Systematic Policy Analysis for High-assurance Services in SELinux

Gail-Joon Ahn † and Wenjuan Xu †
University of North Carolina at Charlotte
{gahn,wxu2}@uncc.edu

Xinwen Zhang
Samsung Information Systems America
xinwen.z@samsung.com

Abstract

Identifying and protecting the trusted computing base (TCB) of a system is an important task to provide high-assurance services since a set of trusted subjects should be legitimately articulated for target applications. In this paper, we present a formal policy analysis framework to identify TCB with the consideration of specific security goals. We also attempt to model information flows between domains in SELinux policies and detect security violations among information flows using Colored Petri Nets.

1 Introduction

Determining whether a system can be trusted or not is a critical problem in systems and network management. Particularly for security reasons, 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, in order to deploy applications in distributed and collaborative computing environments, one machine may need to check if another machine currently runs a known good version of an application software on a well-configured, trusted operating system. Otherwise, a remote machine may run buggy or malicious application codes, or may be improperly configured such that the trusted application can be corrupted by untrustful programs or users. Supporting such an important assurance feature with comprehensive security analysis is necessary to trust a target system based on its current security configurations and policies.

Information flow control is the foundation of many security requirements such as integrity and confidentiality. The earliest information flow analysis works adopted a lattice model to illustrate the flow relationship between objects [9]. Those works assume that every object is labelled with a security attribute, and check information flow by examining the security labels of the objects in a flow path. Particularly, the lattice model requires that information cannot flow from low integrity objects to high integrity objects. In practical systems, however, under various circumstances, information flow is allowed from low integrity subjects to high integrity subjects. Clark-Wilson model [8] attempts to capture this notion. It states that information can flow from low integrity objects to high integrity objects only through certain programs so-called transaction procedures (TP). Jaeger et al. adopted this approach and proposed a CW-lite model, where filters are deployed in all application interfaces handling information flow from low integrity data to high integrity applications [11].

There are several related approaches and tools supporting security policy analysis based on information flow control [1, 10, 11, 12, 17]. However, those tools and approaches cannot analyze the integrity protection of *information domain*, simply called *domain*, between services and applications. In other words, most of previous works focused on the identification of a common and minimum trusted computing base (TCB) which includes trusted subjects for an entire system. Also, they cannot analyze security of high level application domains, where TCB may have various dependable relationships with different trusted subjects. In this paper, we consider a domain as a collection of subjects and objects which jointly function for an application or system service. Hence, the trust of these application and service domains cannot be judged unless information flow between them are appropriately evaluated.

Towards high-assurance applications and services based on a minimum TCB, we propose a framework to identify policy violations caused by interactions between applications and system services. We first seek a general policy analysis method to identify the TCB of individual domains. We then build a tool to automate the analysis task and visualize policy configurations and security violations. Finally, we present general principles to resolve violations based on a domain-based Clark-Wilson security model. This paper is organized as follows. Section 2 describes some related works followed by our methodology for policy analysis in Section 3. Section 4 describes how our methodology can

†The work of Gail-Joon Ahn and Wenjuan Xu was partially supported by the grants from National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).

978-0-7695-3133-5/08 $25.00 © 2008 IEEE
DOI 10.1109/POLICY.2008.18

be applied to analyze SELinux policies and presents some identified security violations and solutions. Section 5 concludes this paper and presents our future work.

2 Background and Related Work

2.1 SELinux Policy Related Work

As most existing works on policy analysis focus on SELinux policy, we first give an overview here. SELinux [18] implements MAC-based mechanism in Linux kernel, where the policy is built on the Type-Enforcement model [6]. In SELinux, domains (e.g. `passwd_t`) are used to label processes (e.g. password management process) and types (e.g. `security_t`) are used to label files and other resources (e.g. files in directory `/security`). An SELinux policy includes a set of policy rules indicating how domain (or subject) types can access object types (e.g. `allow passwd_t security_t:dir{read search getattr};`). An access operation is specified by two pieces of information: a class (e.g. file, directory) and a permission (e.g. read, write). Although SELinux adopts RBAC to help policy organization, existing SELinux policy analysis works mainly focus on the Type-Enforcement model.

Previous methods and tools developed to analyze SELinux policies include Gokyo [11, 12], SLAT (Security Enhanced Linux Analysis Tool) [10], PAL (Policy Analysis using Logic Programming) [17], APOL [1] and SEAnalyzer [7]. Gokyo is used to check the integrity protection of the TCB in SELinux. The integrity of the TCB holds if there is no subject identified by a security type in SELinux –that can be written by a type outside the TCB and read by a type inside the TCB, except for special cases in which a designated trusted program sanitizes untrusted data when it enters the TCB. Gokyo mainly identifies a common TCB in SELinux but a typical system may have multiple security goals with obviously different kinds of trust relationships. Hence, Gokyo cannot cover all the aspects of policy violations. SLAT [10] defines an information flow model to analyze SELinux policies. In SLAT, the model defines information flow relations as flow transitions based on the write and read operations. Through a set of flow transition relationships, a path is defined to reflect a sequence of events. Sarna-Sota et al. [17] use the SLAT information flow model to implement a framework for analyzing SELinux policies, which is called PAL. PAL creates a logic program and is capable of executing queries to analyze policies. PAL is implemented using the XSB logic-programming system [3]. APOL [1] is a tool developed by Tresys Technology to analyze SELinux configuration policies. Its main features include forward and reverse domain type transition analysis, direct and transitive information flow analysis, relabel analysis, and type relationship analysis. SEAnalyzer [7] is a tool based on Colored Petri Net (CPN) to analyze SELinux policies. Policy violations can be identified through creating queries in CPN. Queries in SEAnalyzer are built on the similar information flow model with SLAT, which is mainly about TCB protection.

Although the above-mentioned tools and languages provide different ways for analyzing SELinux policies, we still need a systematic method to analyze policies effectively and seamlessly. Specifically, SLAT, PAL and APOL require an administrator to be well versed in SELinux policies to generate meaningful queries and ultimately extract meaningful information. Gokyo does not address the issue of the application/system service level integrity. SEAnalyzer tries to use CPN to aid analysis processes. However, it expresses generated policy violations in CPN token text expressions. Where the number of identified policy violations is large, especially during the initial states of a analysis process, it is still difficult for a policy administrator to understand and resolve identified violations. Our work demonstrates a new and systematic way to use CPN for automating and visualizing policy analysis.

Table 1. TCB identified.

ipsec_mgmt_t	setfiles_t	sysadm_t	initrc_t	getty_t
load_policy_t	hwclock_t	syslogd_t	mount_t	apt_t
admin_passwd_exec_t	automount_t	ldconfig_t	kernel_t	klogd_t
checkpolicy_t	cardmgr_t	fsadm_t	snmpd_t	init_t
bootloader_t	logrotate_t	newrole_t	quota_t	dpkg_t
local_login_t	sshd_login_t	useradd_t	passwd_t	sshd_t

2.2 Trusted Computing Base

The concept of TCB partitions hardware and software of a system into two parts: the part inside the TCB and the part outside the TCB which are referred to as trusted (TCB) and untrusted (NON-TCB) respectively. Therefore, the identification of TCB is obviously a basic problem in security policy design and management. Jeager et al. [11] attempt to identify TCB in SELinux with the following steps: (i) *Domain type transition-based TCB Initial Identification* to identify the initial minimum TCB based on the domain type transition relationships. The subject types such as `boot_t` and `kernel_t` can be identified as an initial TCB; (ii) *Flow transition–based TCB Identification* to identify other subjects that have information flow to all of the subjects in the initial TCB and to append those identified subjects to the initial TCB; and (iii) *Identified TCB Adjustment* to detect policy violations from NON-TCB to TCB, manually adjusting TCB based on the analysis results as shown in Table 1.

In this paper, we propose *Reference Monitor based Initial TCB Identification* approach for general policy analysis. Based on the definition of TCB, we identify the initial TCB through the identification of subjects functioning as the ref-

erence monitor. Applying our main idea to SELinux, we determine the types that are responsible for the reference monitor such as `checkpolicy_t` and `load_policy_t` as part of the TCB. To support these services, we first identify other types such as `kernel_t` in the initial TCB. After the reference monitor based TCB identification is complete, we apply the method of flow transition and TCB adjustment to help accomplish the TCB identification task. In our work, through automatic identification process, we successfully identify several additional types including `kudzu_t`, `lvm_t`, and `restorecon_t` in the TCB.

3 Our Methodology for Policy Analysis

In this section, we present a formal specification of policy graph and information flow transitions. We then discuss our strategies to identify the security properties of a policy including different TCBs and security violations. Finally, we discuss general principles to resolve policy violations.

3.1 Security Policy

Our security model is similar to many traditional approaches [14]. Specifically, a set of sensitive resources (e.g., files, directories, sockets, and processes) to be protected in a system are called *objects*, and the active entities performing actions (*rights or permissions*) on objects are called *subjects* (e.g., users and representative processes). A security *policy* is composed of a set of subjects, a set of objects, and the corresponding relationships indicating whether 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 $read\_like$ and operations between subjects can be expressed as $calls$ [10]. Figure 1 shows the information flow relations with these three types of operations between subjects and objects. Figure 1 (a) describes that if subject *x* can write to object *y*, then there is information flow from *x* to *y*, which is denoted as $write(x, y)$. Figure 1 (b) shows that if subject *x* can read object *y*, then there is information flow from *y* to *x*, which is denoted as $read(y, x)$. Figure 1 (c) describes that if subject *x* can call another subject *y*, then there is information flowing from *y* to *x*, which is denoted as $call(y, x)$. Based on these concepts, we arrive the definition of policy graph to express information flow relations between subjects, objects, and operations.

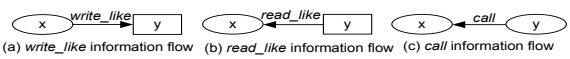

Figure 1. Types of information flow relations.

Definition 1 *A* Policy Graph *of a system is a directed graph* G=(V, E)*, where a set of vertices* V *represents all subjects and objects in the system, and a set of edges* E=$V \times V$ *represents all information flow relations between the 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 a subject* x, $y \in V_s$ *and an object* $o \in V_o$*:*
  - $(x, o) \in E_w$ *if and only if* $write(x, o)$.
  - $(o, y) \in E_r$ *if and only if* $read(o, y)$.
  - $(y, x) \in E_c$ *if and only if* $call(y, x)$.

3.2 Information Flow Transitions

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

Definition 2 *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 3 *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* $flowtrans(s_1, s_n)$*, or* $\exists s_i \in V$, $flowpath(s_1, s_i) \wedge flowpath(s_i, s_n)$.

3.3 TCB of a Domain

Protecting a system's TCB can satisfy a particular security goal such as protecting kernel integrity. However, some of other security goals such as separation of different processes or limiting the privileges of certain processes cannot be satisfied at the same time. To address this, we propose a new concept called the *domain TCB* to determine if a policy can satisfy such kind of security goals.

Definition 4 *Let* d *be an information domain functioning as certain application or system service through a set of related subjects and objects. The* $TCB(d)$ *is a set of subjects in* d *which has the same level of security sensitivity.*

Superficially, a $TCB(d)$ represents a set of subjects and objects that handle or contain all critical data processed for the functionality of domain d. Note that the concept of information domain has much wider scope than the domain type in SELinux in this paper. For example, a web server domain running in a system consists of many subjects–such

as processes, plugins, tools–and objects including data files, config files, and logs. We consider all of these subjects and objects as the TCB of this domain, while its network object such as *socket:80* is not considered as its TCB since it may accept low integrity data. Our objective is to identify the TCB of each domain and consider its integrity for assuring secure applications and system services. The following principles for the TCB(d) identification are proposed. For sake of clarification, the kernel-level minimum TCB we introduced in Section 2.2 is called system or global TCB.

- *Keyword-based TCB(d) identification*: To identify the TCB(d) subjects of certain application or system service domain, we first identify all of subjects related to the integrity of the target domain based on some keywords. For example, in SELinux we use the keyword `httpd` to identify its initial TCB(d) subjects for the TCB of a web server, and `mail` and `sendmail` to identify its initial TCB(d) subjects for the TCB of mail service domain.
- *Flow transition-based TCB(d) identification*: The subjects that can flow only to or from the initial identified TCB(d) are included in the domain TCB.
- *Policy violation-based TCB(d) adjustment:* Through identifying the policy violations, we adjust the TCB(d) with incorrectly included or excluded subjects.

3.4 Domain-based Integrity Model

The concepts of flow transitions and paths are used in our information flow-based policy analysis for domain TCB and policy violation identifications. For this purpose, we need to identify which flow transitions and paths can cause possible violations. Specifically, as integrity protection is the focus of this paper, we identify policy violations based on an integrity model. Traditional integrity models include Biba [5] and Clark-Wilson [16]. Biba integrity property is fulfilled if a high integrity process cannot read lower integrity data, execute lower integrity programs, or obtain lower-integrity data in any other manner. Clark-Wilson provides a different view of dependencies, where low integrity data can flow to high integrity only through particular information flow channels, TPs or filters in CW-lite [11]. A typical example of filter can be a firewall, and a valid flow transition is that a mail application receives mails by calling the firewall application to sanitize data before handling.

As communication and collaboration between applications and services are frequently required in most contemporary systems, one-way information flow with Biba would be sufficient for the most cases. However, filters between high integrity and low integrity are necessary for TCB and NON-TCB isolations, but may not be flexible for domain-based isolations. For example, in SELinux, processes of user applications and staff applications are required to be isolated, which both are beyond the minimum and global system TCB. Hence, our approach is a domain-based Clark-Wilson isolation model. Specifically,

Definition 5 *Clark-Wilson is satisfied for an information domain d only if for any information flow path in d, (i) all nodes along the path are in TCB(d), or (ii) it flows to TCB(d) from system TCB, or (iii) it flows to TCB(d) from other domain TCBs with the legitimate filter.*

Through this definition, Clark-Wilson isolation requires the system's information flow adhere either within a domain TCB from system TCB to a domain TCB, or between domain TCBs via filters. In this paper we do not discuss the integrity of filters, which can be ensured with integrity measurement and attestation mechanisms [15]. Instead, we assume that filters always correctly sanitize data between domains. In practice, filters between domains are application or service dependent. For instance, a firewall works as a filter for mail service by checking incoming mails before sending to a mail application. However, for a service like `logrotate`, the firewall does not work as a filter because `logrotate` enables automatic rotation of log files and does not depend on data in the files. In our work, initially we do not have a set of predefined filters. After detecting a possible policy violation, we try to identify a filter subject to resolve the violation.

3.5 Violation Detection and Resolution

Based on our domain-based Clark-Wilson model, we treat a TCB(d) as an isolated information domain. We use the following two rules for identifying possible policy violations in a system.

Rule 1 *If there is an information flow path from a subject out of the system TCB to the system TCB without passing any identified filter, there is a policy violation for protecting the system TCB.*

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

After initial policy violations are identified with these two rules, we use different strategies to resolve those violations. Specifically, for a violation, we first check if it can be solved by adding or removing related subjects to or from the domain TCB. This causes no change to the policy graph. Secondly, we check if a subject along the violated information flow path can be regarded as a filter. If a filter is identified, then the violation is a false alarm and there is no change to the policy graph. Thirdly, we attempt to modify

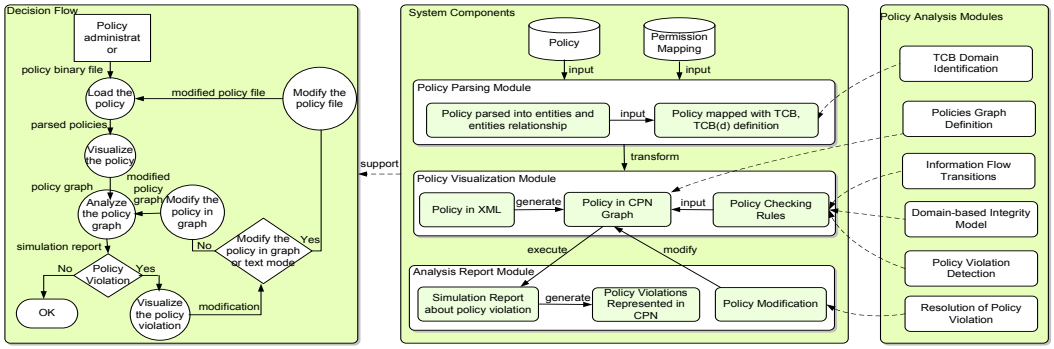

Figure 2. Policy analysis framework.

policy graph, either by excluding subjects or objects from the violated information flow path, or by replacing subjects or objects with more restricted privileges. Finally, we insert filters that act as a high-assurance gateway between unauthorized subjects and protected subjects.

4 Realization and Experiments

We built our prototype based on Colored Petri Nets, which is a powerful graph-based analysis tool for system modeling [13]. A Petri Nets includes three basic components: places, transitions, and arcs. Arc expressions specify a collection of tokens, which can be added to or removed from places. If a transition input place 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 CPN models. The simulation is to validate whether a system works correctly reflecting the design principles. It supports both interactive simulation and automatic simulation. In the interactive simulation, a user can set breakpoints, choose between enabled binding elements, change markings of places, and study the token in detail, while the simulator makes random choices of the enabled binding elements and automatically executes the whole CPN models in the automatic simulation.

Figure 2 shows our policy analysis framework that includes *Policy Analysis Modules*, *System Components*, and *Data Flow* for policy analysis. *Policy Analysis Modules* are composed of several analysis components such as *TCB Domain Identification, Policy Graph Definition, Information Flow Transitions, Domain-based Integrity Model, Policy Violation Detection* and *Resolution of Policy Violation* that are designed based on the policy analysis methodology proposed in Section 3. For the *System Components*, the *Policy Parsing Module* is to parse and map the operations between types to $write\_like$, $read\_like$, or $call$ operations. For example, the operation like `getattr` can be mapped to a $read\_like$ operation. System TCB and domain TCB are discovered and inserted into the policies using the *TCB Domain Identification* module in *Policy Analysis Modules*. Through the *Policy Visualization Module*, a security policy is transformed into an XML file, which is transformed into a CPN graph based on the *Policies Graph Definition* module. A set of policy checking rules are taken as an input into the CPN graph, and the generated CPN graph is executed with the CPN simulation function. The policy checking rules are built by using the modules in *Policy Analysis Modules* such as *Information Flow Transitions, Domain-based Integrity Model*, and *Policy Violation Detection* modules. The *Analysis Report Module* generates a simulation report containing information about policy violations by executing the policies in CPN graph and then the simulation report is transformed into another XML file. The graph along with policy violation specifications is also generated through CPN as an output. In addition, this module supports policy modification performed by *Policy Violation* module. The *Decision Flows* components govern all processes in our framework.

4.1 Policy in XML

Based on the definition of policy graph, we design a scheme to express policies in XML. As shown in Figure 3, 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 out and into other types, and (3) the starting types which cannot have a flow into by any other types. Also, it may contain the types that can only be flew into. The DD element specifies information about non-replicated transition rela-

tionships between types.

In our experiments, we parsed a real world SELinux binary policy file *policy.19* into a defined policy structure based on the source package of APOL [1] with our parsing tool. Then, we retrieved the information about subject types, object types, and related rules. Using LibXML2 [2], we transformed these information into an XML file conforming the defined policy template. Based on our TCB identification strategy mentioned in Section 2.2, we carried out the system TCB identification. The collection of the domain TCBs for all applications and system services was performed based on our TCB(d) identification method. In the example SELinux policy, there are 35 user applications, 37 staff applications, 34 administrative applications, and 84 system services. In user, staff, and sysadm levels, we identified information domains as d_{user}, d_{staff} and d_{sysadm}, respectively. More fine-grained domain identifications are performed against different applications and services within these domains. For example, the `user_mozilla_t` related domain should be isolated from the `user_xserver_t` related domain. Table 2 shows the comprehensive list of TCBs that our tool successfully identified. The list of system service daemons are also shown in Table 2 and each daemon is identified as an individual domain.

```
<?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" />   domain—type relationships
        <dtoperation start="user_games_t" direction="write_like" end="devtty_t" />
        .....
        <dlist d="user_t" />
        <dlist d="user_mozilla_t" />   domains involved in the domain-type related paths
        .....
        <dslist d="kernel_t" />  ——— domains that are the starting points of the domain-type related paths
        .....
        <tlist d="devtty_t" />  ——— types involved in the domain-type related path
        .....
</section>
-<section type ="DD">
        <domaintrans start="sysadm_t" transdomain="mount_t"/>
        <domaintrans start="initrc_t" transdomain="mount_t"/>   domain--domain transition relationship
        .....
        </section>
</policy>
```

Figure 3. SELinux policy rules in XML format.

4.2 Graph-based Policy Analysis and Violation Identification

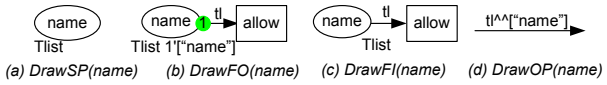

Figure 4. Graph drawing.

To automatically visualize a policy, we use Extensible Stylesheet Language (XSL) [4] to transform a policy in

Table 2. TCB(d) identified.

TCB(d_{user})			
user_crond_t	user_crontab_t	user_xserver_t	
user_gpg_t	user_home_ssh_t	user_mozilla_dbusd_system_t	
user_mail_t	user_screen_t	user_gpg_helper_t	
user_tvtime_t	user_mplayer_t	user_mozilla_javaplugin_t	
user_lpr_t	user_spamc_t	user_games_t	
user_uml_t	user_mozilla_t	user_dbusd_user_t	
user_ssh_t	user_gag_agent_t	user_ssh_agent_t	
user_gph_t	user_chkpwd_t	user_ssh_keysign_t	
user_locate_t	user_vmware_t	user_dbusd_system_t	
user_dbusd_t	user_lockdev_t	user_spamssassin_t	
user_irc_t	user_cdrecord_t	user_gpg_pinentry_t	
user_xauth_t	user_mencoder_t		
TCB(d_{staff})			
staff_crontab_t	staff_xserver_t	staff_mozilla_javaplugin_t	
staff_gpg_t	staff_mencoder_t	staff_ssh_keysign_t	
staff_su_t	staff_cdrecord_t	staff_home_ssh_t	
staff_ssh_t	staff_vmware_t	staff_dbusd_system_t	
staff_gph_t	staff_chkpwd_t	staff_userhelper_t	
staff_crond_t	staff_screen_t	staff_gpg_helper_t	
staff_lpr_t	staff_xauth_t	staff_spamassassin_t	
staff_locate_t	staff_lockdev_t	staff_mplayer_t	
staff_irc_t	staff_spamc_t	staff_mozilla_dbusd_system_t	
staff_mail_t	staff_games_t	staff_gpg_agent_t	
staff_sudo_t	staff_mozilla_t	staff_gpg_pinentry_t	
staff_uml_t	staff_tvtime_t	staff_ssh_agent_t	
staff_dbusd_t			
TCB(d_{sysadm})			
sysadm_su_t	sysadm_mozilla_t	sysadm_mozilla_javaplugin_t	
sysadm_gph_t	sysadm_spamc_t	sysadm_spamassassin_t	
sysadm_sudo_t	sysadm_games_t	sysadm_mozilla_dbusd_system_t	
sysadm_uml_t	sysadm_cdrecord_t	sysadm_ssh_agent_t	
sysadm_irc_t	sysadm_crontab_t	sysadm_gpg_agent_t	
sysadm_crond_t	sysadm_passwd_t	sysadm_userhelper_t	
sysadm_gpg_t	sysadm_mplayer_t	sysadm_ssh_keysign_t	
sysadm_mail_t	sysadm_xserver_t	sysadm_dbusd_sysadm_t	
sysadm_xauth_t	sysadm_dbusd_t	sysadm_dbusd_system_t	
sysadm_ssh_t	sysadm_screen_t	sysadm_gpg_pinetry_t	
sysadm_lpr_t	sysadm_chkpwd_t	sysadm_gpg_helper_t	
sysadm_vmware_t			
Daemons			
uml_switch_t	postfix_bounce_t	vpnc_t	nfsd_t
canna_t	cyrus_t	dhcpc_t	udev_t
ipsec_t	mdadm_t	apmd_t	ntpd_t
ssh_keygen_t	telnetd_t	zebra_t	nscd_t
dovecot_auth_t	slrnpull_t	saslauthd_t	howl_t
prelink_t	tftpd_t	sendmail_t	rpm_t
i18n_input_t	fetchmail_t	smbd_t	gpm_t
privoxy_t	bluetooth_t	nmbd_t	lpd_t
ypbind_t	ypserv_t	crond_t	pppd_t
cupsd_t	mysquld_t	squid_t	mrtg_t
dhcpd_t	dictd_t	arpwatch_t	hald_t
cpuspeed_t	inetd_t	httpd_t	innd_t
spamd_t	timidity_t	irqbalance_t	acct_t
vmware_t	winbind_t	kadmind_t	xdm_t
networkmanager_t	dmesg_t	radiusd_t	rshd_t
printer_t	rhgb_t	radvd_t	ptal_t
slapd_t	gssd_t	auditd_t	rpcd_t
fingerd_t	named_t	dovecot_t	ftpd_t
krb5kdc_t	procmail_t	hostname_t	rdisc_t
postgresql_t	updfstab_t	cpucontrol_t	ping_t
games_t	iptables_t	sound_t	

XML to CPN graph. The following basic functions are designed for the CPN graph generation as shown in Figure 4.

- *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

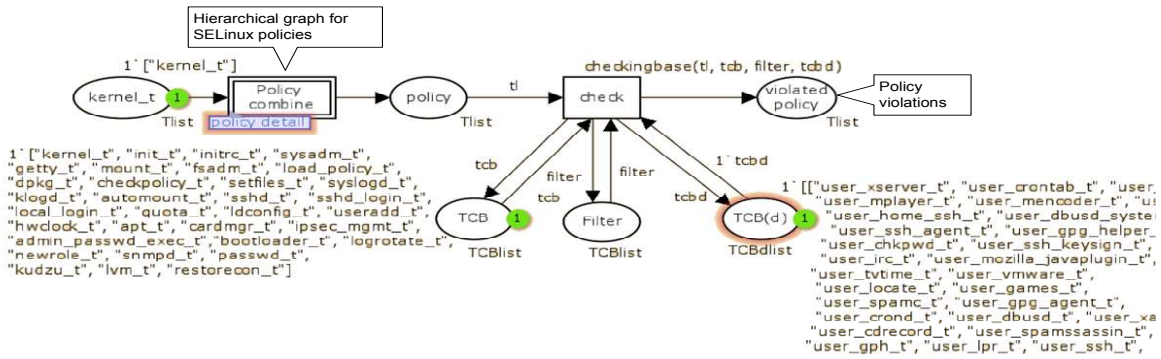

Figure 5. Policy visualization with hierarchy.

information contained in the place and the transition called *allow*. *dlist* is a color set of the place expressing 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 initial markings.
- *DrawFI(name)* is to draw places containing types from which no information can flow out.
- *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.

With the above 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. As stated in Section 3.4, it is hard to initially identify filters so we set initial filter values to *null* and put this information into the policy XML file. Later, when possible policy violations were detected, filters were identified and added to the XML file through CPN. Figure 5 illustrates a snapshot of our policy visualization.

Simulation is a technique supported by CPN to analyze a system by conducting controlled experiments [13]. 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 results are stored in a simulation report. Through parsing the simulation report, we generate 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 6 shows an identified policy violation, specifying that some NON-TCB domain types can write to the type `devtty_t`, which can be read by some TCB types. As examples of system TCB isolation violations, Table 3 includes the identified policy violations caused by the identified information flow path from NON-TCB subjects to system TCB subjects `fsadm_t` and `snmpd_t`. As examples of TCB(d) isolation violations, Table 3 also shows all policy violations identified for protecting `sysadm_xserver_t`, which belongs to TCB(d_{sysadm}).

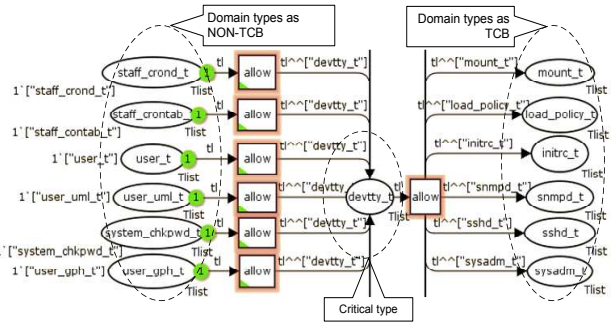

Figure 6. Example violations.

Based on the principles proposed in Section 3, our experiments also dealt with corresponding methods for resolving the policy violations. There are policy violations ignored because of its relationship to the filter. For example, the policy violations caused by writing and reading `devlog_t` can be ignored. `devlog_t` is used to label the log files of `syslogd_t`, which provides two system utilities: system logging and kernel message trapping. `devlog_t` is also used to store log information rather than the information of certain resources. Hence, even though there is an information flow from NON-TCB subject `user_mozilla_t` to TCB subject `fsadm_t` through `devlog_t`, it is a policy violation that can be ignored.

Most policy violations need to be resolved through policy modifications. For example, to the policy violations caused by the read and write accesses to

Table 3. Policy violation examples.

fsadm_t, snmpd_t, and mount_t related violations			
Subject	Type:Class	Subject	Resolution
200	network	fsadm_t	Filter
2	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
3	scsi_generic_device_t:chr_file	fsadm_t	Modify
200	devlog_t:sock_file	fsadm_t	Ignore
200	network	snmpd_t	Filter
2	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
104	initrc_devpts_t:chr_file	snmpd_t	Modify
200	devlog_t:sock_file	snmpd_t	Ignore
sysadm_xserver_t related violations			
mta_agent_t	sysadm_tty_device_t:chr_file	sysadm_xserver_t	Modify
sound_t	sysadm_tty_device_t:chr_file	sysadm_xserver_t	Modify
158	network	sysadm_xserver_t	Filter
10	zero_device_t:chr_file	sysadm_xserver_t	Modify
158	devtty_t: chr_file	sysadm_xserver_t	Modify

devtty_t, our solution is to redefine devtty_t by introducing user_devtty_t, sysadm_devtty_t, staff_devtty_t, system_devtty_t, and daemon_devtty_t. Corresponding policy rules are also modified as follows:

```
allow user_mozilla_t devtty_t:chr_file
{read write getattr ioctl}; is changed to
allow user_mozilla_tuser_devtty_t:chr_file
{read write getattr ioctl};
```

Also, the domain-based Clark-Wilson model is applied to some policy violations such as network related policy rules for violation resolution. The domain type transition relationships between the types and network filter domains need to be defined. We can assign the access of network types using network filters as follows:

```
allow user_xserver_t
networkfilter_t:process transition; or
allow networkfilter_t node_type:node
{udp_send rawip_send rawip_recv};
```

5 Concluding Remarks

In this paper, we have proposed a domain-based Clark-Willson model to analyze system security policies. In particular, our general method shows how we can identify system TCB and domain TCBs in the context of information domains in a system, and presents a set of rules to detect all possible policy violations from NON-TCB to system TCB, and between domain TCBs. We also automated the analysis processes using CPN and visualized graph-based violations. We adopted SELinux policy as an example to demonstrate the functionality and effectiveness of our methodology.

When one or more policy rules are modified in a policy, with our current approach the policy has to be re-loaded and a complete analysis is required to check the resolution results. Developing a fully automatic and dynamic approach for policy analysis remains as our future work, since manual analysis is still needed to identify real violations after the CPN-based analysis in our method.

References

[1] Tresys Technology APOL. Available at http://www.tresys.com/selinux/.

[2] XML Organization. The XML C parser and toolkit of Gnome. Available at http://xmlsoft.org/.

[3] XSB. Available at http://xsb.sourceforge.net/.

[4] XSL. Available at http://www.w3.org/TR/xsl/.

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

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

[7] Y.-M. Chen and Y.-W. Kao1. Information flow query and verification for security policy of security-enhanced linux using cpn. In *International Workshop on Security (IWSEC)*, 2006.

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

[9] D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.

[10] 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.

[11] T. Jaeger, R. Sailer, and X. Zhang. Analyzing integrity protection in the selinux example policy. In *Proceedings of the 12th conference on USENIX Security Symposium*, 2003.

[12] T. Jaeger, X. Zhang, and A. Edwards. Policy management using access control spaces. *ACM Trans. Inf. Syst. Secur.*, 6(3):327–364, 2003.

[13] K. Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use, volume 3*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[14] B. Lampson. Protection. In *5th Princeton Symposium on Information Science and Systems*, pages 437–443, 1971. Reprinted in *ACM Operating Systems Review* 8(1):18–24, 1974.

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

[16] R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, vol.26(no.11):9–19, 1993.

[17] B. Sarna-Starosta and S. D. Stoller. Policy analysis for security-enhanced linux. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS)*, 2004.

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