems that are shared by a security-concerned, trusted user responsible for administration, and a number of non-trusted users. In this mode, the users should still be able to use the SELinux-enabled system, for example to change security contexts of new applications or private files, but they should not have the privileges to bypass the protections.

This extension could be a subject for a sequel work, as it is out of the scope of this thesis.

11.2 Possible improvements and future work

We recognize that our solution is not complete in the terms of providing a perfect security of a Linux desktop system, and that there are possible improvements to our work.

Also, as we mentioned on several occasion in this thesis, SELinux has an enormous security potential. It allows to define very detailed policies, which can be used to provide various levels of security.

In this section, we introduce some possible suggestions for future work.

11.2.1 Support for other Linux distributions

In our thesis, we focused on a Debian distribution, particularly on a GNOME environment. In general, the SELinux policy should also be applicable for other environments and distributions, without radical changes.

A possible extension of our work could be to adjust and test our policy in other Linux environments.

11.2.2 Analysis of other security layers

As stated in Chapters 4 and 10, SELinux itself cannot prevent all malicious techniques, as listed in the MITRE ATT&CK Framework, as some of them are directed against other layers of security protections.

It could be useful to further extend our analysis, and identify which other security tools or measures would be suitable in combination with our SELinux policy, in order to mitigate the other MITRE ATT&CK techniques (i.e. those not covered by SELinux).

11.2.3 SELinux as Intrusion Detection System

Apart from being an access control mechanism, SELinux also has a potential as a base for an Intrusion Detection System.

The SELinux policy can be configured to log security-sensitive (allowed) events, such as a remote login of a user, or modifying sensitive user files, even when they are allowed by the policy[1]. These operations can be specified in the policy, and subsequently logged whenever such a situation occurs, which can facilitate post-attack analysis.

A potential future work could also be dedicated to building a tool analysing patterns in these logs, in order to identify (and stop) an ongoing attack.

This approach could also help mitigate some MITRE ATT&CK techniques that SELinux cannot prevent.

11.2.4 Separation of privileges

Our policy concentrates on limiting abilities of specific applications or groups of applications, rather than isolating the applications from each other. It is important to note that applications of the same type (for example, General local applications) can interfere with each other, e.g. send signals to each other, or modify the configuration files of each other. This decision was made in favour of the simplicity of the policy, but comes with obvious security drawbacks.

Neither our policy provides a mechanism to separate several users from each other, as all the user data are assigned equivalent security contexts.

If configured correctly, these actions are usually prevented by other access controls, such as capabilities of file system permissions, but it would be a better solution not to rely on these mechanisms. A further extension of our work could concentrate on separation of these applications from each other with the mechanisms of SELinux.

11.2.5 More detailed policies

More experienced users could further extend our SELinux policies by modules with more detailed policies, capturing advanced security requirements.

For example, SELinux can control access to the network on the level of network packets. The Linux firewall (`iptables`) can be configured[53] to label packets with security contexts. The SELinux policy can then, in turn, define possible access to packets labeled with these contexts. For example, the firewall can assign a special label to packets sent to/received from specific domains (such as internet banking websites) and a trusted web browser can only be allowed to process packets with these labels.

Similarly, the X-server objects are worth further exploration. In our policy, we allowed practically unlimited access to the objects to all of our application domains. However, controlling access to these objects more strictly could help mitigate malicious

[1] This can be configured with `auditallow` rules.

techniques that remain uncovered by our policy - screen capture, capturing pressed keystrokes and stealing clipboard data.

11.3 Summary

Our SELinux policy addresses the problem of securing a Linux desktop environment. We designed a policy that, by our best knowledge, represents a reasonable compromise between a system security and functionality, but there are several potential improvements to our solution.

We also recognize the need for using other layers of security, as SELinux itself cannot block all stages of attack (for example, it cannot prevent software vulnerabilities or find patterns in the network traffic). We advise a security-concerned user to further explore the possibilities to achieve a better security.

Conclusion

In this thesis, we studied the problem of a secure Linux desktop environment. Our goal was to provide a mechanism for a security-concerned user to protect the system from being corrupted or misused, to protect the user privacy and the confidentiality and integrity of the user data.

We addressed these problems with the use of SELinux mechanisms, by extending a reference SELinux policy, with the focus on the user-applications part.

We implemented 17 new modules in the policy, that confine categories of user applications. We based this classification on our analysis of security and functional requirements on a system. We created most of the categories to represent classes of user applications that share the same set of security and functional requirements. Two applications that are behind the most common attack vectors - web browsers and mail clients - were treated with a special caution. We defined several modules (modes) for these applications with varying sets of privileges, and provided a mechanism to switch between these modes as required.

In the process of development, we encountered a problem with a SELinux reference policy on a Debian desktop installation. Debian developers currently do not particularly focus on providing support for SELinux policy in graphical installations, and so a Debian system (versions 7-9) is currently unusable with the official reference policy. We fixed these issues to make SELinux usable with Debian 9.8, GNOME environment.

We dedicated a part of this thesis to the administration and usage of a system with our SELinux policy, as problems with these are one of the main causes of low popularity of SELinux among users. We described useful official SELinux tools and provided custom scripts and tools for policy installation, configuration and customization.

We provided a more advanced user with an option to further extend our policy, for example to add a module with a different set of privileges. We created a simplified interface over the reference policy, and provided a documentation, manual and a template for creating a new module.

We tested the usability of the system that is configured with our policy, by testing functionality of common user Linux applications. We evaluated the security with the use of an industry-recognized MITRE ATT&CK Framework, and identified possibilities for future improvements of our work, based on this evaluation.

While several layers of protection must be employed to achieve an optimal security of the system, we believe our SELinux policy can significantly contribute to the security of a Linux desktop environment.

Appendix A

Release version

Our SELinux policy can be installed on a system without compilation. The compiled version can be found on the attached CD in a directory called `thesis-release`.

To install the policy, please follow the steps described in Chapter 8. The installation script can be found on the CD in a file named `thesis-release/INSTALL`, and the post-installation script in a file named `thesis-release/POSTINSTALL`.

The tool for labeling user applications is located in a file `thesis-release/set-appgroup-contexts` and is automatically installed on the system by the installation script.

Appendix B

SELinux policy source code

In this thesis, we extended the Debian version of the reference SELinux policy. We based our policy on the reference policy version 2.20161023, which was the most up-to-date at the time of writing the thesis.

The original reference policy source code can be installed on Debian as a package named `selinux-policy-src`. The source code will be installed in the following file:

```
/usr/src/selinux-policy-src.tar.gz
```

The attached CD contains the source code of our policy, and also a diff between our policy and the original reference policy. Should there be any updates to the reference policy, the reader is encouraged to apply this patch to update our policy accordingly.

The source code of our policy can be found on the CD in a directory named `thesis-src`. The `doc` subdirectory contains documentation to the modules, as well as a template for creating additional modules (`appgroup.te`, `appgroup.if`, `appgroup.fc`).

A simple syntax highlighter that we used for *Sublime-text* editor during the policy development is included on the CD in a file `SELinux.sublime-syntax`.

The patch is located on the CD in a file named `refpolicy.patch`.

Bibliography

[1] Haines, R. The SELinux Notebook, 2014. http://freecomputerbooks.com/The-SELinux-Notebook-The-Foundations.html.

[2] Mayer, F., MacMillan, K., Caplan, D. SELinux by Example: Using Security Enhanced Linux. Prentice Hall Open Source Software Development Series, 2006.

[3] Understanding Linux File Permissions, 2010. https://www.linux.com/learn/understanding-linux-file-permissions.

[4] Linux Security Module Usage, . https://www.kernel.org/doc/html/v4.16/admin-guide/LSM/index.html.

[5] An Introduction Into Linux Security Modules, . https://linux-audit.com/introduction-into-linux-security-modules/.

[6] AppArmor: Profile Components and Syntax. https://doc.opensuse.org/documentation/leap/security/html/book.security/cha.apparmor.profiles.html.

[7] AppArmor in Ubuntu. Ubuntu Wiki, . https://wiki.ubuntu.com/AppArmor.

[8] SELinux and AppArmor: An Introductory Comparison, . https://www.scribd.com/document/230617085/SELinux-and-AppArmor-An-Introductory-Comparison.

[9] TOMOYO Linux. https://wiki.archlinux.org/index.php/TOMOYO_Linux.

[10] Smack (software). Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Smack_(software).

[11] Vašut, P. Zabezpečenie pracovnej stanice s OS Linux, 2019.

[12] Smalley S., Vance, C., Salamon, W. Implementing SELinux as a Linux Security Module. National Security Agency, 2006. https:

//www.nsa.gov/Portals/70/documents/resources/everyone/digital-media-center/publications/research-papers/implementing-selinux-as-linux-security-module-report.pdf.

[13] Jahoda, M., Ančincová, B., Gkioka, I., Čapek, T. SELinux User's and Administrator's Guide. Red Hat Enterprise Linux 7, 2018. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/selinux_users_and_administrators_guide/index.

[14] Object Classes and Permissions. SElinux Project Website. https://selinuxproject.org/page/NB_ObjectClassesPermissions.

[15] Bell–LaPadula Model. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Bell%E2%80%93LaPadula_model.

[16] Biba Model. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Biba_Model.

[17] Vadinský, O. Bezpečnostní politiky SELinuxu pro vybrané aplikace prostředí KDE. Master Thesis, Vysoká škola ekonomická v Praze, 2011. https://vskp.vse.cz/26301.

[18] Tools. SELinux Project Repository. https://github.com/SELinuxProject/selinux/wiki/Tools.

[19] Walsh, D. What is SELinux trying to tell me? The 4 key causes of SELinux errors., . https://fedorapeople.org/~dwalsh/SELinux/Presentations/selinux_four_things.pdf.

[20] Flux Research Group. http://www.flux.utah.edu/index.

[21] Security-Enhanced Linux. National Security Agency. https://www.nsa.gov/What-We-Do/Research/SELinux/.

[22] SELinux userspace project. http://userspace.selinuxproject.org.

[23] SELinux integration in Linux kernel project. http://www.kernel.org.

[24] SELinux reference policy project. https://github.com/SELinuxProject/refpolicy.

[25] Jiřinec, J. SELinux politika pro BackupPC. Bachelor Thesis, Vysoká škola ekonomická v Praze, 2013. https://vskp.vse.cz/68071.

[26] SELinux. Debian Wiki, . https://wiki.debian.org/SELinux.

[27] SELinux reference policy adjusted to Debian, . https://salsa.debian.org/selinux-team/refpolicy.

[28] Debian SELinux Status and Issues, . https://wiki.debian.org/SELinux/Issues.

[29] SELinux in Ubuntu. Ubuntu Wiki, . https://wiki.ubuntu.com/SELinux.

[30] Sládek, M. Bezpečnostní politiky SELinuxu pro vybranou aplikaci prostředí Gnome. Bachelor Thesis, Vysoká škola ekonomická v Praze, 2015. https://vskp.vse.cz/45888.

[31] Grifth, D. Targeted configured semi-strict with UBAC for Fedora/Redhat distros, . http://selinux-mac.blogspot.com/2010/07/targeted-configured-semi-strict-with.html.

[32] MITRE ATT&CK, . https://attack.mitre.org/.

[33] Insights from the MITRE ATT&CK-based evaluation of Windows Defender ATP, 2018.

[34] Technique Matrix for Linux. MITRE ATT&CK, . https://attack.mitre.org/matrices/enterprise/linux/.

[35] Effects Tactic. MITRE ATT&CK, . https://attack.mitre.org/tactics/TA0034/.

[36] Internet Security Threat Report. Symantec, 2018. http://images.mktgassets.symantec.com/Web/Symantec/%7B3a70beb8-c55d-4516-98ed-1d0818a42661%7D_ISTR23_Main-FINAL-APR10.pdf?aid=elq_.

[37] What Is AppLocker?, 2017. https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/what-is-applocker.

[38] Essential Eight in Linux Environments, 2019. https://www.acsc.gov.au/publications/protect/essential-eight-linux.htm.

[39] 50 Essential Linux Applications, 2016. http://www.linuxandubuntu.com/home/50-essential-linux-applications.

[40] Tanjim. M. 24 Must Have Essential Linux Applications.

[41] Devčić, I. I. 30 Best Linux Apps And Software, 2016. https://beebom.com/best-linux-apps/.

[42] Best Ubuntu apps for a better Ubuntu experience, . https://itsfoss.com/best-ubuntu-apps/.

[43] Best Linux apps of 2018, . https://www.techradar.com/news/best-linux-apps.

[44] SELinux Project Wiki, . http://selinuxproject.org/page/Main_Page.

[45] MacMillan, K., Case, C., Brindle, J., Sellers, C. SELinux Common Intermediate Language Motivation And Design. SELinux Project GitHub repository. https://github.com/SELinuxProject/cil/wiki.

[46] Vermeulen, S. Where does CIL play in the SELinux system?, 2015. http://blog.siphos.be/2015/06/where-does-cil-play-in-the-selinux-system/.

[47] SELinux Setup. Debian Wiki, . https://wiki.debian.org/SELinux/Setup.

[48] Selinux-basics: check-selinux-installation fails without initscripts. Debian Bug report logs, . https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=860522.

[49] Where to find SELinux permission denial details. https://wiki.gentoo.org/wiki/SELinux/Tutorials/Where_to_find_SELinux_permission_denial_details.

[50] Callejas, A. A sysadmin's guide to SELinux: 42 answers to the big questions, 2018. https://opensource.com/article/18/7/sysadmin-guide-selinux.

[51] Walsh, D. SELinux with cp/mv, 2006. https://danwalsh.livejournal.com/56534.html.

[52] SLIDE. TresysTechnology. http://userspace.selinuxproject.org/archive/slide.php.

[53] Walsh, D. Using SELinux and iptables Together, . https://www.linux.com/learn/using-selinux-and-iptables-together.

[54] Introduction to SELinux. The Debian Administrator's Handbook. https://debian-handbook.info/browse/stable/sect.selinux.html.

[55] Walsh, D. Blog, . https://danwalsh.livejournal.com/.

[56] Grepl, M. Blog, . https://mgrepl.wordpress.com/.

[57] Coker, R. Blog. https://etbe.coker.com.au/tag/selinux/.

[58] Vrabec, L. Blog. https://lukas-vrabec.com/.

[59] Moore, P. Blog. http://www.paul-moore.com/blog/.

[60] Grifth, D. Blog, . http://selinux-mac.blogspot.com/.

[61] Janáček, J. General Purpose Operating System for Security Critical Applications. PhD Thesis, Comenius university in Bratislava, 2011.

[62] S. Travis. The MITRE ATT&CK Framework: What You Need to Know, 2018. https://www.tripwire.com/state-of-security/mitre-framework/mitre-attack-framework-what-know/.

[63] D. Strom. What is Mitre's ATT&CK framework? What red teams need to know, 2018. https://www.csoonline.com/article/3267691/what-is-mitres-attandck-framework-what-red-teams-need-to-know.html.

[64] Independent SELinux Policy. Fedora Project. https://fedoraproject.org/wiki/SELinux/IndependentPolicy.

[65] Grepl, M. Fun with SELinux - Writing SELinux Policy, . https://mgrepl.fedorapeople.org/PolicyCourse/writingSELinuxpolicy_MUNI.pdf.

[66] Vrabec, L., Mojzis, V. SELinux from Developer POV. LinuxDays, 2017. https://www.linuxdays.cz/2017/video/Lukas_Vrabec-SElinux.pdf.

[67] SELinux HowTo. CentOS. https://wiki.centos.org/HowTos/SELinux.

[68] Walsh, D. Awesome new coreutils with improved SELinux support. Dan Walsh's Blog, 2013. https://danwalsh.livejournal.com/67751.html.

[69] Debian Bug report logs: Bugs in source package refpolicy, . https://bugs.debian.org/cgi-bin/pkgreport.cgi?repeatmerged=no&src=refpolicy.

[70] Introduction to AppArmor. The Debian Administrator's Handbook, . https://debian-handbook.info/browse/stable/sect.apparmor.html.

[71] SELinux and AppArmor: An Introductory Comparison. https://www.scribd.com/document/230617085/SELinux-and-AppArmor-An-Introductory-Comparison.

[72] Linux Kernel Security (SELinux vs AppArmor vs Grsecurity), 2009. https://www.cyberciti.biz/tips/selinux-vs-apparmor-vs-grsecurity.html.