SYSTEMATIC POLICY ANALYSIS AND MANAGEMENT

by

Wenjuan Xu

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Information Technology

Charlotte

2010

Approved by:

____
Dr. Gail-Joon Ahn

____
Dr. Mohamed Shehab

____
Dr. Cem Saydam

____
Dr. Kimberly A. Warren

ABSTRACT

WENJUAN XU. Systematic policy analysis and management.
(Under the direction of DR. GAIL-JOON AHN AND DR. MOHAMED SHEHAB)

Determining whether a given policy meets a site's high-level security goals has been a challenging task, due to the low-level nature and complexity of the policy language, various security requirements and the multiple policy violation patterns. In this dissertation, we outline a systematic policy analysis and management approach that enables system administrators to easily identify and resolve various policy violations. Our approach incorporates a domain-based isolation model to address the security requirements and visualization mechanisms to provide the policy administrator with intuitive cognitive sense about the policy analysis and policy violations. Based on the domain-based isolation model and the policy visualization mechanisms, we develop a visualization-based policy analysis and management framework. We also describe our implementation of a visualization-based policy analysis and management tool that provides the functionalities discussed in our framework. In addition, a user study is performed and the result is included as part of our evaluation efforts for the prototype system.

One important application of our policy analysis and management is to support remote attestation. Remote attestation is an important mechanism to provide the trustworthiness proof of a computing system by verifying its integrity. In our work, we propose a remote attestation framework, called Dynamic Remote Attestation Framework and Tactics (DR@FT), for efficiently attesting a target system based on our extended visualization-based policy analysis and management approach. In addition, we adopt the proposed visualization-based policy violation expression to represent integrity violations with a ranked violation graph, which supports intuitive reasoning of attestation results. We also describe our experiments and performance evaluation.

ACKNOWLEDGEMENTS

First and foremost, I owe my deepest gratitude to my advisor Professor Gail-Joon Ahn. He is always there to listen and to give advice. His great personality and dedication to the research have taught me a lot in the past five years. His expert guidance and support have made this work possible, sparked my interest in access control, and were the foundation of a great graduate research experience. I also give great gratitude to Professor Dr.Mohamed Shehab who as my co-advisor guided my work of policy visualization and gave me numerous support in research.

Many thanks to the numerous individuals who worked with me on collaborative papers, and on other topics related to my research over the last few years, including Dr. Xinwen Zhang from Samsung Information Systems America, Professor Irini Kokkinou from Indiana University, Professor Jean-Pierre Seifert from Deutsche Telekom Lab and Technical University of Berlin and graduate students Hongxin Hu, Jing Jin and Moo Nam Ko. Their feedback and criticism enriched my work and helped me to align it with other research projects.

I also wish to express my deep appreciation to my PhD committee members, Professor Gail-Joon Ahn, Professor Mohamed Shehab, Professor Cem Saydam and Professor Kimberly A. Warren. The assistance and guidance they provided in the preparation of this manuscript have been invaluable.

Furthermore, I want to acknowledge the multi-year funding I received for this work via research grants from National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565) to Professor Gail-Joon Ahn.

Finally, I would like to thank my husband Willson Kwok. His support, encouragement, patience and unconditional love enabled me to surpass hardships and complete this work. I also thank my parents FengMei Liu and Ren Xu, my sisters Wenhong Xu and Wenping Xu. They were always supporting me and encouraging me with their best wishes. Finally, I would thank my son Ethan Kwok. He gives me the motivation and courage to get through this work.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Determining whether a system can be trusted or not is a critical problem in system and network management. Particularly for security reason, a local or remote system administrator typically needs to verify if a system meets information security objectives, such as integrity, confidentiality, and availability requirements. For example, to deploy applications in distributed and collaborative computing environments, one machine may want to know if another machine is running a known-good version of an application software on a well-configured, trusted operating system. Without this guarantee, a remote machine may be running buggy or malicious application code, or may be improperly configured such that the trusted application can be corrupted by untrustful programs or users. For these purposes, formal and general security analysis is needed to trust a running system based on its current security configurations, particularly, its security policies, which outline the rules, laws and practices for the network access [5]. This dissertation addresses the issues of security policy analysis and management.

1.1 Security Policy Analysis and Management Challenging

Information flow control is the foundation of many security requirements such as integrity and confidentiality. The earliest information flow analysis work adopts a lattice model to illustrate flow relationship between objects [26]. Those works assume that every object is labeled with a security attribute, and checks information flow by examining the security labels of the objects in a flow path. Especially, the lattice model requires that information cannot flow from low integrity objects to high integrity objects. In practical systems, under various circumstances, information flow is allowed from low integrity subjects to high integrity subjects. Clark-Wilson model [25] attempts to capture this notion.

It states that through certain programs so-called transaction procedures (TP), information can flow from low integrity objects to high integrity objects. Jaeger et al. [36] adopt this notion and propose a CW-Lite model, where filters are deployed in all application interfaces handling information flow from low integrity data to high integrity applications.

There are several related approaches and tools to analyze policies based on certain information flow control [10, 36]. Most of these work focus on the identification of a common and minimum system trusted computing base (TCB) [13, 12] which includes trusted processes for an entire system. These approaches can analyze whether a policy meets security requirements such as no information flow from low integrity processes (e.g., unprivileged user processes) to system TCB (e.g., kernel and init). However, they cannot identify integrity violations for high level services and applications that do not belong to system TCB. In practical, other than system TCB protection, a high level application or system service is required to achieve integrity assurance through controlling any information flow from other applications. TCB community [65] clearly states such a critical situation as follows: "a network server process under a UNIX-like operating system might fall victim to a security breach and compromise an important part of the system's security, yet is not part of the operating system's TCB." Therefore, a more comprehensive policy analysis for TCB identification and policy violations is desired.

Another issue in policy analysis is the large size and high complexity of typical policies in contemporary systems. For example, an SELinux [42] policy has over 30,000 policy statements in a desktop environment. Under these situations, several challenges exist for system designers or administrators in the context of policy analysis. We enumerate such challenges: (1) *Understanding and querying a policy*: All previous policy query tools lack an effective mechanism for a user to understand a particular policy. Furthermore, without understanding a policy, the user even cannot figure out what to query; (2) *Recognizing information flow paths*: In the work of information flow based policy analysis [31, 55], information flow paths are expressed with text-based expressions. However, primitive text-based explanations cannot provide appropriate degree of clarity and visibility for describing flow paths; and (3) *Identifying integrity violation patterns, inducements, and aftermaths*: In the work of Jaeger et al. [36, 56], they elaborate violation patterns using graphs. However, their approach does not leverage the features and properties of graphs for analyzing policies and identifying the causes and effects of integrity violations.

In summary, we need to address several challenging questions for managing security policy:

- How to express the security requirements comprehensively and accordingly?
- How to analyze the security policy and present the analysis result understandably?
- How to build a usable security policy analysis and management framework and how to evaluate its usability?

These are the critical questions to be answered to systematically analyze and manage the security policy. Even though some approaches have been proposed from various aspects to address the policy analysis and management issues, our study clearly indicates that there is a need to design a systematic policy analysis and management approach which is general and flexible enough to reflect and cope with the various access control requirements as well as solving the issues such as policy complexity. In this research, we would attempt to make one step towards this direction.

1.2 Statement of the Hypothesis

Therefore, this research hypothesizes that:

A systematic policy analysis and management approach is critical for achieving system assurance.

We first analyze and articulate the policy analysis requirements for a system in an open environment. These requirements are then reflected and addressed in our proposed policy analysis and management approach in terms of a domain-based isolation model that is validated through a simulation tool, supportive method for visualizing and analyzing the security policies against this model, and essential system components and architecture with detailed policy visualization and enforcement mechanisms. In order to evaluate the feasibility and usability of our approach, a proof-of-concept system is implemented for supporting systematic policy analysis and management. Also, we analyze the usability of this method and tool through the usability study. Furthermore, we extend the policy analysis approach to realize efficient and effective remote attestation in a dynamic environment.

1.3 Summary of Contributions and Dissertation Organization

The contributions of our policy analysis and management are summarized as follows:

- We articulate the access control requirements with a proposed domain-based isolation model and prove it with a simulation tool CPN.
- We propose a visualization-based policy analysis method with graphical query-based mechanisms to support the security policy analysis.
- Based on the proposed security model and visualization method, we develop a visualization based policy analysis framework.
- We evaluate our proposed framework through performing the usability study on our proof-of-concept prototype.
- The visualization-based policy analysis and management work is extended to support the remote attestation in a dynamic environment.

The remainder of this dissertation is organized as follows. Chapter 2 reviews the issues in managing complex security policy and discusses existing policy analysis approaches and information visualization methods. In Chapter 3, we analyze the security requirements of the security policy and define an information flow model to reflect these requirements.

Chapter 4 elaborates our method of policy visualization. Chapter 5 proposes the visualization-based policy analysis and management framework, introduces our prototype implementation based on the proposed framework and further proves the usability of our framework by performing a user study. An extension of policy analysis approach to remote attestation is explained in Chapter 6. Finally, Chapter 7 summarizes this dissertation and presents some directions for future work.

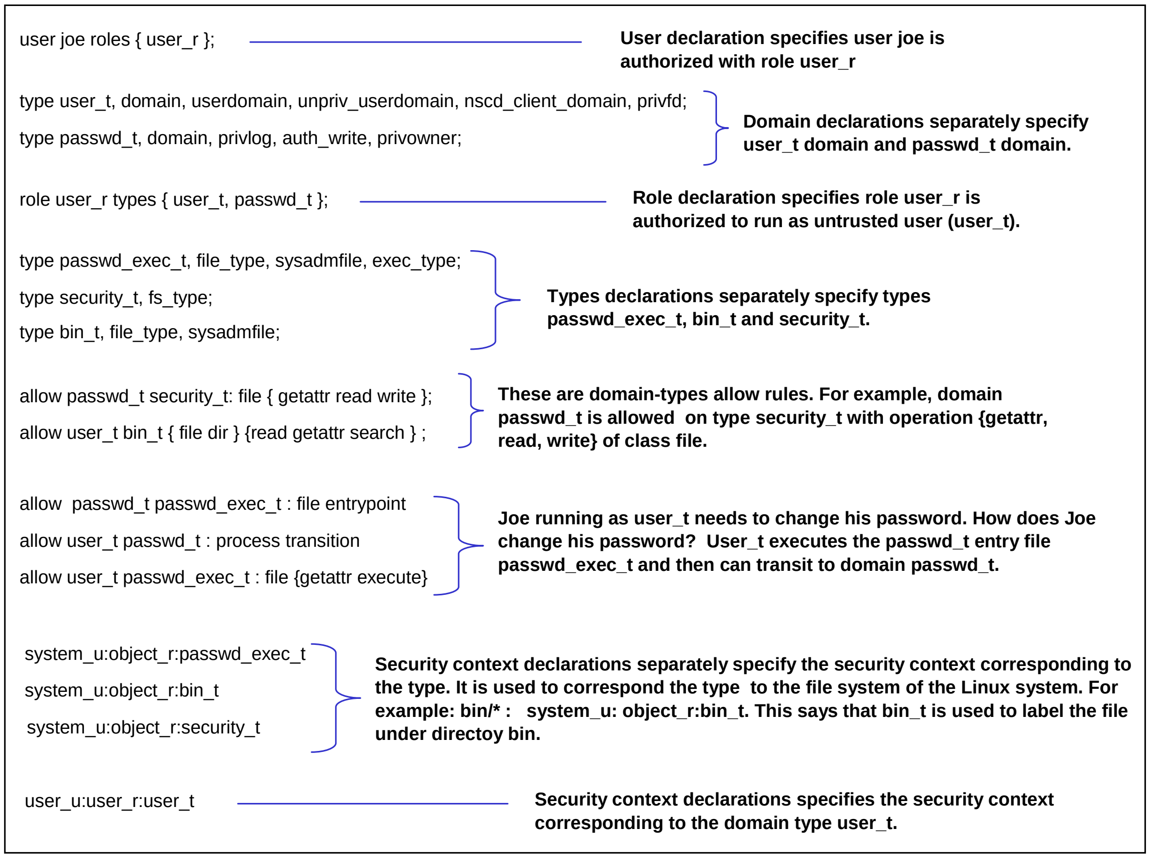
CHAPTER 2: BACKGROUND AND RELATED WORK

2.1 SELinux Overview

In this section, we briefly overview the SELinux policy [59], which is a typical example of complex security policy.

2.1.1 SELinux Policy Introduction

Security-Enhanced Linux (SELinux) [59] enforces mandatory access control (MAC)-based policies to enhance the security of Linux systems, where the MAC mechanism is implemented through the Type-Enforcement (TE) policy model [21, 42]. In TE, *domain types* are used to label processes, while object types are used to label files and other resources. In SELinux, there is no essential difference between domain types and object types, and both are called types in general. SELinux associates a security context with each subject or object (process, files, etc.). A security context is a tuple that identifies a user, a role, and a type. A set of *policy rules* specify how subjects can access objects based on relationships between their security contexts. For example, the national security agency (NSA) SELinux example policy [3] defines a type `security_t` and assigns to all files in directory `/security`. The policy also defines domain types `user_t` and `passwd_t`, which are respectively associated with processes with a specified set of executables for unprivileged users and processes for password management. There is a policy rule allowing `user_t` to call (transit) the `passwd_t` and another rule allowing `passwd_t` processes to directly operate on resources with type `security_t`. Hence, a `user_t` subject can operate on a `security_t` object. An operation in SELinux is identified by two pieces of information: a class (e.g., file, directory, process, and socket) and a permission (e.g., read, unlink, signal, and sendto). Currently SELinux defines 28 classes and approximately 120 permissions.



```
user joe roles { user_r };
```

User declaration specifies user joe is authorized with role user_r

```
type user_t, domain, userdomain, unpriv_userdomain, nscd_client_domain, privfd;
type passwd_t, domain, privlog, auth_write, privowner;
```

Domain declarations separately specify user_t domain and passwd_t domain.

```
role user_r types { user_t, passwd_t };
```

Role declaration specifies role user_r is authorized to run as untrusted user (user_t).

```
type passwd_exec_t, file_type, sysadmfile, exec_type;
type security_t, fs_type;
type bin_t, file_type, sysadmfile;
```

Types declarations separately specify types passwd_exec_t, bin_t and security_t.

```
allow passwd_t security_t: file { getattr read write };
allow user_t bin_t { file dir } {read getattr search } ;
```

These are domain-types allow rules. For example, domain passwd_t is allowed on type security_t with operation {getattr, read, write} of class file.

```
allow  passwd_t passwd_exec_t : file entrypoint
allow user_t passwd_t : process transition
allow user_t passwd_exec_t : file {getattr execute}
```

Joe running as user_t needs to change his password. How does Joe change his password? User_t executes the passwd_t entry file passwd_exec_t and then can transit to domain passwd_t.

```
system_u:object_r:passwd_exec_t
system_u:object_r:bin_t
system_u:object_r:security_t
```

Security context declarations separately specify the security context corresponding to the type. It is used to correspond the type to the file system of the Linux system. For example: bin/* : system_u: object_r:bin_t. This says that bin_t is used to label the file under directoy bin.

```
user_u:user_r:user_t
```

Security context declarations specifies the security context corresponding to the domain type user_t.

Figure 2.1: SELinux policy example.

Other than TE model, SELinux also uses role-based access control (RBAC) model [60] to help organize policy rules. A user is assigned with a role which is an abstraction designed to make policy rules more concise. For example, if many users require the same set of permissions, a role can be defined, a policy rule can be introduced to state that those users can enter this role, and another rule is specified to associate permissions with the role. The set of permissions associated with a role is expressed with types. For all object types, SELinux uses a role `object_r` and a user `system_u` to specify their security contexts. A domain type can be associated with different roles and users for different security contexts. Figure 2.1 shows some policy structures with SELinux policies. A user *Joe* can only login as role `user_r`, which is associated with types `user_t` and `passwd_t`. As `passwd_t` has permissions of `security_t: file {getattr read write}`, Joe can read and modify the password file in the system, which is labeled with `security_t`. Related policy rules for type and security context declarations are also included in Figure 2.1.

2.1.2 SELinux Policy Characteristics

SELinux types are classified into different categorizations corresponding to the functions performed by processes and the operations performed on the different objects [59]. We first present the policy classification which is used in the subsequent sections. The domain and type classifications are defined as follows:

- **Domain Classification:** According to SELinux policy configurations from NSA [59], domain types in SELinux can be classified into *system domains*, *user program domains*, and *user login domains*. *System domains* are composed of domains labeled as system processes (e.g., `kernel_t`, `initrc_t`, and `init_t`) or daemons (e.g., `sendmail_t` and `ftpd_t`). *User program domains* include unprivileged user program domains (e.g., `user_xserver_t`), administrator program domains (e.g., `sysadm`
`_xserver_t`), and some other program domains (e.g., `logrotate_t` and `passwd_t`). *User login domains* are the domains used for user authorization such as `user_t`, `sysadm_t`, and `staff_t`. Due to the large number of vulnerabilities that have been found in daemons (e.g.,`sendmail_t`) we divide system domains into daemons and general system domains.

- **Type Classification:** Types in SELinux can be classified into *security types* (e.g., `security_t`), *device types* (e.g., `fixed_disk_device_t` and `device_t`), *file types* (e.g., `etc_t`), *procfs types* (e.g., `sysctl_kernel_t` and `proc_t`), *devpts types* (e.g., `ptmx_t`), *nfs types* (e.g., `nfs_t`), and *network types* (e.g., `icmp_socket_t` and `port_t`).

The details of domain and type classifications are listed in Table 2.1.

Table 2.1: SELinux characteristics

SELinux Domains Classification		
Domain Classifications	Subjects	Examples
System Domains (SD)	domains defined for system services	kernel_t, initrc_t
Daemons (DAE)	domains for system daemons	klog_t, sendmail_t
Program Domains (PRO)	domains for user programs	user_xserver_t, passwd_t, sysadm_xserver_t
User Login Domains (ULO)	domains for authorization of different users	user_t, staff_t, sysadm_t

SELinux Types Classification		
Types	Objects	Examples
security types (ST)	policy configuration related files.	security_t
device types (DT)	files under /device	fixed_disk_device_t
file types (FT)	files under directory /root, /etc, etc.	etc_t, root_t
procfs types (PT)	pseudo files under /proc.	proc_t
devpts types (DE)	pseudo files under /dev/pts	ptmx_t
NFS types (NF)	files from an NFS server	nfs_t
Network types (NE)	files for network objects	port_t

2.1.3 SELinux Policy Types

SELinux policy can be also classified into targeted policy and strict policy [44], where the targeted policy is derived from the strict policy with the similar structure. However, the strict policy attempts to specify policy rules for most programs, while the targeted policy focuses on network and system services. The primary difference between strict and targeted policies is the use of the unconfined domain types (`unconfined_t`) and removal of any other user domain type (for example, `sysadm_t, user_t`).

From the organization of the security policy, SELinux policy can be further classified into original policy and reference policy [44], where the traditional policy is a combination of sets of rules specifying the security policy. On the other hand, reference policy is targeted to re-engineer the existing policies by organizing most program policies into *.te* file, *.if* file or *.fc* file, which brings the flexibility for policy design.

2.1.4 SELinux Policy Security Goals

Six critical security goals are outlined in SELinux policies [16]: (G1) limiting raw access to data, (G2) protecting kernel integrity, (G3) protecting system file integrity, (G4) confining privileged processes, (G5) separating processes, and (G6) protecting the administrator domain. Among these goals, G2, G3 and G6 are mainly about integrity protection such as boot files, proc files and security policy related objects. As a more complex goal, G1 is to protect both the integrity and confidentiality of system resources. For example, a write operation to fixed disk devices is restricted to `fsck` for checking file system consistency. G4 and G5 deal with least privilege properties through restricting accesses to certain domain types. For instance, a mail server program can only access certain resources such as mail spool file, and a user program is prohibited from interfering with administrator programs.

2.2 Information Flow Related Model

Biba integrity model [20, 26] is a widely cited information flow based integrity model. In Biba model, integrity is preserved for a subject if all high integrity objects meets integrity requirements and all information flows from subjects with equal or higher integrity. Hence, Biba property is fulfilled if a high integrity process cannot read lower integrity data, execute lower integrity programs, nor obtain lower integrity data in any other manner.

Similar to Biba, Low-Water Mark Integrity (LOMAC) [62] allows information flow from high integrity to low integrity without restrictions. However, information is unescapable from low integrity to high integrity in certain cases. For example, in a UNIX system, when a root user logs in, the root user may command the subject executing *emacs()* to observe low-integrity objects. In this case, the proper privilege set for the eamcs subject is to prohibit modification of high-integrity objects. On the other hand, the root user may command the emacs subject to observe and modify only high-integrity objects. In this case, the privilege set which prohibits modification of high-integrity objects would be inappropriate. For solving such ambiguous, LOMAC supports high-integrity process to read low integrity data, while downgrading the process's integrity level to the lowest level that it has ever read. For example, when the root user observes low-integrity objects, this process needs to be downgraded.

Different from Biba model, Clark-Wilson [25, 54] model provides a different view of dependencies, which states that through certain programs so-called transaction procedures (TP), information can flow from low integrity objects to high integrity objects. Based on Clark-Wilson model, the concept of TP is evolved into filters in CW-Lite [56] model. A filter can be a Firewall, an authentication process, or a program interface for downgrading or upgrading the privileges of a process. In CW-Lite model, information can flow from low integrity processes to high integrity processes through filters.

Usable Mandatory Integrity Protection (UMIP) [41] is an integrity model which states that low integrity processes are not allowed to flow to high integrity processes with several exceptions. For example, information flow from low integrity process (i.e., network) to high integrity process (administration) through remote administration such as *sshd* is allowed in the model. Furthermore, low integrity process (unprivileged user) can flow to high integrity process (administration) through a process such as *su*.

Trusted Computing Base (TCB) protection is proposed [27], where TCB is the total combination of protection mechanisms within a computer system, which includes the hardware, software and firmware that are trusted to enforce a security policy. They propose to separate TCB from the remainder of the system and a trust path must also exist so that a user can access TCB without being compromised by other processes or users.

Domain isolation [24] is another way to specify the security requirements in the system. The main purpose of domain isolation is to state the web security requirements, for considering appropriate enforcement of the same-origin principle. This same-origin principle can be interpreted as "a script originated from one Internet domain should not be able to read, manipulate or infer the contents originated from another domain." Failures to enforce this principle may result in severe security consequences such as information leakage or identity theft.

2.3 Information Flow-based Policy Analysis Tool

The closest existing work to ours include Jaeger et al. [36] and Shankar et al. [56]. In these work, they use a tool called Gokyo for checking the integrity of a proposed TCB for

SELinux [59]. Also, they propose to implement their idea in an automatic way. Gokyo mainly identifies a common TCB in SELinux but a typical system may have multiple applications and services with various trust relationships. However, achieving the integrity assurance for these applications and services has not been addressed in Gokyo.

SETools [9] are a set of tools developed by Tresys Technology to provide support for SELinux policy. Among them, *seaudit* is used to analyze audit messages from SELinux; *seaudit-report* generates highly-customized audit log reports; *sechecker-command* line tool performs modular checks on an SELinux policy; *sediff* identifies the policy difference in SELinux policy; and *secmds* is a command-line tool to analyze and search SELinux policy. APOL is a graphical user interface based tool to analyze a SELinux policy file [6]. A lot of functions are supported in APOL such as browsing and searching policy components (e.g., types, attributes, object classes, roles, users, and booleans), searching through type enforcement and other rules, and viewing file contexts from a filesystem. In addition, APOL allows policy administrator to perform domain transition, file relabel, types relationship, and information flow analysis.

SLAT (Security Enhanced Linux Analysis Tool) [31] defines an information flow model and policies are analyzed based on this model. Especially, SLAT has a policy query mechanism which are written in a special-purpose language. A SLAT query is, roughly speaking, a kind of regular expression that specifies the expected form of information flow paths between two specific security contexts. SLAT determines whether all information flow paths between those endpoints meet the expected form. If the answer is “no”, SLAT provides a counterexample. SLAT queries can be converted into a finite automata that can easily be expressed as logic programs. However, SLAT does not support policy editing and only provides text-based interface which is hard for the policy administrator to understand.

PAL (Policy Analysis using Logic Programming) [55] uses SLAT information flow model to implement a framework for analyzing SELinux policies. The main difference between PAL and SLAT is their query language. PAL is realized with logic programs, and is more flexible than SLAT. For example, PAL can answer queries whose results are sets of security contexts, relations between security contexts, etc. Also, PAL can answer high-level queries such as "find all security contexts from which information can flow to security context c without passing through security context d." Although PAL is more flexible than SLAT with respect to the policy query, it also has the problem such as complexity in editing policy and lack of user-friendliness.

SELinux Policy Editor (Seedit) [7] is a tool originally developed by Hitachi Software. The main purpose of Seedit is to over-come the above mentioned problems. In Seedit, it provides a simplified policy specification called Simplified Policy Description Language (SPDL) for the policy administrator to manage the security policy. Also, it has a graphical interface to generate simplified policies so administrators do not have to remember the syntax of SPDL. However, Seedit also has two challenges to solve: (1) Integrating object classes and access vector does not allow to support a fine-grained configuration specification. (2) Seedit compiler compiles the simplified policy into SElinux policy configuration file. However, the compiler needs to be updated whenever new version of SELinux is released.

In summary, all the existing tools attempt to provide a query-based policy analysis for policy developers or security administrators. However, they all display policies and policy query results in text-based expressions, which are difficult to understand and accommodate the analysis results.

2.4 Visualization Technology Application in Security

Information visualization [34] enables users to explore, analyze, reason and explain abstract information by taking advantage of its visual cognitive effects. Several disciplines have adopted information visualization mechanisms to better understand and reason about the collected data.

There are several work to leverage information visualization for network security. Lakkaraju et al. [69] presents a visualization design to enhance the ability of an administrator to detect and investigate anomalous traffic between a local network and external domains. NCSA [70] shows the network data information such as IP addresses is visualized to present better view for system administrators. Event Correlation for Cyber-Attack Recognition System [43] performs aggregation and correlation of the security events based on their semantic content to visualize attack tracks in a real-time manner.

Several intrusion detection work also focus on visualization of intrusions. In addition, Robert et al. [28] proposes a methodology for analyzing network and computer log information with visual analytics of user behaviors. Thompson et al. [63] attempts to determine whether a graphical interface or a standard textual interface is more affective for intrusion detection.

Also, an interactive visualization framework [68] is designed for the automated trust negotiation protocol. This framework provides capabilities to perform the interactive visualization of a trust negotiation session, display credentials and policies, analyze the relations of negotiated components, and refine access control policies and negotiation strategies.

Some efforts in firewall policy visualization are also performed [40, 46, 64]. VisualFirewall [40] seeks to aid in the configuration of firewalls and monitoring of networks by providing four simultaneous views that display packet details and corresponding firewall reactions. Fireviz [46] incorporates a peripheral mapping of the network on a user's screen and displays network events along this periphery. Policyvis [64] visualizes firewall policies to enable a user to place general inquiry such as "does my policy do what I intend to do unrestrictedly? "

2.5 Visualization Representable Technology

The typical visualization technology in this dissertation can be classified into two main streams such as node-link and adjacency matrix methods.

There are a lot of different node-link based visualization method. Semantic Substrates [18] is a visualization method based on node-link targeting to visualize the social network based on semantic substrates. The basic idea is that the graph layouts are based on user-defined semantic substrates, that are non-overlapping regions in which node placement is based on node attributes. Also, users interactively control link visibility to limit clutter and thus ensure comprehensibility of source and destination. Semantic substrates are effective only if there are some categorical attributes or if a numerical attribute can be combined to form categories. A small number of categories is convenient for effectively visualizing the results. Although there are limitations in the implementation, the utility of semantic substrates cope with a large number of nodes and links. Also, with the node-link based diagram, semantic substrates method is very useful to support a small number of graphs. However, in many situations, the graph maybe very large and dense.

Adjacency matrix [39] is a square matrix-based information visualization method. In the adjacency matrix, a non-zero entry means that there is a link from the vertex represented by a column of the matrix to the vertex represented by a row. On the other hand, the non-zero entries in the adjacency matrix are usually visualized as colored blocks and zero entries as blank blocks. Adjacency matrix is widely used in graph visualization because they can effectively display a large and complex graph by interpreting the structural information in a matrix view to a graph. Although adjacency matrix can be used to visualize both directed and undirected graphs, it may cause some difficulties in finding the path from one node to another node in the directed graph.

CHAPTER 3: DOMAIN-BASED ISOLATION MODEL

To comprehensively and effectively analyze the security requirements focusing on access control, in this chapter, we first introduce a domain-based isolation model. Then, we validate through a case study with SELinux security policy for identifying the policy violations.

3.1 Domain-based Isolation Model Definition and Specification

3.1.1 Security Policies

A security *policy* is composed of a set of subjects, a set of objects, and a set of *policy statements or rules* which states that a subject can perform what kind of actions on an object. For information flow purpose, all operations between subjects and objects can be classified as *write_like* or *read_like* [31] and operations between subjects can be expressed as *calls*.

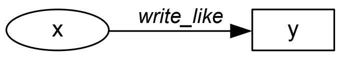


Figure 3.1: Write

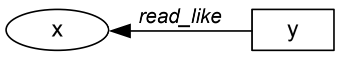


Figure 3.2: Read

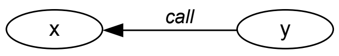


Figure 3.3: Call

Depending on the types of operations, information flow relationships can be identified. If subject *x* can write to object *y*, then there is information flow from *x* to *y* (shown in

Figure 3.1), which is denoted as $write(x,y)$. On the other hand, if subject x can read object y, then there is information flow from y to x denoted as $read(y,x)$ (shown in Figure 3.2). Another situation is that if subject x can call another subject y, then there is information flow from y to x, which is denoted as $call(y,x)$ (shown in Figure 3.3).

Moreover, the information flow relationships between subjects and objects can be further described through *flow transitions*. In a policy, if a subject s_1 can write to an object o which can be read by another subject s_2, then it implies that there is an *information flow transition* from a subject s_1 to a subject s_2, denoted as $flowtrans(s_1,s_2)$. Also, if a subject s_2 can call a subject s_1, there is a flow transition from s_1 to s_2. A sequence of flow transitions between two subjects represents an *information flow path*.

3.1.2 Domain-based Isolation Model (DIM)

Retrospecting the integrity models introduced in related work, one-way information flow with Biba would not be sufficient for many cases as communication and collaboration between application or service domains are frequently required in most systems. Although filters between high and low integrity data are sufficient enough for TCB and NON-TCB isolations, it is not suitable for application or service domain isolations. For example, processes of user applications and staff applications are required to be isolated since both are beyond the minimum and common TCB boundaries. With that reason, we develop a *domain-based isolation model*, in which a concept called *domain TCB* is defined to describe subjects and objects required to be isolated for an information domain. To be clear, the minimum and common TCB of a system is called *system TCB* in our work. Also, for a subject in a system, if it neither belongs to the system TCB, nor belongs to the domain TCB of a particular application or service, then it is in the NON-TCB of the system.

System TCB System TCB is a concept same as the concept of TCB [12]. To identify system TCB, reference monitor-based approach is proposed in [17], where system TCB is identified through the identification of subjects functioning as reference monitor. Applying this idea to SELinux, subjects functioning as reference monitor such as *checkpolicy* and *loading policy* belong to system TCB. Also, subjects used to support reference monitor such as *kernel* and *initial*, should be included into system TCB. After reference monitor-based identification of initial system TCB is completed, other subjects such as *lvm*, *restorecon* should also be added into system TCB based on their relationships with initial system TCB. Other optional methods for identifying system TCB are proposed in [36].

Information Domain As mentioned above, an application or service information domain consists of a set of subjects and objects. Here, we propose two steps to identify an information domain.

- *Step1: Keyword-based domain identification* Generally, subjects and objects in a security policy are described based on their functions, e.g., *http* is always the prefix for describing web server subjects and objects in SELinux. Hence, to identify the web server domain, we use keyword *http* to identify the initial set of subjects and objects.

- *Step2: Flow-based domain identification* In a security policy, some subjects or objects cannot be identified through keyword prefix. However, they can flow to initially identified domain subjects and objects, influencing the integrity of this domain. Therefore, we also need to include these subjects and objects into the domain. For instance, in a Linux system, *var* files can be read by web server subjects such as *httpd*. Hence they should be included in the web server domain.

Domain TCB To protect the integrity of an information domain, a domain TCB is defined. TCB(d) (domain *d*'s TCB) is composed of a set of subjects and objects in domain *d* which has the same level of security sensitivity. In other words, a web server domain running in a system consists of many subjects–such as processes, plugins, and tools, and other objects including data files, configuration files, and logs. We consider all of these subjects and objects as TCB of this domain, while its network object such as $tcp : 80$ (`http_port_t`) is not considered as TCB since it may accept low integrity data from low integrity subjects. In a system, the integrity of an object is determined by the integrity of subjects that have operations on this object. Hence, we need to identify TCB(d) subjects of each information domain and verify the assurance of their integrity.

To ease this task, a minimum TCB(d) is preferred. However, in the situation that the minimum TCB(d) subjects have dependency relationships with other subjects, these subjects should be added to domain TCB or dependencies should be removed. Based on these principles, we first identify initial TCB(d) subjects which are predominant subjects for domain *d*. We further discover TCB(d) considering subject dependency relationships with the initial TCB(d) through *flow transition-based identification* and *violation-based adjustment*.

- *Step1: Initial TCB(d) identification* In an information domain, there always exist one or several predominant subjects, which launch all or most of other subjects functioning in the same domain. Here, we identify the initial TCB(d) subjects based on such subject launching relationships and the number of subjects that those subjects can cascadingly launch. For example, for web server domain, *httpd* launches all other processes such as *httpd_script*, hence it belongs to the initial TCB(d) of this domain.

- *Step2: Flow transition-based TCB(d) identification* The subjects that can flow only to and from the initial TCB(d) are included into domain TCB. For instance, if subject *httpd_php* can flow only to and from *httpd*, then *httpd_php* should be included into TCB(d).

- *Step3: TCB(d) adjustment by resolving policy violations* After identifying policy violations, we adjust the identified TCB(d) to exclude or include subjects accordingly. For example, a subject *awstats_script* (web server statistics script) is initially excluded from TCB(d). After identifying policy violations caused to web server TCB(d) by *awstats_script*, we can figure out that these violations can be ignored. Hence, the TCB(d) should be adjusted to include *awstats_script*.

DIM Definition To protect the identified TCB_s and TCB_d, we develop domain-based isolation model, whose principles are similar to those in Clark-Wilson [54]. Clark-Wilson leverages transaction procedures (TP) to allow information flow from low integrity to high integrity processes. We also adopt the concept of filter. With the identifications of TCB_s, TCB_d and filters, for information domain `d`, all other subjects in a system are categorized as NON-TCB.

Based on the concept of system TCB and TCB(d), a domain-based isolation model is defined as follows.

Definition 1 *:* Domain-based isolation model is satisfied for an information domain *d* if for any information flow to TCB(d), the information flow path is within TCB(d); or the information flow path is from the system TCB to TCB(d); or the information flow path is from another domain TCB and it is filtered.

Through this definition, domain-based isolation model achieves the integrity of an information domain by isolating information flow to TCB(d). This model requires that any information flow happening in a domain *d* adheres within the TCB(d), from system TCB to the TCB(d), or from another domain TCB via filter(s). In this work we do not discuss the integrity of filters, which can be ensured with other mechanisms such as formal verification or integrity measurement and attestation [52]. Filters can be processes or interfaces that normally is a distinct input information channel and is created by a particular open(), accept() or other call that enables data input. For example, Linux *su* process allows a low integrity process (e.g., staff) to be high integrity process (e.g., root) through calling *passwd* process thus `passwd` can be regarded as a filter for processes run by root privilege. For another example, high integrity process (e.g., httpd administration) can accept low integrity information (e.g, network data) through the secure channel such as *sshd*. Consequently, `sshd` can be treated as another example of filter for higher privilege processes. Normally, it is the developer's tasks to build filtering interfaces and prove effectiveness to the community [49]. Generally, without viewing system application codes, an easier way for policy administrator to identify filters is to declare filters with clear annotations during policy development [56]. Here, we assume that filters can be identified through annotations. Also, in our work, initially we do not have a set of predefined filters. After detecting a possible policy violation, we identify or introduce a subject as a filter to resolve policy violations.

Policy Violation Detection Based on the domain-based isolation model, we treat a TCB(d) as an isolated information domain. We propose the following rules for articulating possible policy violations for system TCB and TCB(d) protections.

Rule 1 *:* If there is information flow to a system TCB from its subjects without passing any filter, there is a policy violation for protecting the system TCB.

Rule 2 *:* If there is information flow from TCB(d_x) to TCB(d_y) without passing any filter, there is a policy violation in protecting TCB(d_y).

3.1.3 Policy Violation Resolution

After possible policy violations are identified with violation detection rules, we take systematic strategies to resolve them. Basically, for a violation, we first evaluate if it can be resolved by adding or removing related subjects to/from system or domain TCBs. This causes no change to the policy. Secondly, we try to identify if there is a filter along with the information flow path that causes the violation. If a filter can be identified, then the violation is a false alarm and there is no change to the policy graph. Thirdly, we attempt to modify policy, either by excluding subjects or objects from the violated information flow path, or by replacing subjects or objects with more restricted privileges. In addition, we can also introduce a filter subject that acts as a gateway between unauthorized subjects and protected subjects.

3.2 Analyze Security Policies with CPN Against DIM

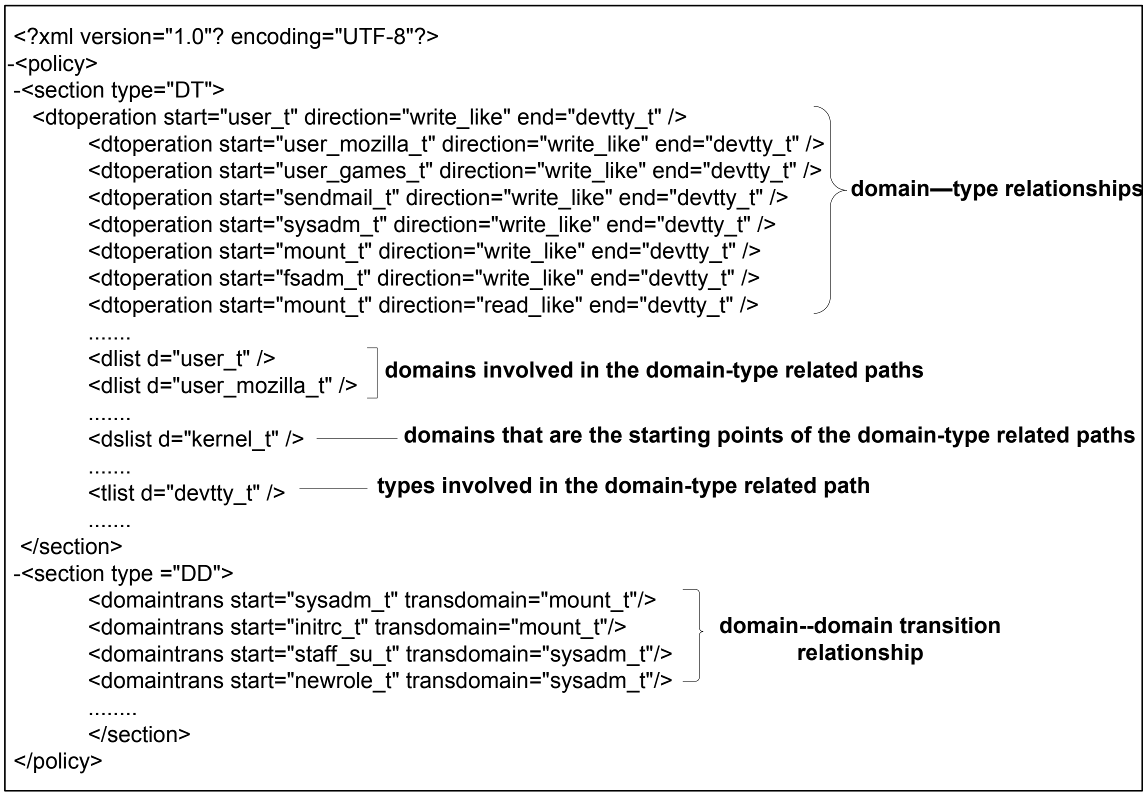
Colored Petri Nets (CPN) is a powerful graph-based analysis tool for system modeling [37]. A Petri Nets includes three basic components: places, transitions, and arcs. Arc expressions specify a collection of tokens, which are added to or removed from places. If a transition input contains at least one token that is equal to the corresponding arc expression, the transition is enabled. The difference between CPN and Petri Nets is the inclusion of color sets in CPN, which can be viewed as abstract data types in a programming language. These types determine data attributes and operations used in arcs, guards, and initialization expression functions. CPN can support graph hierarchy, zoom in, zoom out, color expression, and so on. Also, CPN has a simulator to support execution of specified models. The simulation is to validate whether the system works correctly reflecting the design principles. It supports both interactive and automatic simulations. In the interactive simulation, a user can set breakpoints, choose between enabled binding elements, change markings of places, and study the tokens in detail. However, in the automatic simulation, the simulator makes random choices of the enabled binding elements and automatically executes the whole CPN models.

3.2.1 Policy Simulation with CPN

3.2.1.1 Policy in XML

Based on the definition of policy graph, we design a scheme to express policies in eXtensible Markup Language (XML) so that our policy can be specified in a standard form. As shown in Figure 3.4, the XML file generated from a parsed SELinux policy has two elements: *DT* and *DD*. The *DT* element includes three sets of subelements: (1) non replicated operation relationships between types, (2) the types that can flow to and from other types, and (3) the starting domain types which can not be flown into by any other types. Also, it may contain the types that can only be flown into. The *DD* element specifies information about non-replicated transition relationships between types.

In our experiments, we parse a real world binary policy file, *policy.19* into a defined policy structure based on the source package of APOL [10] with our parsing engine. Then, we retrieve the information about domain types, types and related rules. Using LibXML2 [47], we transform these information into a XML file in the defined template.



```
<?xml version="1.0"? encoding="UTF-8"?>
-<policy>
-<section type="DT">
   <dtoperation start="user_t" direction="write_like" end="devtty_t" />
          <dtoperation start="user_mozilla_t" direction="write_like" end="devtty_t" />
          <dtoperation start="user_games_t" direction="write_like" end="devtty_t" />
          <dtoperation start="sendmail_t" direction="write_like" end="devtty_t" />
          <dtoperation start="sysadm_t" direction="write_like" end="devtty_t" />
          <dtoperation start="mount_t" direction="write_like" end="devtty_t" />
          <dtoperation start="fsadm_t" direction="write_like" end="devtty_t" />
          <dtoperation start="mount_t" direction="read_like" end="devtty_t" />
          .......
          <dlist d="user_t" />
          <dlist d="user_mozilla_t" />
          .......
          <dslist d="kernel_t" />
          .......
          <tlist d="devtty_t" />
          .......
 </section>
-<section type ="DD">
          <domaintrans start="sysadm_t" transdomain="mount_t"/>
          <domaintrans start="initrc_t" transdomain="mount_t"/>
          <domaintrans start="staff_su_t" transdomain="sysadm_t"/>
          <domaintrans start="newrole_t" transdomain="sysadm_t"/>
          ........
          </section>
</policy>
```

domain—type relationships
domains involved in the domain-type related paths
domains that are the starting points of the domain-type related paths
types involved in the domain-type related path
domain--domain transition relationship

Figure 3.4: Example SELinux policy rules in XML

3.2.1.2 Extended CPN to Express Policy

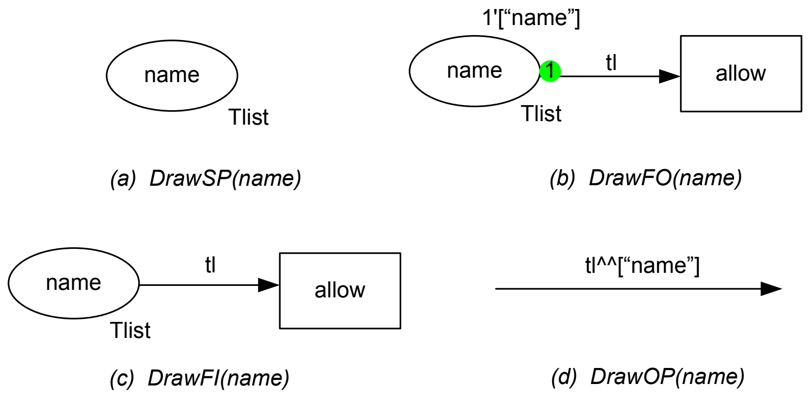


Figure 3.5: Graph drawing

To automatically visualize a policy, we use eXtensible Stylesheet Language (XSL) [33] to transform the policy in XML to CPN graph. The following basic functions are designed for the CPN graph generation with standard ML [8] as shown in Figure 3.5.

- *DrawSP(name)* is to draw the types that no information can flow into. This function draws a place that contains the initial marking whose value is the name of the place, and the arc that has a variable to express information contained in the place and the transition called *allow*. We define *dlist* as a color set of the place, which is a form of a string list containing one or more type names.

- *DrawFO(name)* is to draw the places for types from which information can flow out. It is similar to *DrawSP(name)*, but the places generated with this function do not have the initial marking.

- *DrawFI(name)* defines the function for drawing places containing the types that no information can flow out from them.

- *DrawOP(name)* is to draw an arc that connects types. The expression on the arc is to generate a list expressing flow transitions caused by domain type transitions or operations between types.

3.2.1.3 Policy in CPN Graph

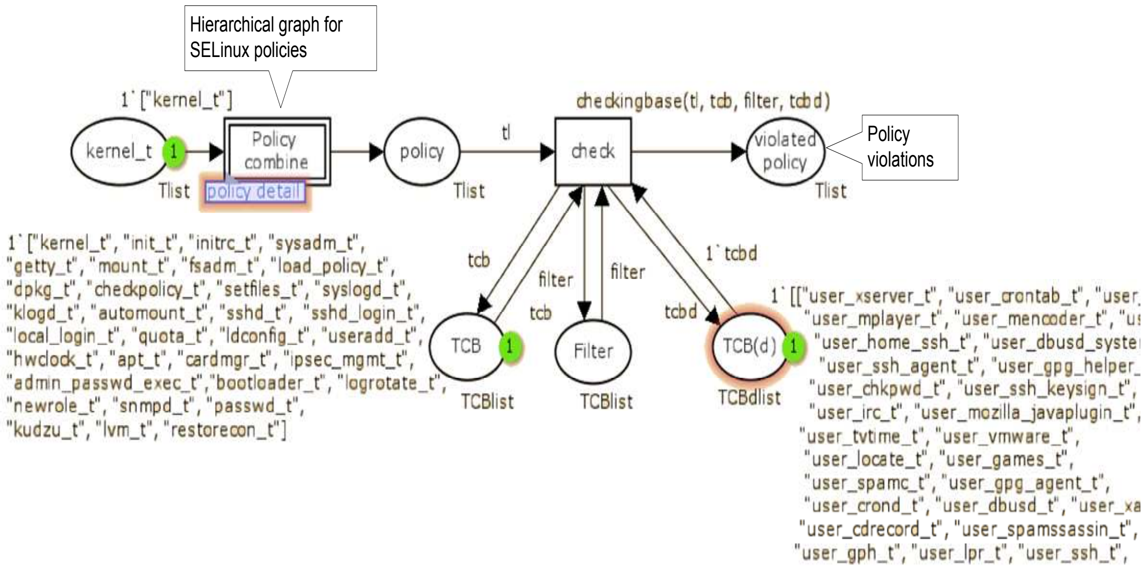

Hierarchical graph for SELinux policies
1`["kernel_t"]
kernel_t
1
Tlist
Policy combine
policy detail
policy
Tlist
tl
checkingbase(tl, tcb, filter, tcbd)
check
violated policy
Tlist
Policy violations
tcb
filter
filter
tcb
tcbd
1`tcbd
TCB
1
TCBlist
Filter
TCBlist
TCB(d)
1
TCBdlist
1`["kernel_t", "init_t", "initrc_t", "sysadm_t", "getty_t", "mount_t", "fsadm_t", "load_policy_t", "dpkg_t", "checkpolicy_t", "setfiles_t", "syslogd_t", "klogd_t", "automount_t", "sshd_t", "sshd_login_t", "local_login_t", "quota_t", "ldconfig_t", "useradd_t", "hwclock_t", "apt_t", "cardmgr_t", "ipsec_mgmt_t", "admin_passwd_exec_t","bootloader_t", "logrotate_t", "newrole_t", "snmpd_t", "passwd_t", "kudzu_t", "lvm_t", "restorecon_t"]
1`[["user_xserver_t", "user_crontab_t", "user_
"user_mplayer_t", "user_mencoder_t", "u
"user_home_ssh_t", "user_dbusd_syste
"user_ssh_agent_t", "user_gpg_helper_
"user_chkpwd_t", "user_ssh_keysign_t",
"user_irc_t", "user_mozilla_javaplugin_t",
"user_tvtime_t", "user_vmware_t",
"user_locate_t", "user_games_t",
"user_spamc_t", "user_gpg_agent_t",
"user_crond_t", "user_dbusd_t", "user_xa
"user_cdrecord_t", "user_spamssassin_t",
"user_gph_t", "user_lpr_t", "user_ssh_t",

Figure 3.6: Policy visualization with hierarchy

With the above extended CPN functions, XML-based SELinux policy rules are parsed into a CPN graph. In order to perform the process of detecting policy violations in CPN, the TCB related definitions, filter information and the rules for policy violation identification are also expressed in XML and transformed into CPN graphs. Figures 3.6 and 3.7 illustrate policy visualization results.

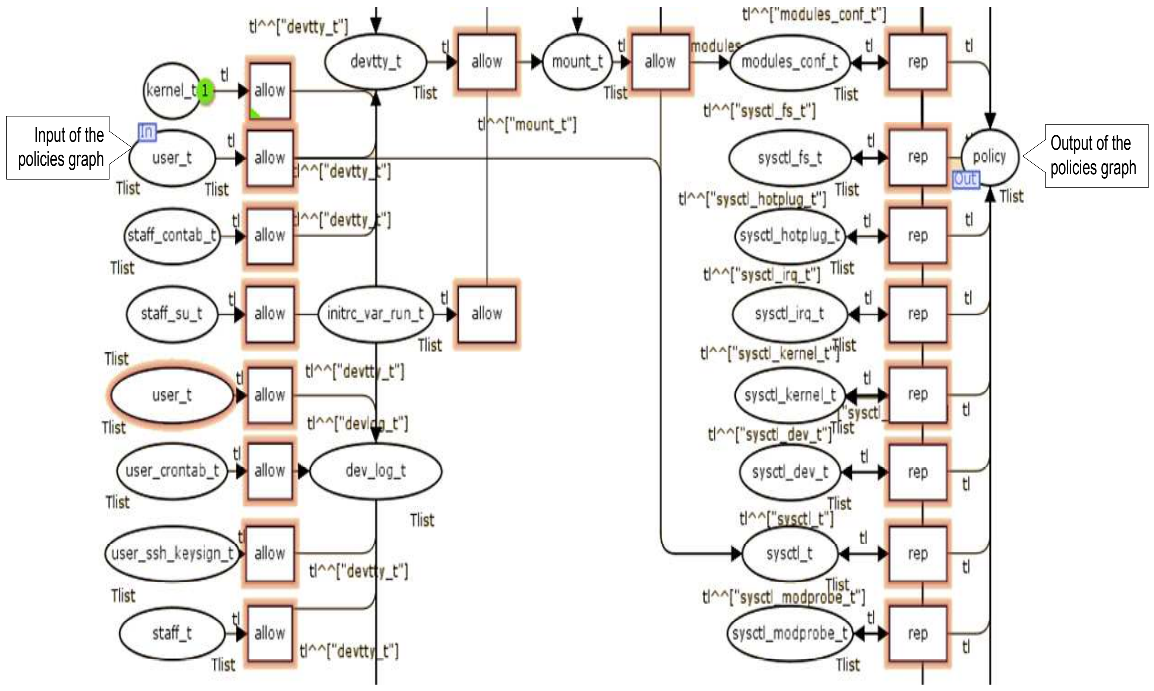


Figure 3.7: Policy visualization details

3.2.2 Policy Violation Identification

In this section, we specify how to identify the policy violations from the CPN graph. Due to the complexity of the policy and potential policy violations, we take Apache as the example to show how to identify the policy violations related to Apache from the CPN graph, instead of demonstrating our framework with an entire system.

3.2.2.1 Apache TCB and TCB(d) Identification

Based on our TCB identification strategy mentioned in the previous sections, we carry out the system TCB identification and the result is shown in Table 3.1. The collection of the TCB(d) for Apache information domain is performed based on our TCB(d) identification method defined in DIM section. Table 3.1 shows the comprehensive list of TCBs that our tool successfully identified. As stated in DIM part, filters are hard to identify, so we set initial filter values to *null* and put this information into the policy XML file. Later, when possible policy violations are detected, filters are identified and added to the XML file through CPN.

Table 3.1: Apache information domain

System TCB			
kernel_t	load_policy_t	initrc_t	bootloader_t
mount_t	ipsec_mgmt_t	useradd_t	automount_t
hwclock_t	admin_passwd_exec_t	cardmgr_t	checkpolicy_t
kudzu_t	sshd_login_t	restorecon_t	newrole_t
syslogd_t	sysadm_t	getty_t	apt_t
dpkg_t	logrotate_t	snmpd_t	ldconfig_t
lvm_t	local_login_t	setfiles_t	sshd_t
quota_t	passwd_t	fsadm_t	klogd_t
init_t			
Identified Key word-based Apache Subjects and Objects			
Apache Subjects			
httpd_staff_script_t	httpd_awstats_script_t	httpd_t	httpd_rotatelogs_t
httpd_unconfined_script_t	httpd_php_t	httpd_sysadm_script_t	httpd_sys_script_t
httpd_prewikka_script_t	httpd_apcupsd_cgi_script_t	httpd_user_script_t	httpd_suexec_t
httpd_helper_t			
Apache Objects			
httpd_staff_script_ra_t	httpd_unconfined_script_ro_t	httpd_cache_t	httpd_user_script_rw_t
httpd_user_script_exec_t	httpd_prewikka_script_ro_t	httpd_exec_t	httpd_apcupsd_cgi_script_ra_t
httpd_user_htaccess_t	httpd_apcupsd_cgi_htaccess_t	httpd_lock_t	http_port_t
httpd_sys_script_rw_t	httpd_apcupsd_cgi_script_rw_t	httpd_tmpfs_t	httpd_awstats_script_ra_t
httpd_helper_exec_t	http_cache_client_packet_t	httpd_log_t	httpd_awstats_script_ro_t
httpd_awstats_htaccess_t	httpd_awstats_script_exec_t	httpd_user_content_t	http_cache_port_t
httpd_awstats_script_rw_t	httpd_apcupsd_cgi_script_exec_t	httpd_staff_htaccess_t	httpd_sysadm_script_rw_t
httpd_sys_script_ra_t	httpd_prewikka_script_exec_t	httpd_suexec_exec_t	httpd_sysadm_script_ro_t
httpd_user_script_ro_t	httpd_unconfined_script_ra_t	httpd_php_tmp_t	httpd_php_exec_t
httpd_prewikka_content_t	httpd_prewikka_htaccess_t	httpd_staff_content_t	httpd_staff_script_ro_t
httpd_rotatelogs_exec_t	httpd_prewikka_script_ra_t	httpd_squirrelmail_t	httpd_unconfined_script_rw_t
http_server_packet_t	httpd_prewikka_script_rw_t	httpd_sys_htaccess_t	httpd_modules_t
httpd_staff_script_rw_t	httpd_sysadm_script_exec_t	httpd_tmp_t	httpd_sys_script_ro_t
httpd_sysadm_htaccess_t	httpd_staff_script_exec_t	httpd_sys_content_t	httpd_apcupsd_cgi_script_ro_t
httpd_sys_script_exec_t	httpd_unconfined_script_exec_t	httpd_config_t	httpd_suexec_tmp_t
httpd_sysadm_content_t	httpd_unconfined_content_t	http_client_packet_t	httpd_unconfined_htaccess_t
httpd_sysadm_script_ra_t	httpd_apcupsd_cgi_content_t	httpd_var_lib_t	httpd_user_script_exec_t
httpd_awstats_content_t	http_cache_server_packet_t	httpd_var_run_t	
Identified Flow-based Apache Subjects and Objects			
Apache Subjects			
applications	staff application	sysadm application	services
user application			
Apache Objects			
*_node_t (10 types)	*_port_t (116 types)	*_fs_t (38types)	*_home_dir_t (4 types)
others			
Identified Apache TCB(d)			
httpd_suexec_t	httpd_awstats_script_t	httpd_t	httpd_helper_t
httpd_sysadm_script_t	httpd_prewikka_script_t	httpd_rotatelogs_t	httpd_apcupsd_cgi_script_t
httpd_php_t			

3.2.2.2 Policy Violation Identification

Simulation is a technique supported by CPN to analyze a system by conducting controlled experiments [37]. In our experiments, we utilize the simulation feature to generate policy violations with text expressions. To better understand the violations, we visualize the generated expressions with another XML transformation. In our simulation, policy analysis result is stored in a simulation report. Through parsing the simulation report, we generate the information about policy violation specifications and produce a new XML file with the similar XSL algorithm. The generated XML file is transformed into another CPN graph for visualization. Figure 3.8 partially shows the identified policy violations, specifying that some NON-TCB domain types can write to the type `devtty_t`, which can be read by some TCB types.

3.2.2.3 Policy Violation Resolving

Due to the limitation of CPN graph expression, it is not easy to view and understand the policy and policy violations. In other words, we can only give several example resolutions of policy violations. Based on the identified policy violations, the integrity of TCB and TCB(d) is identified since NON-TCB subjects have write_like operations on objects that can be read by TCB(d) subjects. Retrieving the method of policy violation resolution, we resolve the policy violations with following methods.

- *Modify Policy Rules:* Many policy violations are caused because related subjects or objects have too much privileges. Hence, rather than just removing related policy statements, we also need to replace these subjects or objects with more restricted rights. For example, for policy violations caused by read and write accesses to `device_t`, our resolution is to redefine `device_t` by introducing `staff_device_t`, `user_device_t`, and `http_device_t`. Corresponding policy rules are also modified as follows:

```
allow httpd_t device_t:chr_file {ioctl read getattr lock write
```

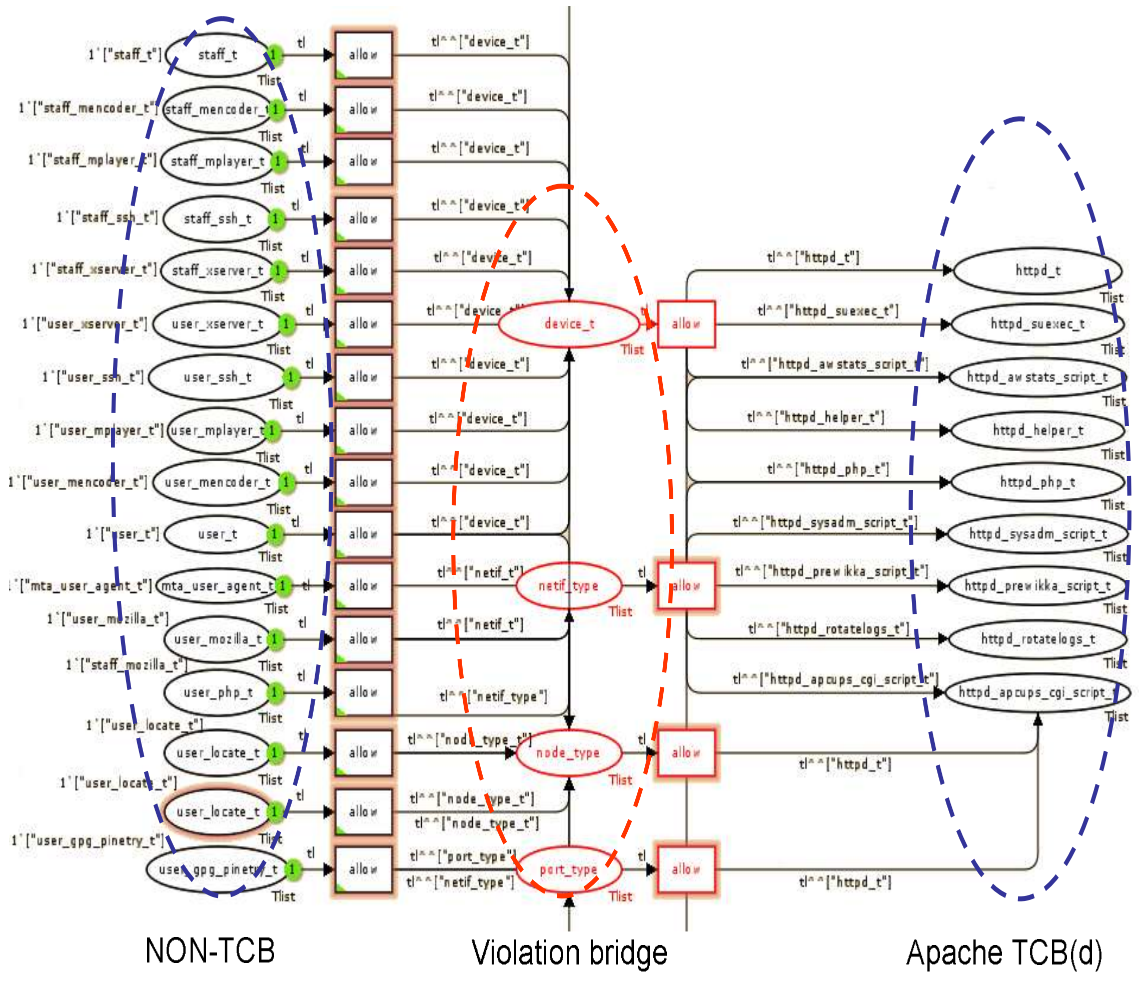


Figure 3.8: Policy violations

append}; is changed to

```
allow httpd_t http_device_t:chr_file {ioctl read getattr lock
write append};
```

- *Add Filter*: Based on the domain-based integrity model, a filter can be introduced into policies to remove policy violations. For example, to remove the violations caused by `port_t`, we introduce a network filter subject as follows:

```
allow user_xserver_t networkfilter_t:process transition;
allow networkfilter_t port_t:tcp_socket {recv_msg send_msg};
```

After the modification is applied, the original policy violations are eliminated. In

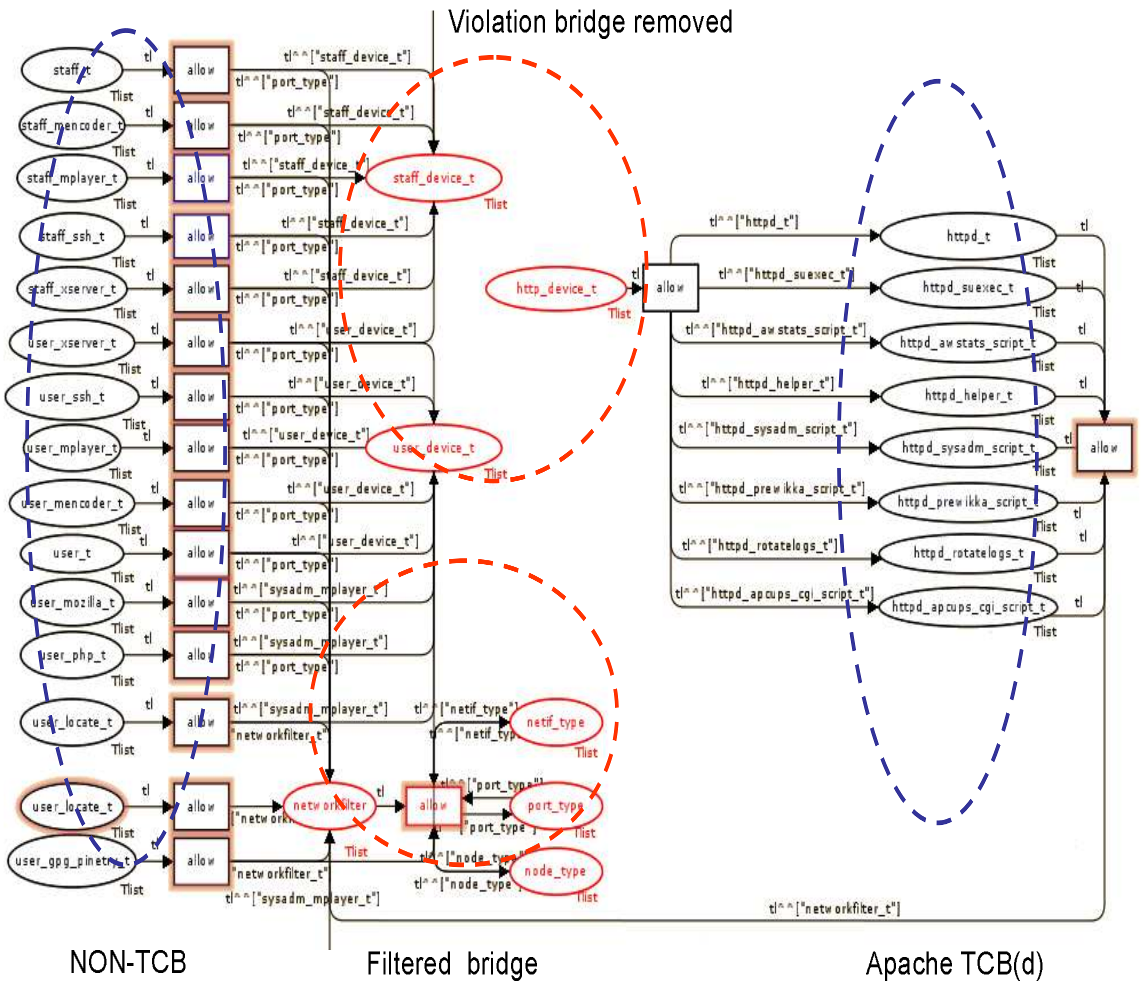


Figure 3.9: Policy violation resolution

general, to validate the result of a policy modification, we recheck the relationships between the policy violation related domains and types. Comparing Figure 3.9 with Figure 3.8, we can observe that all read operations between TCB(d) and type `port_t` are removed. Also, the write operations between NON-TCB and `port_t` are also removed. Instead, a new domain `networkfilter_t` is added, which has write and read operations on `port_t`. Also, all TCB(d) and NON-TCB subjects can transit to this new domain type.

3.2.3 Discussion

In this Chapter, we have proposed a domain-based isolation model to describe the access control requirements. In particular, our general method showed how we can identify system TCBs and domain TCBs regard to information domains in a system, and present a set of rules to detect all possible policy violations from NON-TCB to system TCB and between domain TCBs based on domain-based isolation model. We also automated the analysis processes using CPN as well as visualized graph-based violation detection. We used SELinux policy as an example to show the functionality and effectiveness of our methodology.

However, there are several limitations about using CPN for policy analysis.

- *Manual identification of TCB and TCB(d)* CPN does not provide an effective way for identifying the information domain and domain TCB which are proposed in the domain-based isolation model. Hence, we need to manually identify those information, which are very hard for the policy administrator to handle.

- *Limited graph view of policy and policy violation, and difficulty of policy violation resolution* Due to the huge size of the policy and policy violations, the CPN graphs get complicated, the lines in the graph are crossed and difficult to view. Hence, the CPN graph view cannot clearly give the policy administrator the intuitive understanding about policy and policy violations. Consequently, the policy administrator is difficult to effectively resolve policy violations.

- *Difficult to generate the policy graph*: Although we have successfully designed and implemented the functions for automatically generating the policy CPN graph, due to the complexity of policy, the generation of policy CPN graph was very exhausted.

- *Difficult to identify the policy violations*: Due to the limitation of CPN, the generated policy violations are expressed in a plain text. It has to be transformed into

CPN graph again based on our designed functions. This brings an unexpected overhead in terms of policy violation generation and policy transformation costs, e.g., the generation of policy violation information costs around 6 minutes and the transformation of policy violation information costs around 1 minutes.

CHAPTER 4: POLICY VISUALIZATION

Information visualization leverages highly-developed human visual systems to achieve rapid uptake of abstract information. In our framework we use information visualization techniques to visualize the policy to enable the system administrator to better understand the configured policy. In this Chapter, we first define the policy graphs, information flow graphs and policy violation graphs. Based on these definitions, we present our proposed semantic substrates and adjacency matrix-based policy visualization mechanisms. To facilitate the policy violation identification, we develop a graph-based query mechanism based on the policy visualization graphs.

4.1 Policy Related Graph Definition

4.1.1 Policy Graph

As defined in Chapter 3, a security policy consists of a set of subjects, objects, and operations including *write, read* and *call*. Hereby, we define a policy graph as follows:

Definition 2 *:* A *Policy Graph* of a system is a directed graph G=(V,E), where the set of vertices V represents all subjects and objects in the system, and the set of edges E=$V \times V$ represents all information flow relations between subjects and objects. That is,

- V=$V_o \bigcup V_s$, where V_o and V_s are the sets of nodes that represent objects and subjects, respectively;
- E=$E_r \bigcup E_w \bigcup E_c$. Given the vertices v_{s1},$v_{s2} \in V_s$ separately representing subject s1 and s2, and vertices $v_o \in V_o$ representing object o, $(v_{s1}, v_o) \in E_w$ if and only if $write(s1,o)$, $(v_o, v_{s2}) \in E_r$ if and only if $read(o,s2)$, and $(v_{s1}, v_{s2}) \in E_c$ if and only if $call(s1,s2)$.

4.1.2 Information Flow Graph

Based the specification of information flow in domain-based isolation model, here we arrive the definition of flow transitions and flow path as follows.

Definition 3 *:* In a policy graph G=(V,E), for any $s_1,s_2 \in V$, an information flow transition $flowtrans(s_1,s_2)$ exists if:

- $\exists o \in V, write(s_1,o) \wedge read(o,s_2)$; or
- $call(s_1,s_2)$.

We also say that predicate $flowtrans(s_1,s_2)$ is true if $flowtrans(s_1,s_2)$ exists.

Definition 4 *:* In a policy graph G=(V,E), an information flow path $flowpath(s_1,s_n)$ exists (and predicate $flowpath(s_1,s_n)$ is true) if:

- there exists $flowtrans(s_1,s_n)$; or
- $\exists s_i \in V$, $flowpath(s_1,s_i) \wedge flowpath(s_i,s_n)$.

4.1.3 Policy Violation Graph

Based on the definition of policy graph and information flow path, we also arrive the policy violation graph definition as follows.

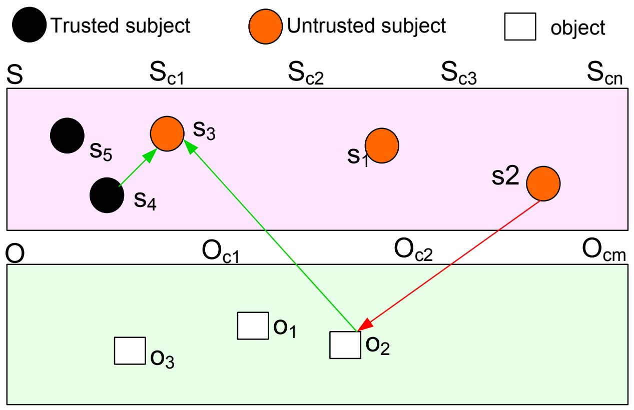
Definition 5 *:* Given a policy graph $G = (V,E)$, the subject vertices belonging to NON-TCB, system TCB, and TCB(d) are represented by V_{NTCB}, V_{TCB}, and V_{TCBd}, respectively. A violation policy graph $G^v = (V^v,E^v)$ for domain d is a subgraph of G where

- $V^v = \{v : v \in V_{NTCB}, \exists u : u \in V_{TCB} \cup V_{TCBd} \wedge (v,u) \in E\}$
- $E^v = \{(u,v) : u,v \in V^v \wedge (u,v) \in E\}$

4.2 Policy Visualization Mechanisms

To visualize the policy and policy violations based on the above definitions, we mainly adopt two kinds of mechanisms including semantic substrates-based policy visualization and adjacency matrix-based policy visualization.

4.2.1 Semantic Substrates-based Policy Visualization



Policy Graph: Links between S_2 and O_2 represents write operation; between S_3 and O_2 expresses read operation; between S_4 and S_3 denotes call operation

Figure 4.1: Semantic substrate-based policy visualization

```
Algorithm 1: [Building Semantic-based Policy Graph]
Input: The Policy file Policy, the Policy Explanation File $F_e$, the Permission Mapping File $F_p$, the Subject Classification File $F_s$, the Object Classification File $F_o$.
Output: A Semantic-based Policy Graph G
Method:
(1) Policy_t: = policyParsing( Policy,$F_e$, $F_p$, $F_s$, $F_o$);/* parsing the policies files into subjects, objects and relationships , and mapping the classification into the parsed policies.
(2) G = drawCanvas (Policy_t); /* constructing the canvas for drawing the graphs and also dividing the graphs into different areas.
(3) G = drawNodes (G , Policy_t); /* reading the entities for the policies and drawing them in nodes into the classified areas based on the Policy_t structure.
(4) G = drawLines (Policy_t, G , n); /* drawing the link from the node to the other nodes and setting the attribute for the link.
```

Figure 4.2: Algorithm for building semantic substrate-based policy graph

Several visualization studies concluded that humans perceive data coded in spatial dimensions far more easily than those coded in non-spatial ones [30, 18]. Based on this concept, we use semantic substrates to display policies. We divide a canvas into different areas based on the classification of entities (subjects and objects) and then layout nodes expressing the entities into corresponding areas. We also use non-spacial cues (e.g., color or shape) to emphasize certain nodes or a group of nodes. Figure 4.1 shows the semantic substrates-based graph design. The Y-axis is divided into regions, where each region contains nodes representing entities such as subjects and objects. Furthermore, in each region, nodes representing entities of different classifications are placed in different spaces along with the X-axis. For subjects and objects in a policy, $S_{c1}...S_{cn}$ and $O_{c1}...O_{cm}$ separately represent certain classifications. Different colors and shapes are used to distinguish the identification of different nodes. Circles and rectangles are used to represent subjects and objects, respectively. Relationships between subjects and objects are expressed with lines in different colors or shapes. For instance, the write operation between a subject s_2 and an object o_2 is expressed with a red link.

Different security policies have different formats and components. To give a uniformed way for policy analysis, we need to preprocess a primitive policy. Figure 4.2 summarizes the procedures of policy graph representation. First, a policy file *Policy* is parsed and mapped through a policy explanation file F_e and a permission mapping file F_p. F_e includes meta information such as the format of the policy and subject/object attributes in the policy. The policy format can be binary or text and is organized in a certain order. The subjects are users or processes and objects are system resources such as files, data, port or labels specifying these resources. F_p states operations between subjects and objects that are mapped to *write()*, *read()*, or *call()*. For instance, if a subject has an operation to get the attribute of an object, the operation is mapped to *read()*. In addition, F_s and F_o files separately define subject and object classifications in the system. After parsing the policy, a canvas is drawn and divided into different areas, on which nodes representing policy entities are drawn and relationships between them are expressed with arrows. Also, during the execution of this algorithm, policy rules are stored as attributes for corresponding graph nodes and arrows.

In addition, different colors and shapes in the policy violation graphs are used to help the identification of different nodes, for example, red circles, black circles and red squares are used to represent trusted domains (TCB and TCB(d)), untrusted domains and protected types respectively.

4.2.2 Adjacency Matrix-based Visualization

The semantic substrates is a very good choice for path finding for the case that the links are not heavily crossed or tangled. For visualizing a path in a dense policy graph we propose to use an adjacency matrix approach [58, 39]. We further enhance the path visualization capabilities of the adjacency matrix approach by adding new characteristics of direction. We also develop a direction based detection that enables the administrator to intuitively trace the visualized paths.

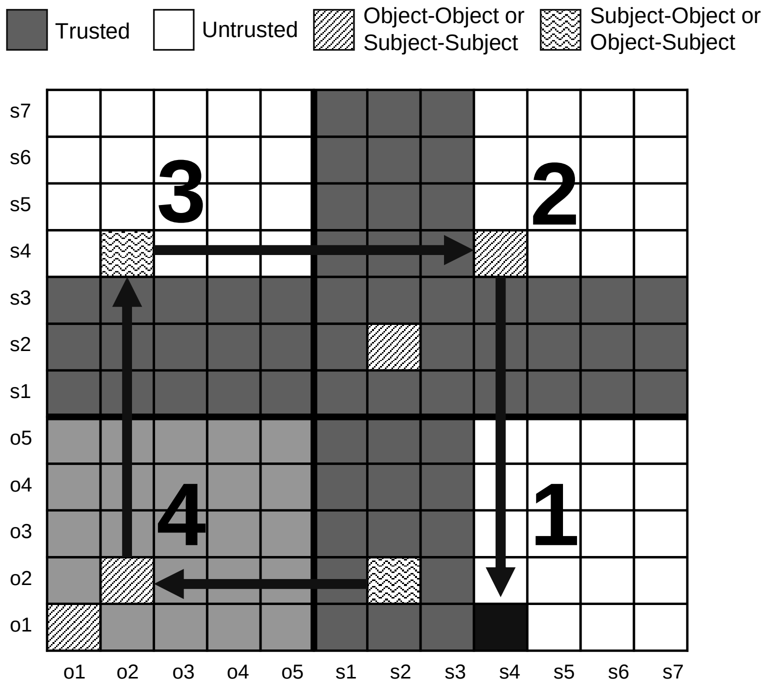


Figure 4.3: Adjacency matrix-based policy visualization

The adjacency matrix is based on the policy graph definitions. Figure 4.3 shows our proposed adjacency matrix visualization template. The nodes are arranged on both the

X-axis and the Y-axis. To visualize a path $P = \{v_0, v_1, \ldots, v_n\}$ in the adjacency matrix, we highlight entries (v_i, v_i) and (v_i, v_{i+1}), for $i = 0, \ldots, n-1$. We draw an arc from entries (v_i, v_i) and (v_i, v_{i+1}) for $i = 0, \ldots, n-1$, and we draw an arc from entries (v_{i-1}, v_i) and (v_i, v_{i+1}) for $i = 1, \ldots, n-1$. Figure 4.3 shows the visualization of path $P = \{s_2, o_2, s_4, o_1\}$. The series of arcs carry all information of the original path. In our template the subjects and objects are arranged on both the X-axis and the Y-axis. Furthermore, the grid is divided into four quadrants:

- **Quadrant 1:** A slot (s_i, o_j) signifies that a subject s_i can write to an object o_j. We refer to this quadrant as the *write quadrant*.
- **Quadrant 2:** A slot (s_i, s_j) signifies that a subject s_i can call a subject s_j.
- **Quadrant 3:** A slot (o_i, s_j) signifies that an object o_i can read by a subject s_j. We refer to this quadrant as the *read quadrant*.
- **Quadrant 4:** A slot (o_i, o_i) is used to enable the transition.

Based on the above assignments, the path $P = \{s_2, s_2, s_4, o_1\}$ represents the information flow $write(s_2, o_2)$, $read(s_4, o_2)$ and $write(s_4, o_1)$. Referring to definitions of flow transition and information flow paths, an information flow path is a sequence of write operations followed by a read operation, in our proposed adjacency matrix template. This requires the path visiting the *write quadrant* then the *read quadrant*. Therefore, information flow paths always follow a *clock-wise* direction. Using this property, an administrator can easily find the directed path information by scanning the adjacency matrix template.

Furthermore, we use different colors to represent trusted, and non-trusted entities in the adjacency matrix to express the policy violation graph. For example, red and grey represent the trusted and non-trusted entities respectively.

4.3 Querying Security Policy

Users have difficulty in writing or formulating a query [61]. The idea of the visual query formulation is to help system administrators specify precise queries on the policy base using an interactive querying technique which is based on visualization. Using an approach similar to the Query-by-Example (QBE) method for querying relational data [50, 51], a graphical user interface allows users to write queries. Our approach provides a user interface and a policy graph that enables the administrator to create and run queries against the policy base. The queries are generated by connecting our proposed query operators to formulate the intended information flows. The query operators are designed to provide functionalities adopted by the previous policy analysis mechanisms [10, 55].

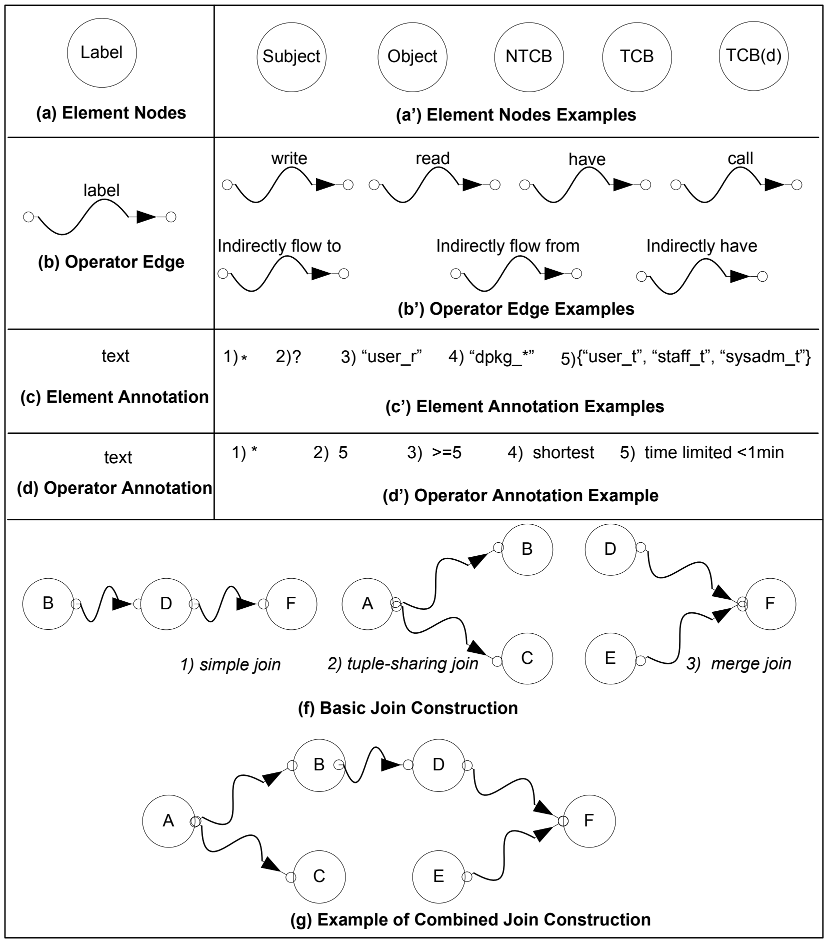

Label
(a) Element Nodes
Subject
Object
NTCB
TCB
TCB(d)
(a') Element Nodes Examples
label
(b) Operator Edge
write
read
have
call
Indirectly flow to
Indirectly flow from
Indirectly have
(b') Operator Edge Examples
text
(c) Element Annotation
1) * 2)? 3) "user_r" 4) "dpkg_*" 5){"user_t", "staff_t", "sysadm_t"}
(c') Element Annotation Examples
text
(d) Operator Annotation
1) * 2) 5 3) >=5 4) shortest 5) time limited <1min
(d') Operator Annotation Example
B
D
F
1) simple join
A
B
C
2) tuple-sharing join
D
E
F
3) merge join
(f) Basic Join Construction
A
B
C
D
E
F
(g) Example of Combined Join Construction

Figure 4.4: Query construction

4.3.1 Basic Query Formulation

Our framework provides an interactive drag and drop query platform that enables the administrators to issue information flow queries by simply connecting the provided components. Our proposed graphical query language is designed to cope with usability issues that the current policy analysis frameworks suffer [10, 55]. We define the basic visual components shown in Figure 4.4, as follows:

- *Element Nodes (E-Nodes)* are shaped as labeled circles; their label represents the attributes of the element such as subject, object, TCB, TCB(d) or NON-TCB.
- *Operator Edge (O-Edge)* is represented as the curve that connects the element nodes to another element nodes. The label of the operator edges represents the query classification of the query. Based on the query classification, the operator edges include *write, read, call, have*, *indirect have, indirect flow to, SOD* and *indirect SOD*.
- *Element Nodes Annotation (EN-Annotation)* is to specify the element nodes value. It can be a single value such as subject name, or a set of value, e.g.,{subject1, subject2....subjectn}. When the policy administrator draws the query, this value can not be omitted but can be partially specified as the wildcards "?" and "*" denote any character and any sequence of characters respectively.
- *Operator Edges Annotation (OE-Annotation)* is to specify the query requirement and determines the return graph of the query. For example, to query the information flow path from one node to the other node, we can specify to find shortest path, all the path or partial path. Here, the default (null) value represents to find all the path, which is the same as "*".

4.3.2 Query Classification

Based on the proposed domain-based isolation model, we are required to identify the domain TCB and possible information flows from NON-TCB to TCB or TCB(d). In

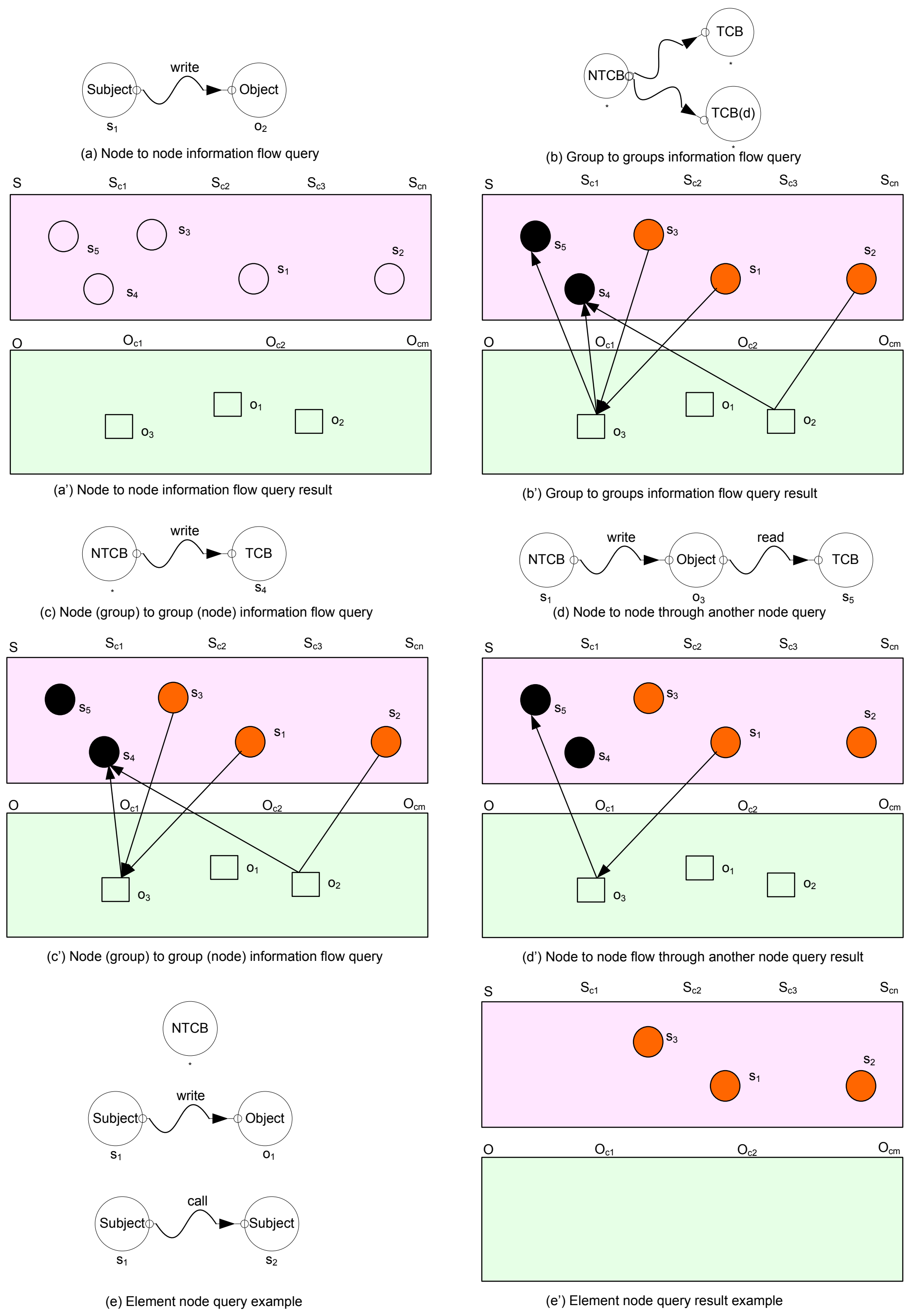

write
Subject
Object
s_1
o_2
(a) Node to node information flow query
NTCB
TCB
TCB(d)
(b) Group to groups information flow query
S
S_{c1}
S_{c2}
S_{c3}
S_{cn}
s_1
s_2
s_3
s_4
s_5
O
O_{c1}
O_{c2}
O_{cm}
o_1
o_2
o_3
(a') Node to node information flow query result
(b') Group to groups information flow query result
write
NTCB
TCB
s_4
(c) Node (group) to group (node) information flow query
write
read
NTCB
Object
TCB
s_1
o_3
s_5
(d) Node to node through another node query
(c') Node (group) to group (node) information flow query
(d') Node to node flow through another node query result
NTCB
write
Subject
Object
s_1
o_1
call
Subject
Subject
s_1
s_2
(e) Element node query example
(e') Element node query result example

Figure 4.5: Example query classification and results

addition, for the other existing query-based policy analysis tools [55], they also mainly focus on identifying information flow relationships between different nodes. Based on our formulation mechanisms, we provide a set of basic query classes that can be enabled by the policy administrator to query the policy base. These supported query classifications can satisfy the requirements including the policy analysis based on domain-based isolation model and can support the basic information flow query supported by other existing policy analysis tools.

C1. *Node to Node information flow paths.* This would enable to query an information flow from a specific subject such as NON-TCB subject to a specific object such as TCB object. The example in Figure 4.5(a') shows the query result in the form of the information flow path from a subject s_1 to an object o_2 (shown in Figure 4.5(a)).

C2. *Group to Groups information flow paths.* This would enable to query an information flow from NON-TCB subjects to TCB and TCB(d) subjects. The example in Figure 4.5(b) and (b') respectively shows a path query and the query result from NON-TCB to TCB and TCB(d).

C3. *Node (Group) to Group (Node) information flow paths.* This would enable to query an information flow from one subject such as NON-TCB subject to the objects such as TCB(d) objects, or from the NON-TCB to other subject. The example in Figure 4.5(c) and (c') respectively shows a query and the query result of finding information flow paths from all NON-TCB to the TCB subject s_4.

C4. *Node to Node information flow paths through another Node.* To find information flow from one type to another type through a certain type, where types can be subjects or objects. The example in Figure 4.5(d) and (d') respectively shows a query and the query result of finding an information flow path from a subject s_1 to a subject s_5 through an object o_3.

C5. *Element query.* This would enable to query the information domain subjects and objects and domain TCB subjects and objects. The example in Figure 4.5(e) and (e') shows a query and the query result of finding NON-TCB subjects.

4.3.3 Join Query Construction

Based on the construction of the basic policy query, we describe the join query modes which are constructed based on the shared E-Nodes. The policy administrator can use the join query to construct more complex queries such as finding the domain that can both write and read the goal protected objects. Also, using the join query the policy administrator can accumulate several query results on a single graph. Here, we summarize three main join query modes: *Simple Join, Merge Join* and *Tuple-sharing Join.*

- *Simple Join* specifies that a set of E-Nodes is connected in sequence through the O-Edges; Given E-Nodes n_i, n_j, n_k and O-Edges o_i,o_j, if $o_i(n_i,n_j)$ and $o_j(n_j,n_k)$, then we say there is a simple join.

- *Tuple-sharing Join* specifies that two or more E-Nodes are connected from the same E-Node through the O-Edges; Given E-Nodes n_i, n_j, n_k and O-Edges o_i,o_j, if $o_i(n_i,n_k)$ and $o_j(n_i,n_j)$, then we say there is a tuple-sharing join.

- *Merge Join* specifies that two or more E-Nodes are merged into one E-Node through the O-Edges; Given E-Nodes n_i, n_j, n_k and O-Edges o_i,o_j, if $o_i(n_i,n_j)$ and $o_j(n_k,n_j)$, then we say there is a merge join.

Based on the three identified join modes, the policy query can be further constructed into more complex structure as shown in Figure 4.4(f).

4.3.4 Query Execution

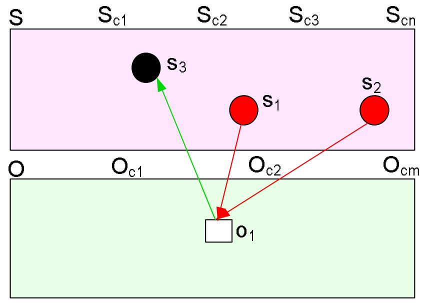
Based on the definitions of the join query construction, the identification of the different join format can facilitate the query execution. The paths are simply computed using the shortest path algorithm such as Dijkstra algorithm. The query execution makes use of

```
Algorithm [Execute Policy Query]
Input:   The Policy Query graph $G_q$
Output:  The Policy graph $G$ with query result
Method:
(1)  $N_{list} = getElementNodes(G_q)$ /* get all the nodes in query graph
(2)  FOR each $n_a \in N_{list}$ DO
(3)      $E_{list} = getConnectEdges(n_a, G_q)$ /* get all the edges of node $n_a$
(4)      FOR each $e \in E_{list}$ DO
(5)          $n_b = findConnectNode(e, n_a, G_q)$  /* get the connected node of $n_a$
(6)          If $n_b.\ getMergeNodes(n_b)$ !=NULL Then
(7)              $n_a = n_a - n_b.\ getMergeNodes(n_b, e)$ /* $n_a$ remove the duplication
caused by merge join
(8)          If $n_a.\ getTupleNode(n_a)$ !=NULL Then
(9)              $n_b = n_b - n_a.\ getTupleNode(n_a, e)$ /* $n_b$ remove the duplication caused
by tuple-sharing join
(10)         If $n_a$ !=NULL && $n_b$ !=NULL Then
(11)             $queryExecution(n_a, e, n_b, G_q)$ /* execute the query
(12)         $n_a.addTupleNode(n_b, e)$ /* save the data of $n_b$ to $n_a$ for the tuple-sharing
(13)         $n_b.addMergeNode(n_a, e)$ /* save the data of $n_a$ to $n_b$ for the merge join
```

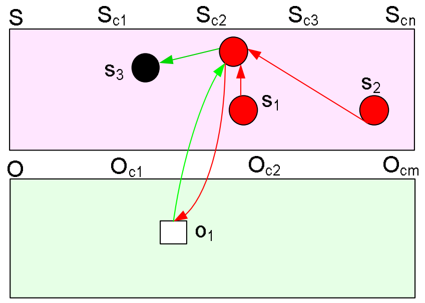
Figure 4.6: Query execution algorithm

the shared nodes between group nodes. For example, in the tuple-sharing join (shown in Figure 4.4), suppose *A* is *NTCB*, *B* is *TCB*, *C* is *fsadm_t* and the O-Edges have the same annotation. Since *fsadm_t* belongs to *TCB*, the query only needs to be executed from *A* to *B*. Similarly, in the merge join, if *D* is *NTCB* and *E* is a subset of *NTCB* (e.g. *xdm_t*) or shares labels with the NTCB. In this case the query evaluates paths from *D* to *F* then the paths from $E - (E \cap D)$ to *F*.

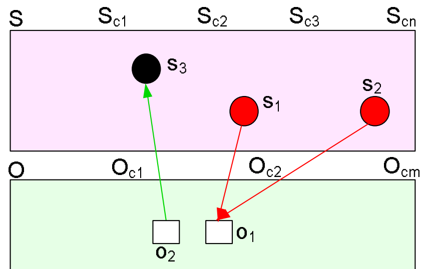
As the algorithm shown in Figure 4.6, the policy query execution algorithm is mainly composed of two main parts. In the first part, the algorithm identifies all the E-Nodes from the query graph using the function $getElementNodes(G_q)$, then for each E-Node n_a, it finds all the outgoing O-Edges connected to node n_a using $getConnectEdges(n_a, G_q)$. In the second part, for each of the identified edges e in the previous step, the algorithm identifies the connected nodes of n_b which is retrieved by the function $findConnectNode(e, n_a, G_q)$. The two cases of merge query and tuple sharing are checked and the duplication is removed. If n_a and n_b are part of the merge query, the duplicated nodes are removed from n_a by using the expression $n_a - n_b.getMergeNodes(n_b, e)$ and the merge join nodes are stored in the n_b attribute and can be retrieved with the expression $n_b.getMergeNodes(n_b, e)$. On the other hand, if n_a and n_b are part of tuple-sharing, the duplication of n_b with the expression $n_b - n_a.getTupleNode(n_a, e)$, where the tuple-sharing nodes are maintained in the n_a attribute and can be retrieved with the expression $n_a.getTupleNode(n_a, e)$. After the information duplication is removed, the query from n_a to n_b with an operator e is executed. Finally, the executed queries are added to the graph by adding the nodes and edge information into n_a and n_b respectively using functions $n_a.addTupleNode(n_b)$ and $n_b.addMergeNode(n_a)$.



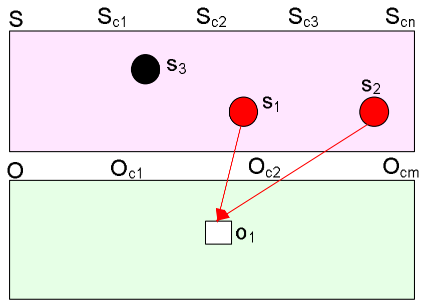
(a) Policy Violation Graph: S_1 and S_2 are NON-TCB subject which can flow to TCB or TCB(d) subject S_3 through object O_1



(b) Policy Violation Graph Solving through Filter, which can be transited by NON-TCB and TCB subjects and can read, write object O_1



(c) Policy Violation Graph Solving through adding new object O_2 which can be read by subject **s_3**



(d) Policy Violation Graph Solving through removing read operation between TCB or TCB(d) subject S_3 and O_1

Figure 4.7: Policy violation and modification example graphs

4.4 Policy Violations and Resolutions in Graph

With a generated policy violation graph, we introduce different approaches to modify the policy graph, remove policy violations and illustrate the expected graph-based result after the modification. Based on the policy violation resolution strategies discussed in DIM, other than ignoring a policy violation through adding related subjects to system or domain TCBs, we can remove the violation by importing a filter subject. Comparing with

Figure 4.7 (a) with violation resolved graph in Figure 4.7 (b), write and read operations by the NON-TCB and TCB are removed, transition relationships between subjects and the filter are added, and the policy violations caused by NON-TCB subjects S_1 and S_2 are resolved. Another optional way for resolving policy violations is to import new subjects or objects to restrict original subjects or objects privileges. As shown in Figure 4.7 (c), new object o_2 is introduced so the information flows between NON-TCB and TCB are removed. Also, we can resolve the policy violations through deleting related policy statements. As shown in Figure 4.7 (d), the read operation between an object o_1 and TCB subject S_3 is removed to resolve policy violations between NON-TCB and TCB.

CHAPTER 5: POLICY ANALYSIS AND MANAGEMENT FRAMEWORK

In this Chapter, we discuss the implementation details of our proposed framework policy visualization analysis tool (PVA). We then discuss how PVA tool can be used to identify the policy violations in the SELinux policy. Finally, we prove the usability of our tool through performing a user study.

5.1 Visualization-based Policy Analysis and Management Framework

Our policy analysis framework includes *Policy Analysis Modules*, *System Components*, and *Data Flow* for policy analysis as shown in Figure 6.1. *Policy Analysis Modules* are composed of several analysis components such as *TCB Domain Identification, Domain-based Isolation Model, and Resolution of Policy Violation* that are designed based on the policy analysis methodology in Chapter 3. For the *System Components*, the *Policy Parsing Module* is to parse and map the operations between types to *write_like*, *read_like*, and *call* operations. For example, the operation like `getattr` can be mapped to a *read_like* operation. TCB(d) is discovered and inserted into the policies using the *TCB Domain Identification* module in *Policy Analysis Modules*. In the *Policy Visualization Module*, the policies are visualized into different graphs based on the *User Input Event*. Then through executing the designed policy queries, t he *Analysis Report Module* generates the *Policy Violations Express in Graph*. In addition, this module supports the policy modification performed by *Policy Violation* module. The *Decision Flows* components govern all processes in our framework.

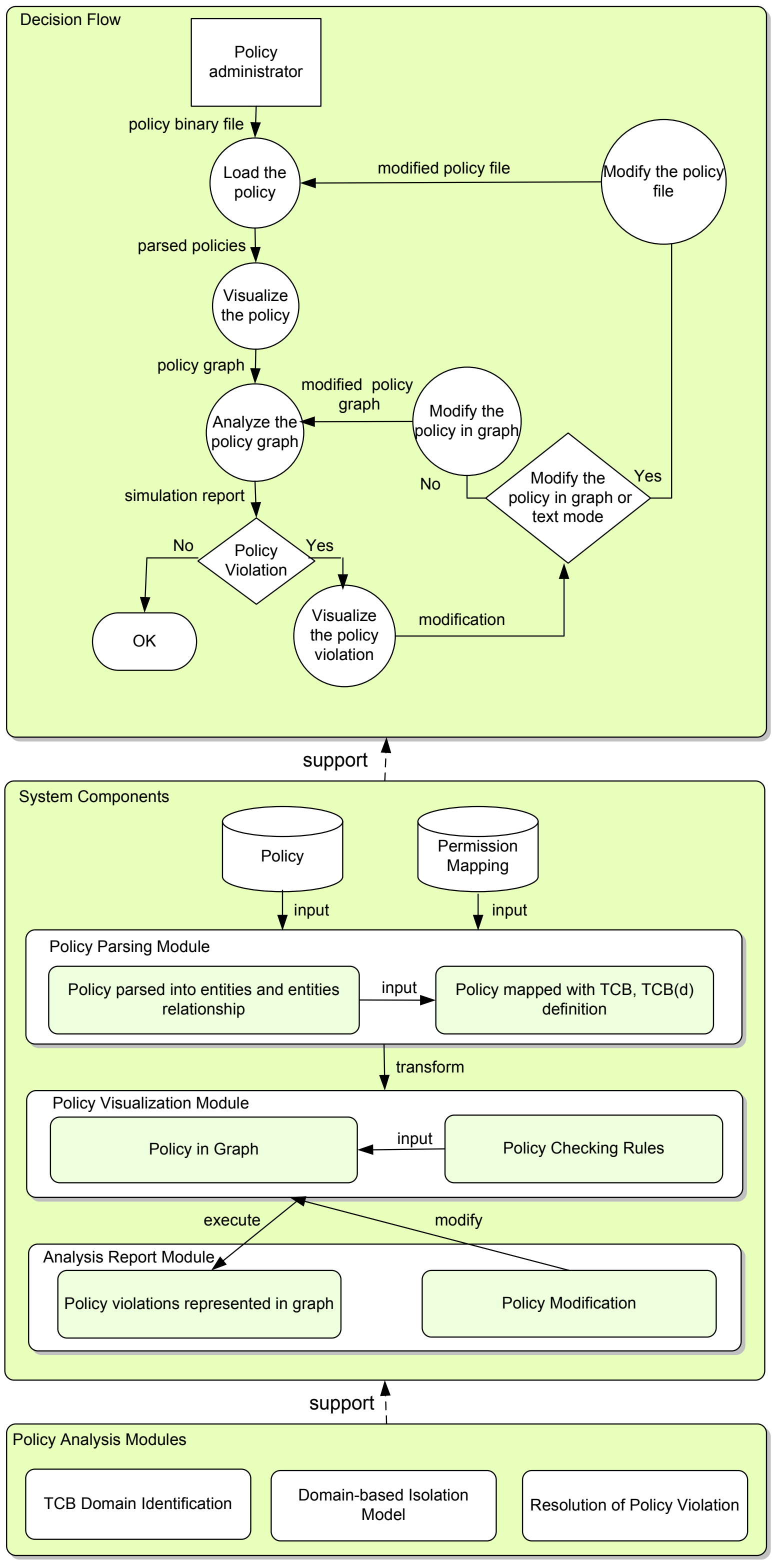


Figure 5.1: Policy visualization-based analysis and management framework

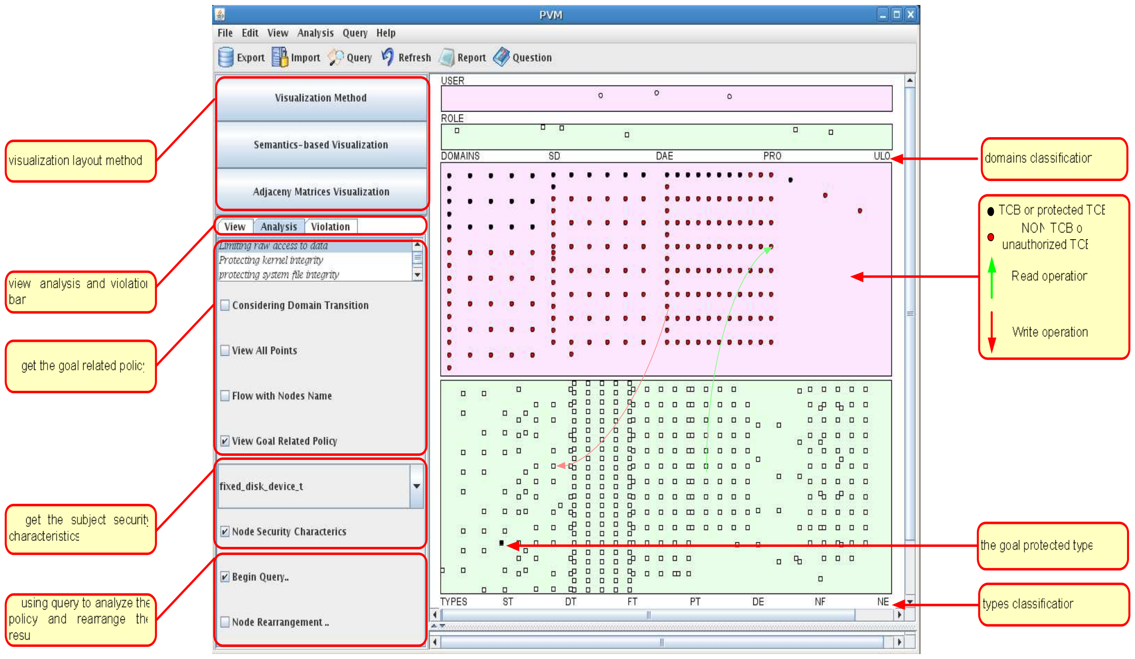
5.2 Visualization-based Policy Analysis and Management Tool (PVA)

5.2.1 Design Principles

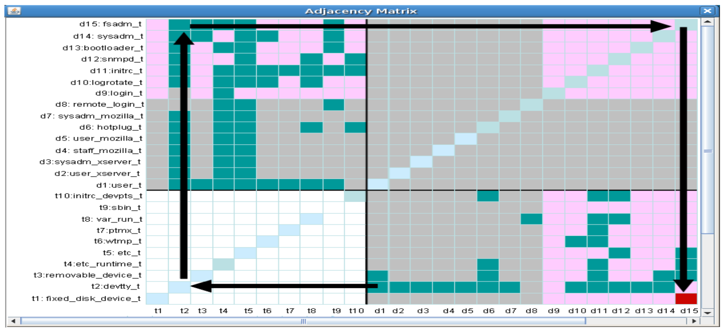
The PVA tool is based on a self-explanatory graphical user interface. To enhance the cognition and understanding of the policy information, we provide implementations of both semantic substrates-based and adjacency matrix-based visualization layouts. Another important aspect of our design is to be expressive and directly mapped to the real system policy analysis. By providing a visualization based policy query platform, our design enables the administrator to build a query by example.

5.2.2 PVA Tool Implementation Details

Our implementation is based on the Java JDK1.6 and supporting libraries. The graph drawing modules are based on our extensions to the open source package Piccolo [19]. Our parsing tool is based on the policy structure adopted by the APOL [10] tool. Figure 5.2(a) shows a snapshot of the our tool. The policy administrator can interactively import, analyze, query and modify the policy. The left window is composed of two parts: semantic substrates-based visualization and adjacency matrix-based visualization, and each window includes the tabs for *view*, *analysis*, and *violation*. The *view* tab provides the GUI for the policy graph overview, content view and detail view including zoom in and zoom out features. The *analysis* tab supports the analysis of the policy by enabling the administrator to select the security goals of interest and ultimately locate the policy violation with the help of the query function. The *violation* tab displays all the policy statements that are involved in security violations. Furthermore, in this tab the policy administrator can directly modify the policy by using the text editor or directly editing the policy graph. In the main window, the policy graphs, query results, goal related policy graphs and the policy violation graphs are displayed.


PVM
File Edit View Analysis Query Help
Export Import Query Refresh Report Question
Visualization Method
Semantics-based Visualization
Adjaceny Matrices Visualization
View Analysis Violation
Limiting raw access to data
Protecting kernel integrity
protecting system file integrity
Considering Domain Transition
View All Points
Flow with Nodes Name
View Goal Related Policy
fixed_disk_device_t
Node Security Characterics
Begin Query..
Node Rearrangement..
USER
ROLE
DOMAINS SD DAE PRO ULO
TYPES ST DT FT PT DE NF NE
visualization layout method
view analysis and violation bar
get the goal related policy
get the subject security characteristics
using query to analyze the policy and rearrange the resu
domains classification
TCB or protected TCB
NON TCB o unauthorized TCB
Read operation
Write operation
the goal protected type
types classification

a). Policy Visualization in Semantic Substrates



b) Policy Visualization in Adjacency Matrix

Figure 5.2: Policy visualization-based analysis and management tool

5.2.3 Policy Graph

The main window in Figure 5.2(a) shows the visualized SELinux policy based on our semantic substrates design, in which there are 1,337 nodes, and 11,134 links. The Y-axis is divided into four regions including USER (3 nodes), ROLE (6 nodes), DOMAINS(308 nodes) and TYPES(1,092 nodes). The X-axis is labeled with the domain and type classifications as we discussed in SELinux overview. The domain regions are divided into four different areas SD (System Domain), DAE (Daemons Domain), PRO (Program Domain) and ULO (User Login Domain). The type regions are divided into seven different areas ST (security types), DT (devpts types), FT (file types), PT (procfs types), DE (devpts types), NF (nfs types), and NE (network types). To help a policy administrator to easily identify the different regions, the elements in non-neighboring regions are represented as different shapes, for example users and domains are expressed with circle, and roles and types are expressed with rectangle. The edges between different regions are represented by different colored lines, for example the write operation between a domain and type are represented by red edges and the read operations by green edges. Also, policy administrator can view node attributes by clicking on the specific nodes. Figure 5.2(b), shows the adjacency matrix-based policy visualization method, which was compiled by selecting a subset of the nodes in the semantic substrates overlay.

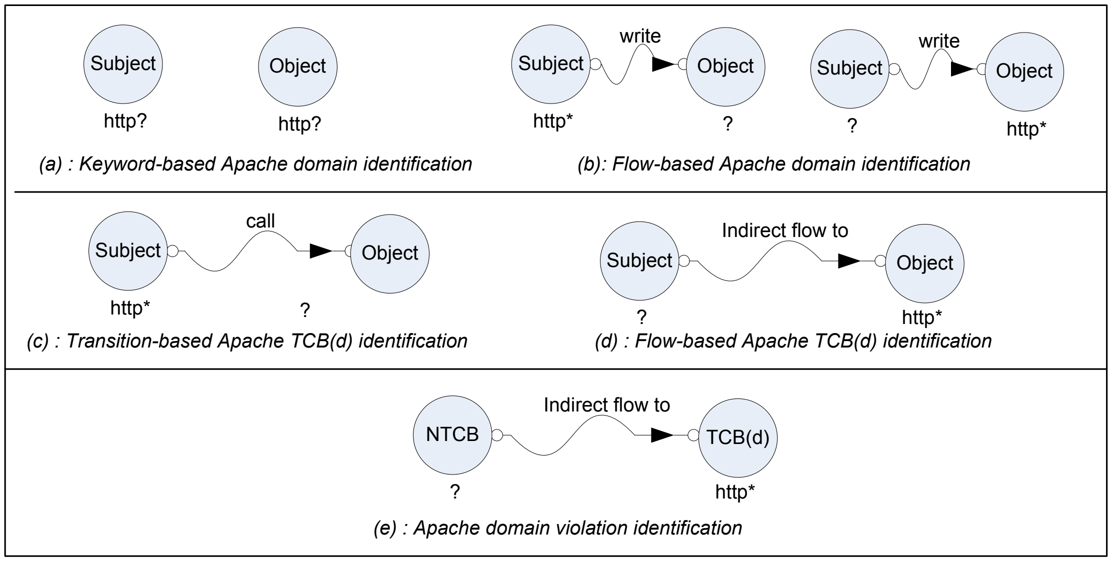
As an example, we use Apache web server as a target service domain to achieve high integrity. We first identify subjects and objects belonging to Apache domain. We then specify Apache TCB(d), list the policy violations identified against our integrity model, and resolve them with different principles.

5.2.3.1 Apache Domain Identification

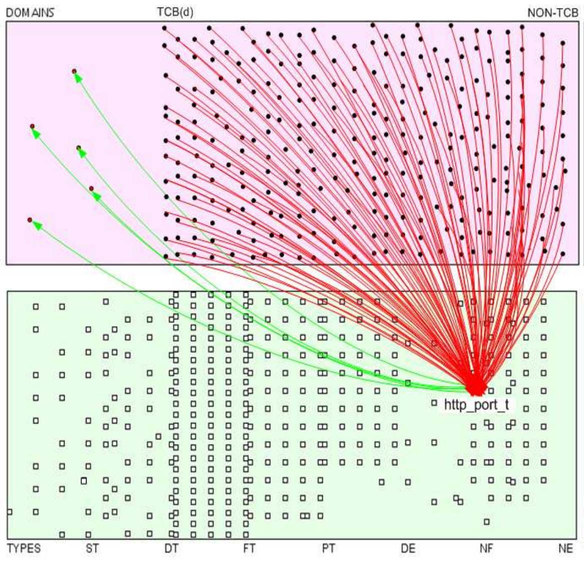
To identify subjects and objects for Apache domain, we have the following queries constructed for Apache information domain identification.

- *Key word-based policy subjects or objects query* As discussed earlier, we use *http* as a keyword prefix. As a result, subjects and objects such as `httpd_t` and `httpd_php_t` are included into Apache domain. Figure 5.3.I.(a) shows the query.

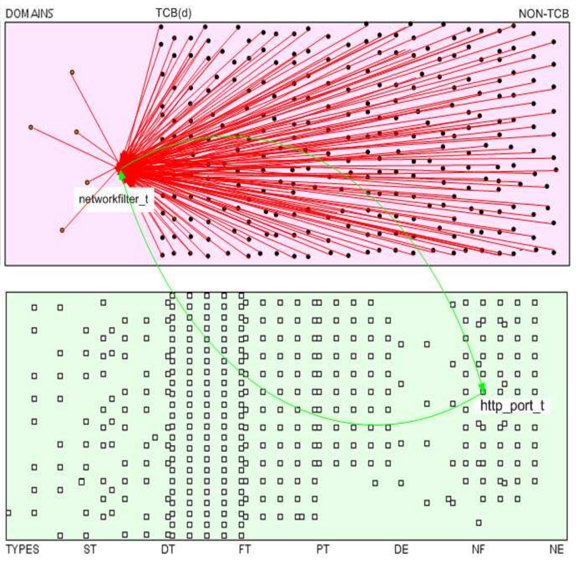
- *Direct flow-based subjects or objects query* To investigate Apache domain subjects and objects based on direct flows, we construct the query shown in Figure 5.3.I.(b). In this query, we discover all subjects and objects that have a direct flow to the initially identified Apache subjects and objects and include them into Apache domain. Table 3.1 shows a selected list of subjects and objects that we detected.


Subject
http?
Object
http?
(a) : Keyword-based Apache domain identification
Subject
http*
write
Object
?
Subject
?
write
Object
http*
(b): Flow-based Apache domain identification
Subject
http*
call
Object
?
(c) : Transition-based Apache TCB(d) identification
Subject
?
Indirect flow to
Object
http*
(d) : Flow-based Apache TCB(d) identification
NTCB
?
Indirect flow to
TCB(d)
http*
(e) : Apache domain violation identification

(I).Query examples for Apache domain



(II). Policy violations caused by NON-TCB subjects flow to TCB(d) subjects through httpd_port_t



(III). Policy violation caused by http_port_t is solved through adding networkfilter into the related policies.

Figure 5.3: Policy violation identification and modification for Apache

5.2.3.2 Apache TCB(d) Identification

For domain TCB identification steps described in DIM, we query TCB(d) for an information domain according to the following steps.

- *Transition-based subjects query* To query a TCB(d), we first identify TCB(d) based on subject dependencies considering all associated entities. Running the query shown in Figure 5.3.I.(c), we get initial TCB(d) subjects from Apache information domain. Specifically, our analysis shows that all subject domains in Apache related policy rules include a set of domain relationships since a domain `httpd_t` can transit to other `httpd` domains such as `httpd_php_t` and so on. Thus, a subject labeled with `httpd_t` is a predominant subject which launches other subjects in Apache server. Similarly, a subject labeled with `httpd_suexec_t` is also a predominant subject since this domain can transit to most entities in other `httpd` domains. Naturally, `httpd_t` and `httpd_suexec_t` are included into Apache TCB(d).

- *Indirect flow-based subjects query* To identify all subjects that can transit only to the initially identified TCB(d), we construct the query as shown in Figure 5.3.I.(d)). Based on the generated query results, `httpd_sysadm_script_t`, `httpd_rotatelogs`
`_t` and `httpd_php_t` can transit only to `httpd_t` and `httpd_suexec_t` other than system TCB subjects.

5.2.3.3 Policy Violation Queries

To generate the policy violation graph, a query is constructed as shown in Figure 5.3.I.(e), where we try to identify the policy violations from NON-TCB to TCB caused by indirect information flow. Figure 5.3 II shows the details of policy violation graph generation based on query. The query operations, NON-TCB, TCB and TCB(d) are elaborated in a file F_{tcb}. Also, NON-TCB, TCB and TCB(d) subject nodes are separately colored. Then we discover all flow transitions from NON-TCB subjects to system TCB subjects or TCB(d) subjects. Note that it is optional for a policy administrator to specify queries from NON-TCB to TCBs through specifying the exact subject names rather than using "*".

As an example of system TCB isolation violations, Table 5.1 includes all identified policy violations caused by NON-TCB subjects flowing into TCB subjects `fsadm_t`, `snmpd_t`, and `mount_t`. Also, Table 5.2 shows all policy violations identified for protecting

Apache information domain.

Table 5.1: Fsadm_t related policy violations example

Subject	Type:Class	Subject	Resolution
200	network	fsadm_t	Filter
rhgb_t, smpmount_t	mnt_t:dir	fsadm_t	Modify
hotplug_t	etc_runtime_t:file	fsadm_t	Ignore
33	unpriv_userdomain:fd use	fsadm_t	Modify
134	initrc_t:fifo_file	fsadm_t	Modify
16	removable_device_t:chr_file	fsadm_t	Modify
sysadm_cdrecord_t, user_cdrecord_t, staff_cdrecord_t	scsi_generic_device_t:chr_file	fsadm_t	Modify
200	network	snmpd_t	Filter
rhgb_t, smpmount_t	mnt_t:dir	snmpd_t	Modify
hotplug_t	etc_runtime_t:file	snmpd_t	Ignore
200	devtty_t:chr_file	snmpd_t	Modify
134	initrc_t:fifo_file	snmpd_t	Modify
131	initrc_devpts_t:chr_file	snmpd_t	Modify
200	network	mount_t	Filter
rhgb_t, smbmount_t	mnt:dir	mount_t	Ignore
hotplug_t	etc_runtime_t:file	mount_t	Ignore
16	removable_device_t:chr_file	mount_t	Modify
200	devtty_t: chr_file	mount_t	Modify
134	initrc_t:fifo_file	mount_t	Modify
3	sysadm_tty_device_t: chr_file	mount_t	Modify
131	initrc_devpts_t:chr_file	mount_t	Modify
25	sysadm_devpts_t:chr_file	mount_t	Modify
16	removable_device_t:chr_file	mount_t	Modify

5.2.3.4 Policy Violation Resolution

Based on the identified policy violations, TCB and TCB(d) integrity are violated because NON-TCB subjects have write_like operations on objects that can be read by TCB(d) subjects. Retrieving the method of policy violation resolution, we resolve the policy violations with the following methods.

Adjust TCB(d) After policy violations are identified, Apache TCB(d) is required to be adjusted and policy violations should be removed. As shown in Table 5.2, `httpd_awstats_script_t` can flow to TCB(d) subjects through `httpd_awstats_script_rw_t`. At the same time, it is flown in by many NON-TCB subjects through some common types such as `devtty_t`. Hence, we ignore the violations caused by this *awstats_script* and include it into TCB(d). Similar situation occurs for `httpd_apcupsd_cgi_script_t` and `httpd_prewikka_script_t`. However, `httpd_staff_script_t` cannot be included into TCB(d) since it would lead the unlimited file access for the staff services such as `staff_t`,

Table 5.2: Policy violations of Apache domain

Policy Violations			
NON-TCB	**Type:Class**	**TCB(d) Subject**	**Solve**
270	*_node_t: node	TCB(d) subjects	Filter
270	*_port_t: tcp_socket	TCB(d) subjects	Filter
270	netif_t: netif	TCB(d) subjects	Filter
6 subjects	dns_client_packet_t :packet	TCB(d) subjects	Filter
6 subjects	dns_port_t:packet	TCB(d) subjects	Filter
25	sysadm_devpts_t:chr_file	httpd_t	Modify
104	initrc_devpts_t: chr_file	httpd_t,httpd_rotatelogs_t	Modify
16	console_device_t: chr_file	httpd_t,httpd_suexec_t	Modify
270	devlog_t :sock_file	httpd_t,httpd_suexec_t	Modify
270	device_t:chr_file	TCB(d) subjects	Modify
270	devtty_t:chr_file	TCB(d) subjects	Modify
3	sysadm_tty_device_t:chr_file	httpd_t	Modify
5	urandom_device_t:chr_file	httpd_t	Modify
270	zero_device_t:chr_file	TCB(d) subjects	Modify
134	initrc_t:fifo_file	TCB(d) subjects	Modify
5	var_run_t:dir	httpd_t	Modify
72	var_log_t: dir	httpd_t	Modify
72	tmpfs_t:dir	httpd_t	Modify
httpd_staff_script_t	httpd_staff_script_*_t:file	httpd_t	Modify
httpd_user_script_t	httpd_user_script_*_t:file	httpd_t	Modify
httpd_sys_script_t	httpd_sys_script_*_t:file	httpd_t	Modify
httpd_unconfined_script_t	httpd_unconfined_script_*_t:file	httpd_t	Modify
webalizer_t	httpd_sys_content_t:file	httpd_t	Modify
httpd_apcupsd_cgi_script_t	httpd_apcupsd_cgi_script_*_t:file	httpd_t	Ignore
httpd_awstats_script_t	httpd_awstats _script_*_t:file	httpd_t	Ignore
httpd_prewikka_script_t	httpd_prewikka_script_*_t:file	httpd_t	Ignore
Further Policy Violations Example			
NON-TCB	**Type:Class**	**Adjusting Subject**	**Solve**
270	devtty_t:chr_file	httpd_prewikka_script_t	Modify
270	devtty_t:chr_file	httpd_awstats_script_t	Modify
270	devtty_t:chr_file	httpd_apcupsd_cgi_script_t	Modify

`staff_mozilla_t`, and `staff_mplayer_t`.

Remove Policy Rules Another way for resolving policy violations is to remove the related policy statements. For example, `webalizer_t` is to label a tool for analyzing the log files of web server and is not necessary to modify the information of web server. To resolve the policy violations caused due to the write access to `httpd_sys_content_t`, we remove the policy rule stating write_like operations between `webalizer_t` and `httpd_sys_content`
`_t`.

Modify Policy Rules Many policy violations are caused because related subjects or objects are given too much privileges. Hence, rather than just removing related policy statements, we also need to replace these subjects or objects with more restricted rights. For example, for policy violations caused by read and write accesses to `initrc_devpts_t`, our solution is to redefine `initrc_devpts_t` by introducing `initrc_devpts_t`, `system_initrc_devpts_t`, and `*_daemon_initrc_devpts_t`(* representing the corresponding service name). Corresponding policy rules are also modified as follows:

```
allow httpd_t initrc_devpts_t:chr_file {ioctl read getattr lock
write append}; is changed to
allow httpd_t httpd_daemon_initrc_devpts_t:chr_file {ioctl read
getattr lock write append};
```

Add Filter Based on the domain-based integrity model, a filter is introduced into policies to remove policy violations. For example, to remove the violations caused by `http_port_t`, we introduce a network filter subject as follows:

```
allow user_xserver_t networkfilter_t:process transition;
allow networkfilter_t http_port_t:tcp_socket {recv_msg send_msg};
```

After the modification is applied, the original policy violations are eliminated. In general, to validate the result of a policy modification, we recheck the relationships between domains and types associated with the policy violations. Comparing Figure 5.3.II with Figure 5.3.I, we can observe that all read operations between TCB(d) and type `http_port_t` are removed. Also, the write operations between NON-TCB and `http_port_t` are also removed. Instead, a new domain `networkfilter_t` is added, which has write and read operations on `http_port_t`. Also, all TCB(d) and NON-TCB subjects can transit to this new domain type.

5.3 User Study

To investigate the usability of PVM, we performed the user study from two perspectives. One is to focus on PVM functionality for browsing and analyzing the security policy. For this viewpoint, APOL is the only existing tool for policy browsing and analysis. Hence we compared APOL with PVM to prove the policy browsing and analysis ability of PVM.

In addition, based on DIM, PVM can identify policy violations and display them with semantic-substrates and adjacency-matrix methods, respectively. Thus, we also asked the participants to compare semantic-substrates and adjacency-matrix methods in visualizing SELinux security policies and policy violations.

5.3.1 Participant Enrolling and Characteristics

We first contacted some student participants for the participant enrollment after giving a guest lecture in the access control class. Other participants were enrolled after we invited people from the information security lab, the network lab, the visualization lab, the human computer interaction lab and system administrator office of the College of Computing and Informatics. In addition, we emailed some people and get some participants who are Linux experts or system administrators.

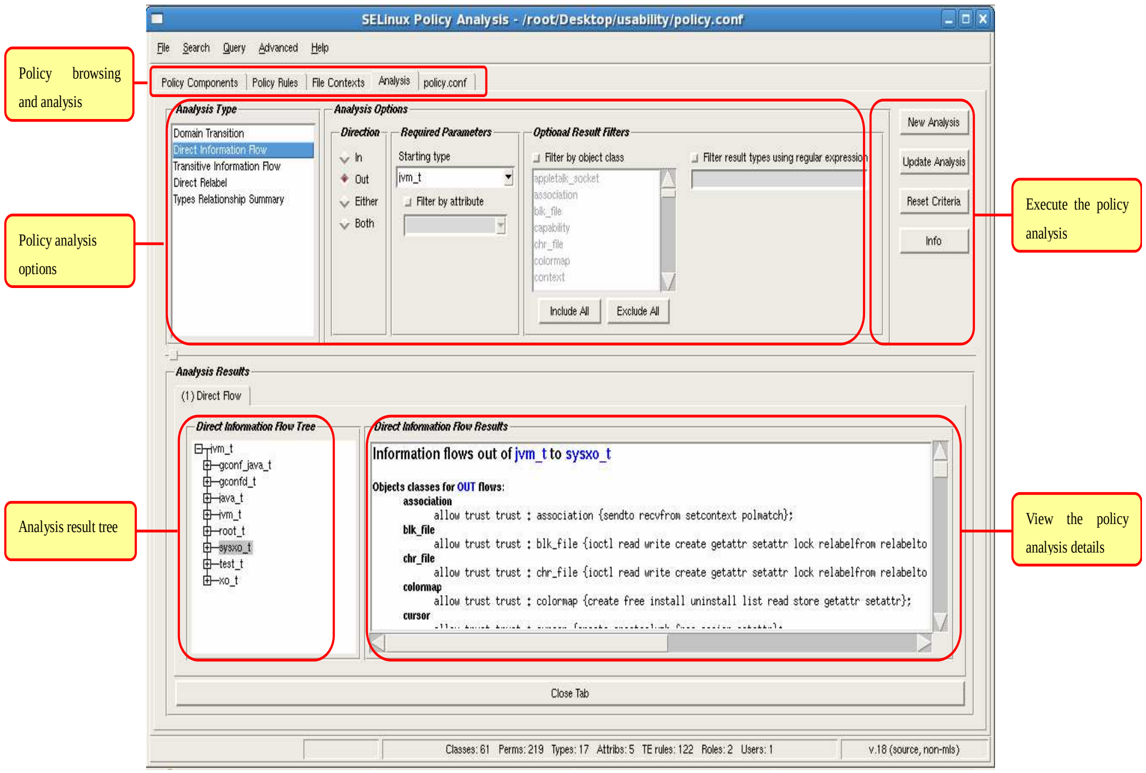
The sample for this study is consisted of 33 system administrators (15.2%), graduate students (72.7%), and undergraduate students (12.1%) with different specializations within computer science (69.7% information security; 6.1% computer networking; 6.1% database systems; 6.1% computer graphics and visualization; and 12.1% other) of various ages [12.1% were 18 $\tilde{2}2$ years old (y. o.), 42.4% are 22 $\tilde{2}6$ y. o., 30.3% are 26 $\tilde{3}0$ y. o., and 15.2% are 30 $\tilde{4}0$ y. o.]. Participants differ in terms of their most frequently used OS (15.2% Linux; 72.7% Windows; and 12.1% Mac OS), the frequency with which they change OS configuration settings (27.3% never; 54.5% monthly; 12.1% weekly; and 6.1% daily), the frequency with which they configure or check their OS security policies (30.3% never; 54.5% monthly; 6.1% weekly; and 9.1% daily), whether or not they use special software tools to manage their OS security policies (21.2% use special software tools; 63.6% do not use special software tools; and 15.2% do not know whether or not they use special software tools), the extent to which they agree that configuring security policies is an important task (3% strongly disagree; 0% disagree; 0% neither agree not disagree; 45.5% agree; and 51.5% strongly agree), and the amount of time they would be willing to spend on configuring a security policy (6.1% no time; 33.3% up to 15 minutes; 30.3% up to 30 minutes; 6.1% up to 1 hour; 3% up to 2 hours; and 21.2% more than 2 hours).

5.3.2 Policy Used

The participants were asked to analyze the same security policies with APOL and PVM. The policy we choose is real SELinux policies from major publisher. We chose this policy for the following reasons:

- It would be necessary to to simulate the realistic situation that the policy administrator might counter in practice.
- It would also essential to consider the reasonable size of security policies so that the user study in both APOL and PVM would be more convenient to the participants. The size of the security policy in our user study is approximately 200KB.

5.3.3 Instruction



```
Information flows out of jvm_t to sysxo_t

Objects classes for OUT flows:
        association
                allow trust trust : association {sendto recvfrom setcontext polmatch};
        blk_file
                allow trust trust : blk_file {ioctl read write create getattr setattr lock relabelfrom relabelto
        chr_file
                allow trust trust : chr_file {ioctl read write create getattr setattr lock relabelfrom relabelto
        colormap
                allow trust trust : colormap {create free install uninstall list read store getattr setattr};
        cursor
```

Figure 5.4: APOL tool introduction

To give ideas about how to use APOL and PVM and how to compare the two visualization mechanisms in PVM, other than communicating the SELinux policy knowledge to the participants, an instruction was provided with screen shots in two main parts (APOL and PVM sample results are shown in Figures 5.2 and 5.4), instead of explaining the SELinux policy to all participants.

In the first part of the instruction, we described how to use APOL and PVM to browse and analyze the security policy with the same tasks. The participants were instructed to load the security policy and browse any contents they were interested in. Then the participants were required to follow the instructions to complete the tasks including *(1)* identifying the information types containing `gcon`, *(2)* identifying all the direct information flow flowing out from `jvm_t` , *(3)* identifying direct information flow from `jvm_t` to `java_t`, *(4)* identifying indirect information flow from `test_t` to `user_install_t` , *(5)* identifying direct information flow from `jvm_t, test_t` to `root_t` , *(6)* identifying information flow from `jvm_t` to `sysxo_t` through `root_t` , *(7)* identifying direct information flow from group1= `{test_t, jvm_t}` to group 2=`{xo_t, root_t}` and group 3=`{user_install_var_t, user_install_exec_t}`, *(8)* repeating step 6 and step 7 to see the combined result of analysis. In addition, the participants were asked to play the tool freely to see such as how the interface was designed, how the security policy was composed, and other experimental interests.

The second part of the instruction mainly elaborated how to use semantic-substrates and adjacency-matrix methods separately to visualize the security policy, read information from the visualized policy, generate and display security policy violations against predefined security goals, and identify the reasons of the existing policy violation such as why an untrusted domain `user_install_t` can flow to a trusted domain `sysxo_t` (semantic substrates and adjacency matrix sample figures shown in Figure 5.2).

5.3.4 User Study Measurement

In the questionnaire, we design the total 20 items to get the participant attitude towards measurements including the ease of constructing queries, the ease of understanding policy analysis results and the overall ease of using the tool interface. Among these measurement, the first two items are to measure PVM and APOL in policy analysis aspect and the latter is to focus on policy browsing, understanding and the overall experience of the tool interface. For each measurement, since we designed the same items for APOL and PVM, we summarize our measurement items with one of methods-APOL–as follows.

5.3.4.1 Ease of Constructing Policy Analysis

Ease of constructing queries ($\alpha = .90$) using APOL and PVM for policy analysis was measured with 4 items. Participants were asked to rate how easy the processes described in the items using the APOL and the PVM functions are with a 5-point rating scale (1= very complicated to 5 = very easy). The four items for APOL were listed as " combining several analysis requests are: " " composing a policy analysis in APOL was:" " using APOL to analyze was:" and " figuring out what to analyze with APOL was:"

5.3.4.2 Ease of Understanding Policy Analysis Results

Ease of understanding the query results ($\alpha = .81$) using APOL and PVM for policy analysis was measured by 3 items with a 5-point rating scale (1= strongly disagree to 5 = strongly agree). The items are composed of " the analysis result in APOL can be easily investigated," " the entities relationship in the analysis can be identified easily," and " the different analysis result can be easily combined."

5.3.4.3 Overall Experience of Using the Tool Interface

Overall ease of use for APOL and PVM interfaces ($\alpha = .90$) was measured by 3 items with a 5-point scale (1= strongly disagree to 5 = strongly agree). The sample items include " browsing and understanding the security policy using APOL is:" " you would like to use APOL interface for policy analysis," and " the APOL interface shows how the different types and domains are inter-connected."

5.3.5 Measurement for Semantic-substrates versus Adjacency-matrix

To get the idea of participants experience to our visualization methods, we designed the total 18 items focusing on three aspects including ease of identifying policy violations, ease of tracing policy violation elements, and satisfaction with the policy visualization. For each measure, the same items were developed for semantic-substrates and adjacency-matrix methods. In the following, we summarize the measurement items of our user study for those visualization methods.

5.3.5.1 Ease of Identifying Policy Violations

Ease of identifying policy violations (α = .75) using the semantic-substrates and adjacency-matrix methods were measured by 3 items with a 5-point rating scale (1= strongly disagree to 5 = strongly agree). The sample items are composed of " the policy violation identification of semantic-substrates is easy to perform," " the untrusted domain and trusted domain in semantic-substrates are easy to identify," and " it is very flexible in semantic-substrates for identifying different policy violations."

5.3.5.2 Ease of Tracing Policy Violation Elements

Ease of tracing policy violation elements (α = .72) using the semantic-substrates and adjacency-matrix method were measured by 3 items with a 5-point rating scale (1= strongly disagree to 5 = strongly agree). The items include " semantic-substrates is easy to trace back domains and types included in a policy violation," " it is easy to identify transitions between trusted and untrusted domains in semantic-substrates," and " it is easy to understand the policy violation in semantic-substrates."

5.3.5.3 Satisfaction With the Visualization Policy

Satisfaction with the visualization policy (α = .73) using the semantic-substrates and adjacency-matrix methods were measured by 3 items. Participants were asked to rate the degree to which they liked different aspects of the two policy visualization methods with a

5-point rating scale (1= do not like at all to 5 = like it very much). The sample items are " viewing specific policy element relationships (domains and types)," " this visualization is intuitive and is easy to use," and " the visualization is clear, not crowded and not cluttered."

5.3.6 Procedure

To perform the user study, we installed APOL and PVM in our lab machine in an independent room without having any unexpected interruptions to the participants. The participants were required to come to our lab for the experiments. They first opened and browsed the security policies, which were then analyzed using APOL and PVM based on our instructions. Then the participants were required to visualize and identify security policy violations with semantic-substrates and adjacency-matrix methods. With these steps, the participants were required to use the same example SELinux policy.

After completing the policy analysis, participants were asked to complete a post-session questionnaire to assess their experience with the two tools and two visualization methods, their experiences with security policy analysis, their general control inclinations, and their demographic characteristics.

5.3.7 Data Collection

In our questionnaire design, we implemented the questionnaire using the lime survey tool [2] and posted it on our lab server. For each participant, our lab sever recorded their answers to the questions, whether or not they finished the questions. All these answers were stored and could be exported through a database for the later analysis.

5.4 Results

5.4.1 PVM versus APOL Results

	APOL		PVM	
Measure items	Mean	SD	Mean	SD
Ease of constructing queries	2.78	.94	4.36	.41
Ease of understanding query results	3.13	.78	4.32	.40
Overall ease of using the interface	2.41	.86	4.42	.43

Table 5.3: PVM versus APOL.

Paired samples t-tests [4] were conducted in order to compare participants' experience towards the two approaches to policy analysis. Corresponding to each measurement, we obtained the following results (shown in Table 5.3).

- *Ease of constructing queries:* Through calculating the answers from the participants for this measurement, the satisfaction tendency for APOL is (M=2.78, SD= .94), compared to PVM (M = 4.36, SD = .41). Therefore, constructing queries with PVM is perceived to be significantly easier than constructing queries with APOL (t(32) = -9.67, p < .001)

- *Ease of understanding query results:* Based on the three answers from the participants for this measurement, the satisfaction tendency for APOL is (M = 3.13, SD = .78), compared to PVM (M = 4.32, SD = .40). Hence, understanding query results with PVM is also perceived to be significantly easier than understanding queries with APOL (t(32) = -9.07, p < .001).

- *Overall ease of using the interface:* The user study indicates that the satisfaction tendency for APOL is (M=2.41, SD = .86), compared to PVM (M=4.42, SD = .43). Hence, the overall PVM interface is perceived to be significantly easier to use than the APOL interface (t(32) = -11.82, p < .001).

In summary, the participants clearly indicated that all examined aspects of the PVM approach to policy analysis are easier than the corresponding aspects of the APOL approach.

5.4.2 Semantic-substrates versus Adjacency-matrix Results

	semantic-substrates		adjacency-matrix	
Measure items	Mean	SD	Mean	SD
Satisfaction with the visualization policy	4.36	.50	4.11	.65
Ease of identifying policy violations	4.40	.40	4.17	.64
Interpretability of visualization results	4.23	.64	3.99	.70

Table 5.4: Semantic-substrates comparing Adjacency-matrix

Paired samples t-tests [4] were also conducted in order to compare participants' experience towards two visualization methods for policy analysis and policy violation visualization. Based on the measurement criteria, we obtained the following results (shown in Table 5.4) with a 5-point rating score (1= do not like at all to 5 = like it very much).

- *Ease of identifying policy violations:* Through calculating the answers from the participants for this measurement, the satisfaction tendency for semantic-substrates is (M=4.40, SD= .40), compared to adjacency-matrix (M = 4.17, SD = .64). Therefore, identifying policy violations with the semantic-substrates visualization is perceived to be significantly easier than identifying policy violations with the adjacency-matrix visualization (t(32) = 2.21, p < .05).

- *Satisfaction with the visualization policy:* Based on the answers from the participants for this measurement, although there is a tendency for participants to be more satisfied with the semantic-substrate visualization (M = 4.36, SD = .50), compared to the adjacency-matrix visualization (M = 4.11, SD = .65), this tendency is not significant (t(32) = 1.98, p = .056).

- *Interpretability of visualization results:* The user study also indicates that the satisfaction tendency for semantic-substrates is (M=4.23, SD= .64), compared to adjacency-matrix (M = 3.99, SD = .70). Therefore, interpreting the visualization results with both methods is perceived to be similar (t(32) = 1.98, p < .056).

In summary, although there is a tendency for participants' perceptions of the semantic-substrates visualization to be more favorable, compared to their perceptions of the adjacency-matrix visualization, this tendency is only significant with regards to the ease of identifying policy violations.

5.5 Lessons Learned

Our evaluations of PVM were carried out through the steps presented in the previous sections. We summarize some lessons that we have learned from the user study.

Lesson 1: Ensure that participants understand the assigned task .

Communicating clearly with survey participants and confirming the participants with the proper information before the survey were crucial. We found out that the best way to perform the user study is to repeatedly remind the participants what they understood and what they needed to do.

Lesson 2: Test materials before sending out It is very important to run a pre-test before starting the survey. In our experiment, the researchers tested all the materials including instruction, tools and questionnaire first. Then we invited few participants to test our materials and setup, so that the design of the overall experiment could be fully debugged, modified and validated.

Lesson 3: Use as little paper work as possible Initially, we designed the paper-based questionnaire but the paper work caused significant overhead for analyzing and maintaining user study data. Hence, we changed the questionnaire to be the online-based one, which brought the benefits of taking care of answering sessions, data collection, analysis, and backup.

Lesson 4: Minimize the chances that participants may make mistake Minimizing the chances that participants may make mistakes unrelated to the user study is a challenging problem. For example, some participants were confused with PVM since it displays the results on two display panes.

Lesson 5: Understand Limitations of User Study Due to the availability of current policy analysis tool and survey participants, we identified several limitations. First, we could only use the most sophisticated tool APOL to measure the effectiveness and efficiency of our PVM tool. This biased attempt would not be a fair to PVM. Second, since APOL does not have trusted and untrusted concepts in the tool, we were not able to compare this important characteristics of PVM with APOL.

CHAPTER 6: DYNAMIC REMOTE ATTESTATION

In distributed computing environments, it is crucial to measure whether remote parties run buggy, malicious application codes or are improperly configured by rogue software. Remote attestation techniques have been proposed for this purpose through analyzing the integrity of remote systems to determine their trustworthiness. Typical attestation mechanisms are designed based on the following steps. First, an attestation requester (*attester*) sends a challenge to a target system (*attestee*), which responds with the evidence of integrity of its hardware and software components. Second, the attester derives runtime properties of the attestee and determines the trustworthiness of the attestee. Finally and optionally, the attester returns the attestation result, such as integrity measurement status, to the attestee. Remote attestation can help reduce potential risks that are caused by a tampered system.

Various attestation approaches and techniques have been proposed. Trusted Computing Group (TCG) [11] specifies trusted platform module (TPM) which can securely store and provide integrity measurements of systems to a remote party. Integrity measurement mechanisms have been proposed to facilitate the capabilities of TPM at application level. For instance, Integrity Measurement Architecture (IMA) [53] is an implementation of TCG approach to provide verifiable evidence with respect to the current run-time state of a measured system. However, IMA does not provide a comprehensive attestation mechanism, which evaluates the trustworthiness of a target system with the measurements. Several attestation methods have been proposed to accommodate privacy properties [23], system behaviors [32], and information flow model [35]. However, these existing approaches still need to cope with the efficiency when attesting a platform where its *system state* frequently changes due to system-centric events such as security policy updates and software package installations. In addition, an efficient attestation mechanism should be especially considered for dealing with such a dynamic nature of systems since the frequency of system changes and the volume of system state information would be tremendously increased due to the recent technological innovation of distributed computing. Last but not least, existing attestation mechanisms do not have an effective and intuitive way for presenting attestation results and reflecting such results while resolving identified security violations.

Due to these attestation requirements, our policy analysis and management work is required to be extended. First, based on our original analysis and management framework, we are required to extend this tool to realize policy updates rather than analyzing a whole policy. Second, based on the policy violation graph identified through our policy management work, a ranking mechanism is required to prioritize the policy violations, and thus provides the method for describing the *trustworthiness* of different system states with *risk levels*. Other than these extension work, remote attestation mechanisms are also required to be proposed and developed, e.g., how the attestee can measure its codes and data and generate the policy updates, and how the attester can reliably get these attestee information.

As a summary, the following lists the objectives we would like to achieve for this part of research.

- Extend existing policy visualization-based analysis and management framework, develop a dynamic attestation framework built based on this extension, and solve the issues including verifying the system efficiently and describing the attestation result effectively.

- Demonstrate the feasibility of proposed approach through prototype implementations, which incorporate a component extended from the visualization-based policy analysis and management tool, and other remote attestation related components.

- Evaluate the performance of the attestation prototype, and prove the efficiency and effectiveness of our dynamic attestation framework.

In the follows, Section 6.1 overviews existing attestation work and system integrity models. Section 6.2 presents the design requirements and attestation procedures of DR@FT, followed by policy analysis methods and their usages in Section 6.3. We elaborate the implementation details and evaluation results in Section 6.4, and conclude this Chapter in Section 6.5.

6.1 Background

6.1.1 Attestation

The TCG specification [11] defines mechanisms for a TPM-enabled platform to report its current hardware and software configuration status to a remote challenger. A TCG attestation process is composed of two steps: (i) an attestee platform measures hardware and software components starting from BIOS block and generates a hash value. The hash value is then stored into a TPM Platform Configuration Register (PCR). Recursively, it measures BIOS loader and operating system (OS) in the same way and stores them into TPM PCRs; (ii) an attester obtains the attestee's digital certificate with an attestation identity key (AIK), AIK-signed PCR values, and a measurement log file from the attestee which is used to reconstruct the attestee platform configuration, and verifies whether this configuration is acceptable. From these steps we notice that TCG measurement process is composed of a set of sequential steps up to the bootstrap loader. Thus, TCG does not provide effective mechanisms for measuring a system's integrity beyond the system boot, especially considering the randomness of executable contents loaded by OS.

IBM IMA [53] extends TCG's measurement scope to application level. A measurement list M is stored in OS kernel and composed of m_0 ... m_i corresponding to loaded executable application codes. For each loaded m_i, an aggregated hash H_i is generated and loaded into TPM PCR, where H_0=H(m_0), and H_i=H(H$_{i-1}$ || H(m_i)). Upon receiving the measurements and TPM-signed hash value, the attester proves the authentication of measurements by verifying the hash value, which helps determine the integrity level of the platform. Also, it strives to maintain the measurement performance by caching the measurement results unless the executable contents are altered. However, IMA has the following limitations. First, IMA requires to verify the entire components of the attestee platform while the attestee may only demand the verification of certain applications. Second, the integrity status of a system is validated by testing each measurement entry independently, focusing on the high integrity processes. However, it is impractical to disregard newly installed untrusted applications or data from the untrusted network.

PRIMA [35] is an attestation work based on IMA and CW-Lite integrity model [57]. PRIMA attempts to improve the efficiency of attestation by only verifying codes, data, and information flows related to trusted subjects. On one hand, PRIMA needs to be extended to capture the dynamic nature of system states such as software and policy updates, which could be an obstacle for maintaining its efficiency. On the other hand, PRIMA represents an attestation result with binary decision (trust or distrust) and does not give semantic information about how much the attestee platform can be trusted or untrusted.

Property-based attestation [23] is an effort to protect the privacy of a platform by collectively mapping related system configurations to attestation properties. For example, "SELinux-enabled" is a property which can be mapped to a system configuration indicating that the system is protected with an SELinux policy. That is, this approach prevents the configurations of a platform from being disclosed to a challenger. However, due to the immense configurations of the hardware and software of the platform, mapping all system configurations to properties is infeasible and impractical. Semantic remote attestation [32] is proposed with a trusted virtual machine running on a platform. Through monitoring the policy attached to a running software, its behavior is verified. But this approach does not define the correct behavior of a security policy with respect to integrity, which leads the attestation to be intractable. Behavior-based attestation [15] attempts to attest system behaviors based on an application level policy model. Such a model-based approach may not comprehensively realize the complex behaviors of dynamic systems.

6.1.2 Integrity Models

As described in Chapter 2, there exist several different integrity models. However, there is a gap between concrete measurements of a system's components and verification of its integrity status. We believe an application-oriented and domain-centric approach accommodates the requirements of attestation evaluation better than advocating an abstract-level models. For example, in a Linux system, a subject in one of traditional integrity models can correspond to a set of processes, belonging to a single application domain. For instance, an Apache domain may include various process types such as `httpd_t`, `http_sysadm_devpts_t`, and `httpd_ prewikka_script_t`. All of these types can have information flows among them, which should be regarded as a single integrity level. Also, sensitive objects in a domain should share the same integrity protection of its subjects. To comprehensively describe the system integrity requirements, in this attestation work, we use the previous proposed domain-based isolation approach as our system integrity requirements.

6.2 Design of DR@FT

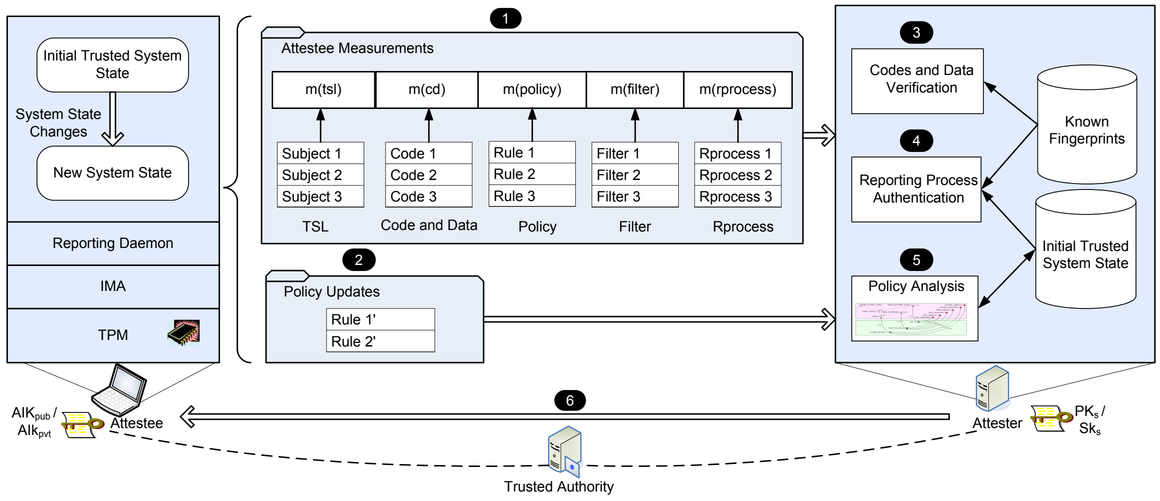


Figure 6.1: Overview of DR@FT

DR@FT consists of three main parties: an attestee (the target platform), an attester (attestation challenger), and a trusted authority, as shown in Figure 6.1. The attestee is required to provide its system state information to the attester to be verified. Here, we assume that an attestee initially is in a *trusted system state*. After certain system behaviors, the system state is changed to a new state. An attestation reporting daemon on the attestee gets the measured new state information (step 1) with IMA, and generates the policy updates (step 2). This daemon then gets AIK-signed PCR value(s) and sends to the attester. After the attester receives and authenticates the information, with the attestee's AIK public key certificate from the trusted authority, it verifies the attestee integrity through codes and data verification (step 3), reporting process authentication (step 4) and policy analysis (step 5).

6.2.1 System State and Trust Requirement

For the attestation purpose, the system state is a snapshot of an attestee system at a particular moment, where the factors characterizing the state can influence the system integrity in any future moment of the system. Based on the domain-based isolation, the attestee system integrity can be represented via information flows, which are characterized by the trusted subject list, filters, policies, and the trustworthiness of TCB_s. Based on these, we define the system state of the attestee as follows.

Definition 6 *:* A *system state* at the time period *i* is a tuple T_i={ TSL_i, CD_i, $Policy_i$, $Filter_i$, $RProcess_i$ }, where

- TSL_i={s_0, s_1....s_n} represents a trusted subject list containing a set of subjects s_0, s_1....s_n in TCB_s and TCB_d. The purpose of this trusted subject list is to specify the high integrity processes or application processes that should be protected on this platform.

- CD_i={cd (s_0), cd (s_1)....cd (s_n)} is a set of codes and data for loading a subject $s_j \in TSL_i$.

- $Policy_i$ is the security policy currently configured on the attestee.
- $Filter_i$ is a set of processes that are defined to allow information flow from low integrity processes to high integrity processes.
- $RProcess_i$ represents a list of processes that measure, monitor, and report the current TSL_i, CD_i, $Filter_i$ and $Policy_i$ information. IMA agent and the attestation reporting daemon are the examples of the $RProcess_i$.

According to this definition, a system state does not include a particular application's running state such as its memory page and CPU context (stacks and registers). It only represents the security configuration or policy of an attestee system. A system *state transition* indicates one or more changes in TSL_i, CD_i, $Policy_i$, $Filter_i$, or $RProcess_i$. A system state T_i *is trusted* if

- TSL_i belongs to TCB_s and TCB_d;
- CD_i does not contain untrusted codes and data;
- $Policy_i$ satisfies domain-based isolation;
- $Filter_i$ belongs to the defined filter in domain-based isolation; and
- $RProcess_i$ codes and data do not contain malicious codes and data and these $RProcess_i$ processes are integrity protected from the untrusted processes via $Policy_i$.

As mentioned, we assume there exists an initial trusted system state T_0 which satisfies. Through changing the variables in T_0, the system transits to states T_1, T_2 ... T_i. The attestation purpose is to verify if any of these states is trusted.

6.2.2 Attestation Procedures

Attestee Measurements The measurement at the attestee side has two different forms, depending on *how much* the attestee system changes. In case any subject in TCB_s is updated, the attestee must be fully remeasured from the system reboot and the attester needs to attest it completely, as this subject may affect the integrity of subjects in *RProcess* of the system such as the measurement agent and reporting daemon. After the reboot and all TCB_s subjects are remeasured, a trusted initial system state T_0 is built. To perform this re-measurement, the attestee measures a state T_i and generates the measurement list M_i containing each measurement.

- Trusted subject list (TSL_i) measurement: With TPM and measurement mechanisms such as IMA, trusted subject list TSL_i is measured with a result m_{tsl}, which is added to a measurement list by $M_i = M_i||m_{tsl}$. $\mathscr{H}(M_i)$ is extended to a particular PCR of the TPM, where $\mathscr{H}$ is a hash function such as SHA1.

- Codes and data (CD_i) measurement: For every subject s_j in TSL_i, its codes and data $cd(s_j)$ are measured. To specify the mapping relationship, measurement m_{cdsj} consists of the information of $cd(s_j)$ and s_j. After m_{cdsj} is generated, it is added to the measurement by M_i=$M_i||m_{cdsj}$ and its hash value is extended to PCR.

- Policy ($Policy_i$) measurement: Corresponding to $Policy_i$, the attestee generates the measurement m_p, which is added to a measurement list with M_i=$M_i||m_p$, and corresponding PCR is extended with its hash value.

- Filter ($Filter_i$) measurement: The codes and data of $Filter_i$ are measured as m_f, which is extended to a measurement list M_i=$M_i||m_f$, and the PCR is extended.

- Attestation Process ($RProcess_i$) measurement: The codes and data of $AProcess_i$ are measured as m_r, which is added to a measurement list M_i=$M_i||m_r$, and the PCR is also extended.

In another case, where there is no TCB_s subject updated and the TSL_i or $Filter_i$ subjects belonging to TCB_d are updated, the attestee only needs to measure the updated codes and data loading the changed TSL or filter subject, and generates a measurement list M_i. The generation of this measurement list is realized through the run-time measurement supported by the underlying measurement agent.

To support both types of measurements, we develop an attestation reporting daemon which monitors the run-time measurements of the attestee. In case the run-time measurements for the TCB_s are changed, the attestee is required to be rebooted and fully measured with IMA. The measurements values are then sent to the attester by the daemon. On the other side, the changed measurement value is measured by IMA and captured with the reporting daemon only if the measurement for TCB_d is changed. Obviously, this daemon should be trusted and is included as part of TCB_s. That is, its codes and data are required to be protected with integrity policy and corresponding hash values are required to be stored at the attester side.

Policy Updates To analyze if the current state of the attestee satisfies domain-based integrity property, the attester requires information about the current security policy loaded at the attestee side. Due to the large volume of policy rules in a security policy, sending all policy rules in each attestation and verifying all of them by the attester may cause the performance overhead. Hence, in DR@FT, the attestee only generates policy updates from the latest attested trusted state and sends them to the attester for the attestation of such updates.

To support this mechanism, we have the attestation reporting daemon monitor any policy update on attestee system and generate a list of updated policy rules. The algorithm of this operation is shown in Algorithm 1. With a function *threadMonitor*, upon the system policy $Policy_{i-1}$ is changed on state T_{i-1}, the daemon process reads the policy rules in the stored security policy $Policy_0$ of the trusted system state T_0 and the current policy $Policy_i$ separately, and compares these rules with a function *compareResult*. Then the daemon finds the added, changed and deleted rules through functions *added, changed, deleted* separately, and saves them into the policy update file. Note that the policy comparison is performed between the current updated policy and the stored trusted security policy

$Policy_0$ or previously attested and trusted $Policy_{i-1}$. The complexity of this policy update algorithm is *O(nr)*, where *nr* represents the number of the policy rules in the new policy file $Policy_i$.

Algorithm 1: Generating Policy Updates

```
Input: Currently configured policy file $P_i$, stored trusted policy file $P_0$ , currently
       running updates thread $U_s$
Output: Policy updates file $P_{updates}$
1  $state := threadMonitor(U_s, P_i)$
2  if $isChanged(state)$ then
3      $fileFlow := Initialize(P_{updates})$
4      $R_{list} := readPolicy(P_i)$
5      foreach $r_i \in R_{list}$ do
6          $compareResult := (r_i, P_0)$
7          if $compareResult == 1$ then
8              $added(fileFlow, r_i)$
9          else
10             if $compareResult == 2$ then
11                 $changed(fileFlow, P_0, r_i)$
12                 $markPolicy(P_0, r_i)$
13             else
14                 $markPolicy(P_0, r_i)$
15 $R_{list} := readPolicy(P_0)$
16 foreach $r_i \in R_{list}$ do
17     if $isnotMarked(P_0, r_i)$ then
18         $deleted(fileFlow, r_i)$
```

Codes and Data Verification With received measurement list M_i and AIK-signed PCRs, the attester first verifies the measurement integrity by re-constructing the hash values and compares with PCR values. After this is passed, the attester performs the analysis. Specifically, it obtains the hash values of CD_i and checks if they corresponds to known-good fingerprints. Also, the attester needs to assure that the TSL_i belongs to TCB_s and TCB_d. In addition, the attester also gets the hash value of $Filter_i$ and ensures that they belong to the filter list defined on the attester side. In case this step successes, the attester has the assurance that target processes on attestee side are proved without containing any untrusted code or data, and the attester can proceed to next steps. On the other side, if this step fails, the attester sends a proper attestation result denoting this situation.

Authenticating Reporting Process To prove that the received measurements and updated policy rules are from the attestee, the attester authenticates them by verifying that all the measurements, updates and integrity measurement agent processes in the attestee are integrity protected. That is, the $RProcess_i$ does not contain any untrusted codes or data and its measurements correspond to PCRs in the attester. Also, there is no integrity violated information flow to these processes from subjects of TSL_i, according to the domain isolation rules. Note that these components can also be updated, but after any update of these components, the system should be fully re-measured and attested from boot time as aforementioned, i.e., to re-build a trusted initial system state T_0.

Policy Analysis by Attester DR@FT analyzes policy using a graph-based analysis method [14, 66, 67]. In this method, a policy file is first visualized into a graph, then this policy graph is analyzed against pre-defined security model such as our domain-based isolation, and a policy violation graph is generated. The main goal of this approach is to give semantic information of attestation result to the attestee, such that its system or security administrator can quickly and intuitively obtain any violated integrity configuration.

Note that verifying all the security policy rules in each attestation request decrease the efficiency, as loading policy graph and checking all the policy rules one by one cost a lot of time. Thus, we need to develop an efficient way for analyzing the attestee policy. In our method, the attester stores the policy of initial trusted system state T_0 or the latest trusted system state T_i, and its corresponding policy graph is loaded without any policy violation. Upon receiving the updated information from the attestee, the attester just needs to analyze these updates to see if there is new information flow violating integrity.

Algorithm 2 realizes this policy analysis method with a policy graph constructed from a complete policy file [14]. First, a function *policyParse* parses the updated policy information into subjects, objects, and their relationships with permission mapping.

Second, we load this information onto the previously generated *Policy*$_0$ graph with functions *changeNodes* and *changeLinks*, respectively. Then it determines if there is new information flow generated on this graph with a function *getnewFlow*. Third, we need to check if this new information flow violates our domain-based isolation rules using a function *identifyViolation*. Through this algorithm, rather than analyzing all the policy rules and all information flows for each attestation, we verify the new policy through only checking the updated policy rules and the newly identified information flow. The complexity of this policy analysis algorithm is *O(nn +nl +nt)*, where *nn* represents the number of changed subjects and objects, *nl* is the number of changed subjects and objects relationship in the policy update file; and *nt* represents the number of changed TCB in the TSL file.

Algorithm 2: Finding Integrity Violations with Policy Graph and Updates

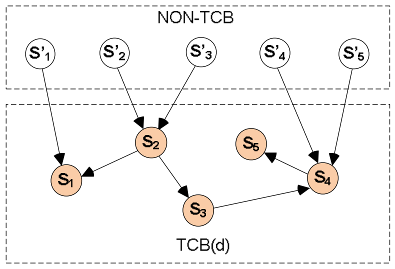
```
Input:  Initial trusted policy graph $G_0$, policy updating file $P_u$, TSL file $T_{listi}$, policy
        explanation file $F_e$, permission mapping File $F_p$, subject classification file $F_s$,
        object classification file $F_o$
Output: Policy violation graph $G_{flow}$
1  $Policy_u := policyParse(P_u,_{Fe}, F_p, F_s, F_o)$
2  $N_{list} := getUpdateSO(Policy_u)$
3  foreach $r_i \in R_{list}$ do
4      $changeNodes(n_a, G_0)$
5  $L_{list} := getUpdateLink(Policy_u)$
6  foreach $l_a \in L_{list}$ do
7      $changeLinks(l_a, G_0)$
8      $getnewFlow(l_a, G_0)$
9  foreach $t_a \in T_{list}$ do
10     $changeTCB(t_a, G_0)$
11     $G_{flow} := identifyViolation(t_a, G_0)$
```

Attestation Result Sending to Attester In case the attestation is successful, a new trusted system state is developed and the corresponding information is stored at the attester side for subsequent attestations. On the other hand, if the attestation fails, there are several possible attestation results including *CD*$_i$ *Integrity Fail*, *CD*$_i$ *Integrity Success,*

$RProcess_i$ Unauthenticated, and $Policy_i$ Fail/Success, and *CD_i Integrity Success, $RProcess_i$ Authenticated, and $Policy_i$ Fail /Success*. To assist the attestee reconfiguration, the attester also sends a representation of the policy violation graph to the attestee. Moreover, with this policy violation graph, the attester specifies the violation ranking and the trustworthiness of the attestee, which is explained in next section.

6.3 Integrity Violation Analysis

As we discussed above, existing attestation solutions such as TCG and IMA lack the expressiveness of the attestation result. In addition to their boolean-based response for attestation result, DR@FT adopts a graph-based policy analysis mechanism, where a policy violation graph can be constructed for identifying all policy violations on the attestee side. We further introduce a risk model built on a ranking scheme, which gives the implication of how severe the discovered policy violations are, and how to efficiently resolve them.



(a) A policy violation graph

Subject	SubjectRank
S_1	0.2799
S_2	0.3999
S_3	0.0799
S_4	0.4319
S_5	0.3456

(b) SubjectRank in TCB(d)

Path	PathRank
$<S'_1, S_1>$	0.2799
$<S'_2, S_2>$	0.8104
$<S'_3, S_2>$	0.8104
$<S'_4, S_4>$	0.6048
$<S'_5, S_4>$	0.6048

(c) PathRank between NON-TCB and TCB(d)

Figure 6.2: Example policy violation graph and rank

6.3.1 Policy Violation Graph

Information flow represents how data flows among system subjects, both trusted and untrusted. A sequence of information flow transitions between two subjects shows an information flow path. Applying our domain-based isolation approach, policy violations can be detected to identify information flows from low integrity subjects to high integrity subjects. Corresponding information flow paths representing such policy violations are named violated information flow paths (simply *violation paths*).

Obviously, we can find out two kinds of violation paths, *direct violation paths* and *indirect violation paths*. A direct violation path is a one-hop path through which an information flow can go from a low integrity subject to a high integrity subject. We observe that information flows are transitive in general. Therefore, there may exist information flows from a low integrity subject to a high integrity subject via several other subjects. This multi-hop path is called indirect violation path. All direct and indirect violation paths belonging to a domain can construct a policy violation graph for this domain.

Definition 7 *:* A policy violation graph for a domain d is a directed graph $G^v = (V^v, E^v)$ where

- $V^v \subseteq V^v_{NTCB} \cup V^v_{TCBd} \cup V^v_{TCB}$ where V^v_{NTCB}, V^v_{TCBd} and V^v_{TCB} are subject vertices containing in direct or indirect violation paths of domain d and belong to NON-TCB, TCB_d, and TCB_s, respectively.

- $E^v \subseteq E^v_{Nd} \cup E^v_{dT} \cup E^v_{NT} \cup E^v_{NTCB} \cup E^v_{TCBd} \cup E^v_{TCB}$ where $E^v_{Nd} \subseteq (V^v_{NTCB} \times V^v_{TCBd})$, $E^v_{dT} \subseteq (V^v_{TCBd} \times V^v_{TCB})$, $E^v_{NT} \subseteq (V^v_{NTCB} \times V^v_{TCB})$, $E^v_{NTCB} \subseteq (V^v_{NTCB} \times V^v_{NTCB})$, $E^v_{TCBd} \subseteq (V^v_{TCBd} \times V^v_{TCBd})$, and $E^v_{TCB} \subseteq (V^v_{TCB} \times V^v_{TCB})$, and all edges in E^v are contained in direct or indirect violation paths of domain d.

Figure 6.2 (a) shows an example of policy violation graph which examines information flows between NON-TCB and TCB_d[1]. Five direct violation paths are identified in this graph: <S'_1, S_1>, <S'_2, S_2>, <S'_3, S_2>, <S'_4, S_4>, and <S'_5, S_4>, crossing all the boundaries between NON-TCB and TCB_d. Also, eight indirect violation paths exist. For example, <S'_2, S_5> is a four-hop violation path passing through other three TCB_d subjects S_2, S_3, and S_4.

6.3.2 Ranking Policy Violation Graph

In order to explore more features of policy violation graphs and facilitate efficient policy violation detection and resolution, we introduce a scheme for ranking policy violation

[1]Similarly, the information flows between NON-TCB and TCB_s, and between TCB_d and TCB_s can be examined accordingly.

graphs. There are two steps to rank a policy violation graph. First, TCB_d subjects in the policy violation graph are ranked based on dependency relationships among them. The rank of a TCB_d subject shows reachable probability of low integrity information flows from NON-TCB subjects to the TCB_d subject. In addition, direct violation paths in the policy violation graph are evaluated based on the ranks of TCB_d subjects to indicate severity of these paths which allow low integrity information to reach TCB_d subjects. The ranked policy violation graphs are valuable for a system administrator as they need to estimate the *risk level* of a system and provide a guide for choosing appropriate strategies for resolving policy violations efficiently.

6.3.2.1 Ranking Subjects in TCB_d

Our notation of *SubjectRank* (SR) in policy violation graphs is a criterion that indicates the likelihood of low integrity information flows coming to a TCB_d subject from NON-TCB subjects through direct or indirect violation paths. The ranking scheme we introduce in this section adopts a similar process of rank analysis applied in hyper-text link analysis system, such as Google's PageRank [22] that utilizes a link structure provided by hyper links between web pages to gauge their importance. Comparing with PageRank which focuses on analyzing a web graph where the entries are *any* web pages contained in the web graph, the entries of low integrity information flows to TCB_d subjects in a policy violation graph are only identified NON-TCB subjects.

Consider a policy violation graph with N NON-TCB subjects, and s_i is a TCB_d subject. Let $N(s_i)$ be the number of NON-TCB subjects from which low integrity information flows could come to s_i, $N^{'}(s_i)$ the number of NON-TCB subjects from which low integrity information flows could *directly* reach to s_i, $In(s_i)$ a set of TCB_d subjects pointing to s_i, and $Out(s_j)$ a set of TCB_d subjects pointed from s_j. The probability of low integrity information flows reaching a subject s_i is given by:

$$SR(s_i) = \frac{N(s_i)}{N}\left(\frac{N^{'}(s_i)}{N(s_i)} + \left(1 - \frac{N^{'}(s_i)}{N(s_i)}\right) \sum_{s_j \in In(s_i)} \frac{SR(s_j)}{|Out(s_j)|}\right) \tag{6.1}$$

SubjectRank can be interpreted as a *Markov Process*, where the states are TCB_d subjects, and the transitions are the links between TCB_d subjects which are all evenly probable. While a low integrity information flow attempts to reach a high integrity subject, it should select an entrance (a NON-TCB subject) which has the path(s) to this subject. Thus, the possibility of selecting correct entries to a target subject is $\frac{N(s_i)}{N}$. After selecting correct entries, there still exist two ways, through direct violation paths or indirect violation paths, to reach a target subject. Therefore, the probability of flow transition from a subject is divided into two parts: $\frac{N'(s_i)}{N(s_i)}$ for direct violation paths and $1-\frac{N'(s_i)}{N(s_i)}$ for indirect violation paths. The $1-\frac{N'(s_i)}{N(s_i)}$ mass is divided equally among the subject's successors s_j, and $\frac{SR(s_j)}{|Out(s_j)|}$ is the rank value derived from s_j.

Figure 6.2 (b) displays a result of applying Equation (1) to the policy violation graph showing in Figure 6.2 (a). Note that even though subject s_4 has two direct paths from NON-TCB subjects like subject s_2, the rank value of s_4 is higher than the rank value of s_2, because there is another indirect flow path to s_4 (via s_3).

6.3.2.2 Ranking Direct Violation Path

We further define *PathRank* (PR) as the rank of a direct violation path[2], which is a criterion reflecting the severity of the violation path through which low integrity information flows may come to TCB_d subjects. Direct violation paths are regarded as the entries of low integrity data to TCB_d in policy violation graph. Therefore, the ranks of direct violation paths give a guide for system administrator to adopt suitable defense countermeasures for solving identified violations.

To calculate *PathRank* accurately, three conditions are needed to be taken into account: (1) the number of TCB_d that low integrity flows can reach through this direct violation path; (2) SubjectRank of reached TCB_d subjects; and (3) the number of hops to reach a TCB_d subject via this direct violation path.

[2]It is possible that a system administrator may also want to evaluate indirect violation paths for violation resolution. In that case, our ranking scheme could be adopted to rank indirect violation paths as well.

Suppose $< s_i^{'}, s_j >$ is a direct violation path from a NON-TCB subject $s_i^{'}$ to a TCB_d subject s_j in a policy violation graph. Let $Reach(< s_i^{'}, s_j >)$ be a function returning a set of TCB_d subjects to which low integrity information flows may go through a direct violation path $< s_i^{'}, s_j >$, $SR(s_l)$ the rank of a TCB_d subject s_l, and $H_s(s_i^{'}, s_l)$ a function returning the hops of the shortest path from a NON-TCB subject $s_i^{'}$ to a TCB_d subject s_l. The following equation is utilized to compute a rank value of the direct violation path $< s_i^{'}, s_j >$.

$$PR(< s_i^{'}, s_j >) = \sum_{s_l \in Reach(< s_i^{'}, s_j >)} \frac{SR(s_l)}{H_s(s_i^{'}, s_l)} \tag{6.2}$$

Figure 6.2 (c) shows the result using the above-defined equation to calculate the *PathRank* of the example policy violation graph. For example, $< s_2^{'}, s_2 >$ has a higher rank than $< s_1^{'}, s_1 >$, because $< s_2^{'}, s_2 >$ may result in low integrity information flows to reach more or important TCB_d subjects than $< s_1^{'}, s_1 >$.

6.3.2.3 Usage of Ranked Policy Violation Graph

We achieve the following benefits with our ranked policy violation graph for security analysis.

- *Policy violation resolution*: A system administrator is able to adopt different countermeasures to resolve identified violations with respect to the different rank values of violation paths. For example, a system administrator may apply a strong defense countermeasure, such as adding more strict filters, even disabling an information flow, to resolve a highly ranked violation path. On the other hand, it would be better for a system administrator to first eliminate highly ranked violation paths, because higher ranked violation paths, in general, involve in more policy violations and threat more TCB_d subjects.

- *Security measurement*: The total rank of all violation paths provides a measurement of risk level of a system. Regarding attestation process, even though an attestation may be failed, such rank value could reflect the trustworthiness of attested system accurately.

- *Assistance in visual policy analysis*: An entire policy violation graph may be in a huge size and very complicated. Thus, it is hard for system administrators to understand and analyze the whole piece of a policy violation graph. Ranks aid in selectively showing certain important parts of a policy violation graph based on the priority. Therefore, a system administrator could only concentrate on portions of a policy violation graph containing highly ranked information flow paths, which make the analysis and resolution of policy violation more effective.

6.3.3 Evaluating Trustworthiness

Let P_d be a set of all direct violation paths in a policy violation graph. The entire rank, which can be considered as a risk level of the system, can be computed as follows:

$$RiskLevel = \sum_{<s_i^{'},s_j>\in P_d} PR(<s_i^{'},s_j>) \tag{6.3}$$

The calculated risk level could reflect the trustworthiness of an attestee. Generally, the lower risk level indicates the higher trustworthiness of a system. When an attestation is successful and there is no violation path being identified, the risk level of the attested system is *zero*, which means an attestee has the highest trustworthiness. On the other hand, when an attestation is failed, corresponding risk level of a target system is computed. A *selective service* could be achieved based on this fine-grained attestation result. That is, the number of services provided by a service provider to the target system may be decided with respect to the trust level of the target system. On the other hand, a system administrator could refer to this attestation result as the evaluation of his system as well as security guidelines since this quantity response would give her a proper justification to select security countermeasures for improving the trustworthiness of a system.

6.3.4 Attestee Reconfiguration

When the current state of an attestee does not satisfy integrity requirements, a system administrator needs to change system configuration based on the attestation result to enhance a system's integrity and improve its trust level. There are two ways to reconfigure a system:

- Changing CD_i and TSL_i: In case an attestation is failed because of loading malicious or unknown codes or data into the system, removing such root causes can change the system state to be attested successfully.
- Changing $Policy_i$: Based on the violated information flow paths and corresponding ranks, a system administrator needs to first eliminate highly ranked violation paths since they generally involve more violations and more TCB_d subjects. Through modifying policy rules, $Pflow_i$ is changed to satisfy our domain isolation requirement. In other words, an attestee can maintain integrity requirements by modifying system policies properly for a newly installed software which interacts with TSL subjects.

6.4 Implementation and Evaluation

We have implemented DR@FT to evaluate its effectiveness and performance. This section first describes our experimentation setup, followed by implementation details and effectiveness evaluation, and performance study.

6.4.1 Attestee Configuration

Our attestee platform is a Lenovo ThinkPad X61 with Intel Core 2Duo Processor L7500 1.6GHz, 2 GB RAM, and Atmel TPM, and Fedora Core 6 is installed. We enable SELinux with the default policy based on the current distribution of SELinux [42]. To measure the attestee system with TPM, we update the Linux kernel to 2.6.26.rc8 with the latest IMA implementation [1], where SELinux hooks and IMA functions are enabled.

```
....
10 033e4f559247b256b5a163568275d7901ebe3734  GBK.so
10 041789c808648eac2d7a20c5105f7996f55a6c56  GEORGIAN-PS.so
10 4d019ac9fd103d42f2f7f7d081becc3dd5beb806  IBM850.so
10 73b86a44681c8d778bb143bdd233a6477dc19884  IBM852.so
10 9348eeafbbe7fe8627330b543c3f88d028c7e16f  ECMA-CYRILLIC.so
....
10 38f30a0a967fcf2bfee1e3b2971de540115048c8  initrc
....
```

(a) Measurement example

```
....
10 033e4f559247b256b5a163568275d7901ebe3734  GBK.so
10 041789c808648eac2d7a20c5105f7996f55a6c56  GEORGIAN-PS.so
10 4d019ac9fd103d42f2f7f7d081becc3dd5beb806  IBM850.so
10 73b86a44681c8d778bb143bdd233a6477dc19884  IBM852.so
10 9348eeafbbe7fe8627330b543c3f88d028c7e16f  ECMA-CYRILLIC.so
....
10 d63d12ced978aca120bfe6ee7683e394c2ffaef0  initrc
....
```

(b) Measurement after installing rootkit

Figure 6.3: Example of the DR@FT measurement list

Having IMA enabled, we configure the measurement of the attestee information. Specifically, after the attestee system kernel is booted, we mount the `sysfs` file system and inspect the measurement list values in `ascii_runtime_measurements` and `ascii_bios_measur`
`ements`. Figure 6.3(a) shows a sample of the measurement list. The fields include the PCR which is extended with the hash, the hash of the file, and the file name.

6.4.2 Attestation Implementation

We start from a legitimate attestee and make measurements of the attestee system for the later verification. To invoke a new attestation request from the attester, the attestation reporting daemon runs in the attestee and monitors the attestee system. This daemon is composed of two main threads. One is to monitor and get the new system state measurements and the other is to monitor and obtain the policy updates of the attestee. In case the attestee system state is updated due to new software installation, changing policy, and so on. An appropriate thread of the daemon automatically obtains the new measurement values as discussed in 6.2. The daemon then securely transfers the attestation information to the attester based on the security mechanisms supported by the trusted authority.

After receiving the updated system information from the attestee, the measurement module of the attester checks the received measurements against the stored PCR to prove its integrity. To analyze the possible revised attestee policy, the policy analysis module is developed as a daemon, which is ported from a policy analysis engine. We extend the engine to identify violated information flows from the updated policy rules based on domain-based isolation rules. We also accommodate the algorithm presented in Section 6.2.2, as well as our rank scheme to evaluate the trustworthiness of the attestee.

6.4.3 Evaluation

To assess the proposed attestation framework, we attest our testbed platform with Apache web server installed. To configure the trusted subject list of the Apache domain, we first identify the TCB_s based on the reference monitor-based TCB identification, including the integrity measurement, monitoring agents, and daemon. For TCB_d of the Apache, we first identify the Apache information domain subjects and objects with keyword `http` in the policy. Subjects such as `httpd_t`, `httpd_ssh_t`, and objects such as `httpd_cache_t` are regarded as Apache domain subjects and objects. We then identify the subjects and objects that can flow to these subjects and objects. Within the identified information domain, we discover Apache TCB_d such as `httpd_t` and `httpd_suexec_t` through the domain-based isolation principles. In addition, for Apache domain, we identify the initial filters including `sshd_t, passwd_t, su_t`. After this, we install the unverified codes and data to evaluate the effectiveness of our attestation framework. The detail process can be seen in Chapter 3 for how to identify the information of apache and TCB(d) of Apache.

Installing Malicious Code We first install a Linux rootkit, which is designed to compromise critical system processes and leave Trojan horse into it. Here, we assign the rootkit with the domain `unconfined_t` that enables information flows to domain `initrc_t` labeling `initrc` process, which belongs to TCB_s of the attestee.

Following the framework proposed in Section 6.2, the attestee system is measured from the bootstrap with IMA. After getting the new measurement values, the reporting daemon sends these measurements to the attester. Note that there is no policy update in this experiment. After getting the measurements from the attestee, attester verifies them by trying to match the measured hash values. Figure 6.3 shows the initial measurements of the `initrc` (in a trusted initial system state) and the changed value because of the installed rootkit. The difference between these two measurements indicates the original `initrc` is altered, and the attester confirms that the attestee is not in a trusted state.

Table 6.1: Mplayer policy information

Mplayer Subjects	
staff_mencoder_t	staff_mplayer_t
user_mplayer_t	sysadm_mencoder_t
sysadm_mplayer_t	user_mencoder_t
Mplayer Objects	
mencoder_exec_t	mplayer_etc_t
mplayer_exec_t	staff_mplayer_client_xevent_t
staff_mplayer_xproperty_t	staff_mplayer_property_xevent_t
staff_mplayer_default_xevent_t	staff_mplayer_focus_xevent_t
staff_mplayer_home_t	staff_mplayer_input_xevent_t
staff_mplayer_tmpfs_t	staff_mplayer_manage_xevent_t
user_mplayer_default_xevent_t	user_mplayer_focus_xevent_t
user_mplayer_home_t	user_mplayer_input_xevent_t
user_mplayer_xproperty_t	user_mplayer_client_xevent_t
user_mplayer_manage_xevent_t	user_mplayer_property_xevent_t
sysadm_mplayer_default_xevent_t	sysadm_mplayer_focus_xevent_t
user_mplayer_tmpfs_t	sysadm_mplayer_client_xevent_t
sysadm_mplayer_home_t	sysadm_mplayer_input_xevent_t
sysadm_mplayer_manage_xevent_t	sysadm_mplayer_property_xevent_t
sysadm_mplayer_xproperty_t	sysadm_mplayer_tmpfs_t

Installing Vulnerable Software In this experimentation, we install a vulnerable software called Mplayer on the attestee side. Mplayer is a media player and encoder software which is susceptible to several integer overflows in the real video stream dumuxing code. These flaws allow an attacker to cause a denial of service or potentially execution of the arbitrary code by supplying a deliberately crafted video file. After a Mplayer is installed, a Mplayer policy module is also loaded into the attestee policy. In this policy module, there are several different subjects such as `staff_mplayer_t`, `sysadm_mplayer_t`, and so on. Also, some objects are defined in security policies such as `user_mplayer_home_t` and `staff_mplayer_home_t`. A complete list of these subjects and objects is shown in Table 6.1.

After the Mplayer is installed, the attestation daemon finds that the new measurement of Mplayer is generated and the security policy of the system is changed. As the Mplayer does not belong to TCB_s and Apache TCB_d, the attestation daemon does not need to send the measurements to the attester. Consequently, the daemon only computes the security policy updates and sends the information to the attester.

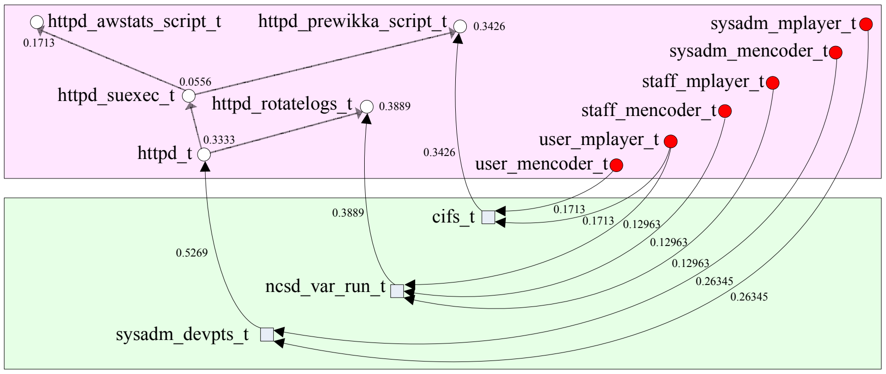


Figure 6.4: Information flow verification of Mplayer

Table 6.2: Attestation performance analysis (in seconds)

Policy Change	**Dynamic**				**Static**		
Size	$T_{Pupdate}$	T_{send}	$T_{Panalysis}$	**Overhead**	T_{Psend}	$T_{Panalysis}$	**Overhead**
0	0.23	0	0	0.23	14.76	90.13	104.89
-0.002MB (Reduction)	0.12	0.002	0.02	0.14	14.76	90.11	104.87
-0.019MB (Reduction)	0.08	0.01	0.03	0.12	14.74	89.97	104.34
-0.024MB (Reduction)	0.04	0.02	0.03	0.09	14.74	89.89	104.23
0.012MB (Addition)	0.37	0.01	0.03	0.41	14.77	90.19	104.96
0.026MB (Addition)	0.58	0.02	0.03	0.63	14.78	90.33	105.11
0.038MB (Addition)	0.67	0.03	0.04	0.74	14.79	90.46	105.25

Upon receiving the updated policies, we analyze these updates and obtain a policy violation graph as shown in Figure 6.4, where the filled circle nodes representing subjects and objects related to Mplayer; unfilled circle nodes representing subjects and objects related to Apache; and unfilled rectangle nodes representing other objects. The links show the information flow from Mplayer to Apache. The rank values on the paths indicate the severity of the corresponding violation paths.Through objects such as `cifs_t`, `sysadm_devtps_t`, `ncsd_var_run_t`, information flows from Mplayer can reach Apache domain. In addition, rank values are calculated and shown in the policy violation graph, which guides effective violation resolutions. For example, there are three higher ranked paths including path from `sysadm_devpts_t` to `httpd_t`, from `ncsd_var_run_t` to `httpd_rotatelogs_t`, and from `cifs_t` to `httpd_prewikka_script_t`. Meanwhile, a risk level value (1.2584) reflecting the trustworthiness of the attestee system is computed based on the ranked policy violation graph.

Once receiving the attestation result shown in Figure 6.4, the attestee administrator solves the violation that has the higher rank than others. Thus, the administrator can first resolve the violation related to `httpd_t` through introducing `httpd_sysadm_devpts_t`.

```
allow httpd_t httpd_sysadm_devtps_t:chr_file {ioctl read write
getattr lock append};
```

After the policy violation resolution, the risk level of the attestee system is lowered to *0.7315* . Continuously, after the attestee resolves all the identified policy violations and the risk level is decreased to be *zero*, the attestation daemon gets a new policy update file and sends it to the attester. Upon receiving this file, the attester verifies whether these information flows violate domain-based isolation integrity rules since these flows are within the NON-TCB–even though there are new information flow compared to the stored $Policy_0$. Thus, an attestation result is generated which specifies the risk level (in this case, *zero*) of the current attestee system. Consequently, a new trusted system state is built for the attestee. In addition, the information of this new trusted system state is stored in the attester side for the later attestation.

6.4.4 Performance

To examine the scalability and efficiency of DR@FT, we investigate how well the attestee measurement agent, attestation daemon, and the attester policy analysis module scale along with the increased complexity, and how efficiently DR@FT performs by comparing it with the traditional approaches.

In DR@FT, the important situations influencing the attestation performance include system updates and policy changes. Hence, we evaluate the performance of DR@FT by changing codes and data to be measured and modifying the security policies. Based on our study, we observe that normal policy increased or decreased no more than 40KB when installing or uninstalling software. Also, a system administrator does not make the enormous changes over the policy. Therefore the performance is measured with the range from zero to around 40KB in terms of policy size.

Performance on the attestee side Based on DR@FT, the attestee has three main factors influencing the attestation performance. (1) Time spent for the measurement: Based on our experimentation, the measurement time increases roughly linearly with the size of the target files. For example, measuring policy files with 17.2MB and 20.3MB requires 14.63 seconds and 18.26 seconds, respectively. Measuring codes around 27MB requires 25.3sec. (2) Time spent for identifying policy updates $T_{Pupdate}$: Based on the specification in Section 6.2, policy updates are required to be identified and sent to the attester. As shown in Table 6.2, for a system policy which is the size of 17.2MB at its precedent state, the increase of the policy size requires more time for updating the policy and vice versa. (3) Time spent for sending policy updates T_{Psend}: Basically, the more policy updates, the higher overhead was observed.

Performance on the attester side In DR@FT, the measurement verification is relatively straightforward. At the attester side the time spent for policy analysis $T_{Panalysis}$ mainly influences its performance. As shown in Table 6.2, the analysis time roughly increases when the policy change rate increases.

Comparison of dynamic and static attestation To further specify the efficiency of DR@FT, we compare the overhead of DR@FT with a static attestation. In the static approach, the attestee sends all system state information to an attester, and the attester verifies the entire information step by step. As shown in Table 6.2, the time spent for static attestation is composed of T_{Psend} and $T_{Panalysis}$, which represent the time for sending policy module and analyzing them, respectively. Obviously, the dynamic approach can dramatically reduce the overhead compared to the static approach. It shows that DR@FT is an efficient way when policies on an attestee are updated frequently.

6.5 Conclusion

Through extending visualization-based policy analysis and management approach, w have presented a dynamic remote attestation framework called DR@FT for efficiently verifying if a system satisfies integrity protection property and indicates integrity violations which determine its trustworthiness level. The integrity property of our work is based on an DIM, which is utilized to describe the integrity requirements and identify integrity violations of a system. To achieve the efficiency and effectiveness of remote attestation, DR@FT focuses on system changes on the attestee side. We have extended a powerful policy analysis engine to represent integrity violations with the rank scheme. In addition, our results showed that our dynamic approach can dramatically reduce the overhead compared to static approach. We believe such an intuitive evaluation method would help system administrators reconfigure the system with more efficient and strategic manner. DR@FT provides a fine-grained remote attestation not only for TCB_s but also for user-space domains. For our future work, we would further investigate how our attestation framework can be applied to open network environments dealing with a trusted and delegated attestation service for the attester. We also plan to optimize DR@FT and release it to open source community.

CHAPTER 7: CONCLUDING REMARKS

7.1 Summary

To analyze and manage the security policy, a fundamental question is to understand the security requirements of the security policy in an open and distributed environment. Instead of focusing on analyzing the security requirements for a specific policy, in general, a trusted computing base is required to be protected in every system. From this aspect, we need to analyze the requirements for protecting the trusted computing base followed by the requirements of protecting other system components. Another important issue is how to provide a usable security policy analysis and management approach to the stakeholders. The usability includes how to present the security policy in an understandable way, how to provide an easy and understandable policy analysis method, and how to provide an efficient way for identifying policy violations such as how to conveniently identify the root causes for the policy violations.

To tackle the above mentioned issues, we proposed a domain-based isolation model to specify how to protect the system trusted computing base and how to protect other system applications by realizing application isolations. Based on this model, several rules were proposed to identify the policy violations. This model was first proved based on a CPN simulation tool by analyzing the SELinux example policies. Also, we proposed the semantic substrates-based and adjacency matrix-based visualization methods to visualize the security policies for addressing usability issues. To provide an easy way for the policy analysis, we developed a graphical query mechanisms for querying the security policy and identifying policy violations. Still using semantic substrates-based and adjacency matrix-based method to represent the policy violations, the critical points and violation path are marked out with different color, shape and route to specify the origin of the policy violations. In addition, we implemented a tool called PVM based on our visualization-based policy analysis and management framework. To measure the effectiveness and efficiency of our tool, we also successfully performed the user study that produced the favorable results. Finally, we extended our policy analysis and management framework to support remote attestation for proving the trustworthiness of the system in an open and distributed environment.

In addition to the above contributions, we strongly believe this work has several potential impacts to the information security community. As an information flow-based security model, DIM has significantly demonstrated the applicability of information flow analysis in open and distributed environments. The method of using CPN also provides the possibility of using such a system modeling tool for proving other security models such as RBAC. Also, our information visualization methods bring the first-hand practical experience for analyzing different security policies–such as policies for firewall policies and other security appliance–with visual analytics. By sharing our experience with the information security community, we hope to further improve a method for evaluating the usability of technologies in information security.

7.2 Future Work

This ground-breaking work also has several future research directions:

7.2.1 Issues in Security Requirements

In our proposed approach, a domain-based isolation method was proposed, in which the system TCB and TCB(d) are identified through specific strategies. This layouts high-level requirements of the security policy. However, in case that the security policy is not that well organized, it is very hard to identify such items to be protected. In addition, the filter is normally based on a certain information flow. One process can work as a filter in one situation but cannot work as a filter in the other situation. Hence, the filter identification should be based on the policy violations as we proposed in our work. This would be a big challenge for the security policy architect. In the future, we would further investigate the security model to handle this dilemma.

7.2.2 Security Policy Analysis and Management for Mobile Device

A possible application domain of our work would be security policy analysis and management on mobile devices. However, due to the different security models such as delegation models are applied into the mobile device policy, a more comprehensive security policy analysis model is needed.

In addition, the performance of policy analysis is a big challenge since mobile devices require a more efficient policy analysis tool. Also, for mobile devices, an owner would be the policy architect, thus the policy specification and policy violation should be convenient and easy for them to figure out. Due to these specific requirements, it is tremendously necessary to extend and improve our proposed policy visualization methods and a corresponding tool in both design and performance aspects. Furthermore, our dynamic attestation work still has many unsolved issues to accommodate those issues on mobile devices [48, 29, 72].

7.2.3 Security Policy Management

We will also continuously discover novel ways in effectively generating and managing security policy for computing systems including operating systems and mobile devices as well as new emerging infrastructures such as smart grid [45] and cloud computing [38, 71]. Especially identifying the security requirements related to access control for these new infrastructures and developing new security policy framework would advance the field of information technology security and help establish more trustworthy computing environments for organizational users to safeguard sensitive and critical information.

BIBLIOGRAPHY

[1] LIM Patch. http://lkml.org/lkml/2008/6/27.

[2] Lime Survey Tool. http://www.limesurvey.org/.

[3] NSA Security-Enhanced Linux Example Policy. http://www.nsa.gov/selinux/.

[4] Paired Samples T-tests. http://www.statisticssolutions.com/methods-chapter/statistical-tests/paired-sa

[5] Security Policy. http://en.wikipedia.org/wiki/Securitypolicy.

[6] SELinux Example Policy. http://www.nsa.gov/research/selinux/code/download-stable.shtml.

[7] SELinux Policy Editor. http://www.selinux.hitachi-sk.co.jp/en/tool/selpe/selpe-top.html.

[8] Standard ML. http://www.smlnj.org/sml.html.

[9] STOOLS. http://oss.tresys.com/projects/setools/wiki/download.

[10] Tresys Technology Apol. http://www.tresys.com/selinux/.

[11] Trusted Computing Group. https://www.trustedcomputinggroup.org/home.

[12] *Trusted Computer System Evaluation Criteria*. United States Government Department of Defense (DOD), Profile Books, 1985.

[13] System management concepts: Operating system and devices, 1999. 1 edition.

[14] Gail-Joon Ahn, Wenjuan Xu, and Xinwen Zhang. Systematic policy analysis for high-assurance services in selinux. In *POLICY '08: Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, pages 3–10, Washington, DC, USA, 2008. IEEE Computer Society.

[15] Masoom Alam, Xinwen Zhang, Mohammad Nauman, Tamleek Ali, and Jean-Pierre Seifert. Model-based behavioral attestation. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 175–184, New York, NY, USA, 2008. ACM.

[16] Peter A.Loscocco and Stephen D.Smalley. Meeting critical security objectives with security-enhanced linux. In *In Proceedings of the Ottawa Linux Symposium*, 2001.

[17] AMJ. P. Anderson. Computer security technology planning study. *Technical Report ESD-TR-73-51*, II, 1972.

[18] Aleks Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733–740, 2006. Senior Member-Ben Shneiderman.

[19] Human Computer Interaction Lab at University of Maryland. Piccolo. *Available from http://www.cs.umd.edu/hcil/jazz/download/index.shtml*.

[20] K. J. Biba. Integrity consideration for secure compuer system. Technical report, Mitre Corp. Report TR-3153, Bedford, Mass., 1977.

[21] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the Eighth National Computer Security Conference*, Gaithersburg, Maryland, 1985.

[22] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[23] Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A protocol for property-based attestation. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 7–16, New York, NY, USA, 2006. ACM.

[24] Shuo Chen, David Ross, and Yi-Min Wang. An analysis of browser domain-isolation bugs and a light-weight transparent defense mechanism. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 2–11, New York, NY, USA, 2007. ACM.

[25] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. *Proceedings of the IEEE symposium on security and privacy*, 1987.

[26] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5), May 1976.

[27] DOD. *Department of Defense trusted computer system evaluation criteria.* in the glossary under entry Trusted Computing Base (TCB), 1985.

[28] R.F. Erbacher. Intrusion behavior detection through visualization. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 2507–2513, Oct 2003.

[29] Andy Green. Management of security policies for mobile devices. In *InfoSecCD '07: Proceedings of the 4th annual conference on Information security curriculum development*, pages 1–4, New York, NY, USA, 2007. ACM.

[30] Marc Green. Toward a perceptual science of multidimensional data visualization: Bertin and beyond. *Available from http://www.ergogero.com/dataviz/dviz2.html*, 1998.

[31] J. Guttman, A. Herzog, and J. Ramsdell. Information flow in operating systems: Eager formal methods. In *Workshop on Issues in the Theory of Security (WITS)*, 2003.

[32] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation: a virtual machine directed approach to trusted computing. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*, pages 3–3, Berkeley, CA, USA, 2004. USENIX Association.

[33] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, and R. Lutz. Software fault tree and colored petri net based specification, 2000.

[34] I. Herman, G. Melancon, and M.S. Marshall. Graph visualization and navigation in information visualization: Asurvey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[35] Trent Jaeger, Reiner Sailer, and Umesh Shankar. Prima: policy-reduced integrity measurement architecture. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 19–28, New York, NY, USA, 2006. ACM.

[36] Trent Jaeger, Reiner Sailer, and Xiaolan Zhang. Analyzing integrity protection in the selinux example policy. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.

[37] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. 1997. Three Volumes.

[38] Lori M. Kaufman. Data security in the world of cloud computing. *IEEE Security and Privacy*, 7:61–64, 2009.

[39] René Keller, Claudia M. Eckert, and P. John Clarkson. Matrices or node-link diagrams: which visual representation is better for visualising connectivity models? *Information Visualization*, 5(1):62–76, 2006.

[40] C.P. Lee, J. Trost, N. Gibbs Beyah Raheem, and J.A. Copeland. Visual firewall: Real-time network security monitor. In *IEEE Workshops Visualization for Computer Security*, pages 129–136, 2005.

[41] Ninghui Li, Ziqing Mao, and Hong Chen. Usable mandatory integrity protection for operating systems. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 164–178, 2007.

[42] Peter Loscocco and Stephen Smalley. Integrating flexible support for security policies into the linux operating system. In *USENIX Annual Technical Conference, FREENIX Track*, pages 29–42, 2001.

[43] S. Mathew, R. Giomundo, S. Upadhyaya, M. Sudit, and A. Stotz. Understanding multistage attacks by attack-track based visualization of heterogeneous event streams. In *VizSEC '06: Proceedings of the 3rd international workshop on Visualization for computer security*, pages 1–6, New York, NY, USA, 2006. ACM.

[44] Frank Mayer, Karl MacMillan, and David Caplan. *SELinux by Example: Using Security Enhanced Linux (Prentice Hall Open Source Software Development Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[45] Patrick McDaniel and Stephen McLaughlin. Security and privacy challenges in the smart grid. *IEEE Security and Privacy*, 7:75–77, 2009.

[46] Sharma Nidhi. Fireviz: A personal firewall visualizing tool. In *Thesis (M. Eng.), Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science*, 2005.

[47] XML Organization. The xml c parser and toolkit of gnome. http://xmlsoft.org/.

[48] Anand Patwardhan, Vlad Korolev, Lalana Kagal, and Anupam Joshi. Enforcing policies in pervasive environments. *Mobile and Ubiquitous Systems, Annual International Conference on*, 0:299–308, 2004.

[49] Niels Provos, Markus Friedl, and Peter Honeyman. Preventing privilege escalation. *12th USENIX Security Symposium*, page 11, August 2003.

[50] H. Reiterer and G. Muler. A visual information seeking system for web search. In *Proceedings of the Oberquelle H, Oppermann R, Krause J (eds) Mensch & Computer conference*, pages 297–306, March 2001.

[51] H. Reiterer, G. Tullius, and T. Mann. Insyder: a content-based visual-informationseeking system for the web. *Springer-Verlag GmbH , International Journal on Digital Libraries*, 2005.

[52] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *USENIX Security Symposium*, pages 223–238, 2004.

[53] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[54] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.

[55] Beata Sarna-Starosta and Scott D. Stoller. Policy analysis for security-enhanced linux. In *Proceedings of the 2004 Workshop on Issues in the Theory of Security (WITS)*, pages 1–12, April 2004.

[56] Umesh Shankar, Trent Jaeger, and Reiner Sailer. Toward automated information-flow integrity verification for security-critical applications. In *NDSS*. The Internet Society, 2006.

[57] Umesh Shankar, Trent Jaeger, and Reiner Sailer. Toward automated information-flow integrity verification for security-critical applications. In *NDSS*. The Internet Society, 2006.

[58] Z. Shen and K. Ma. Path visualization for adjacency matrices. In *In Proceedings of Eurographics/IEEE Symposium on Visualization (EuroVis)*, May 2007.

[59] Stephen Smalley. Configuring the selinux policy. http://www.nsa.gov/SELinux/docs.html, 2003.

[60] Ravi S.Sandhu, EdwardJ.Coyne, Hal L.Feinstein, and Charles E.Youman. Role-based access control models. *IEEE Computer*, 29(2), February 1996.

[61] A. G. Sutcliffe, M. Ennis, and S. J. Watkinson. Empirical studies of end-user information searching. *Journal of the American Society for Information Science*, 51(13):1211–1231, November 2000.

[62] T.Fraser. Lomac: Low water-mark integrity protection for cots environment. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2000.

[63] Ramona Su Thompson, Esa M. Rantanen, William Yurcik, and Brian P. Bailey. Command line or pretty lines?: comparing textual and visual interfaces for intrusion detection. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, page 1205, New York, NY, USA, 2007. ACM.

[64] Tung Tran, Ehab S. Al-Shaer, and Raouf Boutaba. Policyvis: Firewall security policy visualization and inspection. In *LISA*, pages 1–16, 2007.

[65] WIKIPEDIA. Trusted computing base. Available at http://en.wikipedia.org/wiki/Tusted_Computing_Base.

[66] Wenjuan Xu, Mohamed Shehab, and Gail-Joon Ahn. Visualization based policy analysis: case study in selinux. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 165–174, New York, NY, USA, 2008. ACM.

[67] Wenjuan Xu, Xinwen Zhang, and Gail-Joon Ahn. Towards system integrity protection with graph-based policy analysis. In *DBSec*, pages 65–80, 2009.

[68] Danfeng Yao, Michael Shin, Roberto Tamassia, and William H. Winsborough. Visualization of automated trust negotiation. In *VizSEC 05: IEEE Workshop on Visualization for Computer Security*, Oct 2005.

[69] Xiaoxin Yin, William Yurcik, Michael Treaster, Yifan Li, and Kiran Lakkaraju. Visflowconnect: netflow visualizations of link relationships for security situational awareness. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 26–34, New York, NY, USA, 2004. ACM.

[70] William Yurcik. Visualizing netflows for security at line speed: the sift tool suite. In *LISA'05: Proceedings of the 19th conference on Large Installation System Administration Conference*, pages 16–16, Berkeley, CA, USA, 2005. USENIX Association.

[71] Xinwen Zhang, Joshua Schiffman, Simon Gibbs, Anugeetha Kunjithapatham, and Sangoh Jeong. Securing elastic applications on mobile devices for cloud computing. In *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 127–134, New York, NY, USA, 2009. ACM.

[72] Imran A. Zualkernan, Sina Nikkhah, and Mohammad Al-Sabah. A lightweight distributed implementation of ims ld on google's android platform. In *ICALT '09: Proceedings of the 2009 Ninth IEEE International Conference on Advanced Learning Technologies*, pages 59–63, Washington, DC, USA, 2009. IEEE Computer Society.