Towards System Integrity Protection with Graph-Based Policy Analysis

Wenjuan Xu[1], Xinwen Zhang[2], and Gail-Joon Ahn[3]

[1] University of North Carolina at Charlotte, wxu2@uncc.edu
[2] Samsung Information Systems America, xinwen.z@samsung.com
[3] Arizona State University, gahn@asu.edu

Abstract. Identifying and protecting the trusted computing base (TCB) of a system is an important task, which is typically performed by designing and enforcing a system security policy and verifying whether an existing policy satisfies security objectives. To efficiently support these, an intuitive and cognitive policy analysis mechanism is desired for policy designers or security administrators due to the high complexity of policy configurations in contemporary systems. In this paper, we present a graph-based policy analysis methodology to identify TCBs with the consideration of different system applications and services. Through identifying information flows violating the integrity protection of TCBs, we also propose resolving principles to using our developed graph-based policy analysis tool.

1 Introduction

In an operating system, an information flow occurs when one process can write to a resource (e.g., device or file) which can be read by another process. Inter-process communication between these processes may also cause possible information flows. Integrity goal is violated if there exists information flow from an unprivileged process to a privileged process or between two different applications or services where information flow should be controlled. In a typical system, information flows are controlled through security policies. Our objective in this paper is to provide an effective framework for analyzing system security policies and finding policy rules causing information flows with integrity violations.

There are several related approaches and tools to analyze policies based on certain information flow models [1, 9, 11, 16]. Most of these work focus on the identification of a common and minimum system TCB 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 flowing from other applications. An existing argument in [20] clearly states the 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." Accordingly, a more comprehensive policy analysis for TCB identification and integrity 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 [12] policy has over 30,000 policy statements in a desktop environment. Under this situation, several challenges exist for system designers or administrators in the context of policy analysis. We enumerate such challenges to derive the motivation of this paper: (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 [9, 16], 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. [11, 17], 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 this paper, we consider an information domain as a collection of *subjects* (e.g., processes) and *objects* (e.g., files, ports, devs) which jointly function for an application or service. We further define the *domain TCB* as subjects which should have the same integrity level in the domain. The integrity of the domain cannot be judged unless information flows between this domain TCB and the rest of the system are appropriately controlled. Based on these notions, we propose a *domain-based integrity model*. Based on this model, we build a graph-based policy analysis methodology for identifying policy rules that cause integrity violations (or simply *policy violations*). Our graph-based approach can provide several benefits since information visualization helps a user heuristically explore, query, analyze, reason, and explain obtained information. Also, our graph-assisted approach simplifies analysis and verification tasks while rigorously producing distinctive representation of analysis results. In addition, we describe a set of principles for resolving identified policy violations in policy graphs. A graph-based policy analysis tool (GPA) is also developed based on our approach.

The rest of this paper is organized as follows. Section 2 describes background and some work related to our graph-based policy analysis. The principles and methodology of our work are illustrated in Section 3. Section 4 presents how to develop and use GPA for analyzing policies. In this section, we adopt SELinux policies as an example. Section 5 concludes this paper presents our future work.

2 Background and Related Work

2.1 Trusted Computing Base

The concept of TCB partitions a system into two parts: the part inside TCB which is referred as to be trusted (TCB) and the part outside TCB which is referred as to be untrusted (NON-TCB). Therefore, the identification of TCB is always a basic problem in security policy design and management. The famous Orange Book [2] proposes TCB as part of a system that is responsible for enforcing information security policies. Reference monitor-based approach is proposed in [4], where a system's TCB not only includes reference monitor components, but also encompasses all other functionalities

that directly or indirectly affect the correct operation of the reference monitor such as object managers and policy database. Considering an operating system, its TCB includes kernel, system utilities and daemons as well as all kinds of object management and access control functions. Typical object management functions are responsible for creating objects and processing requests while typical access control functions consist of both rules and security attributes that support decision-making for access control.

2.2 Integrity Model

To describe information flow-based integrity protection, various models are proposed and developed in past years, such as Biba [7], Clark-Wilson [15], LOMAC [19] and CW-lite [17]. 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. LOMAC supports high integrity process's reading low integrity data, while downgrading the process's integrity level to the lowest level that it has ever read. Clark-Wilson 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. Later the concept of TP is evolved into filter in CW-Lite 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 (NON-TCB) to high integrity processes (TCB) through filters.

2.3 Policy Analysis

The closest existing work to ours include Jaeger et al. [11] and Shankar et al. [17]. In these work, they use a tool called Gokyo for checking the integrity of a proposed TCB for SELinux [18]. 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 variant trust relationships. Still, achieving the integrity assurance for these applications and services is not addressed in Gokyo.

Several query-based policy analysis tools have been developed. APOL [1] is a tool developed by Tresys Technology to analyze SELinux policies. SLAT (Security Enhanced Linux Analysis Tool) [9] defines an information flow model and policies are analyzed based on this model. PAL (Policy Analysis using Logic Programming) [16] uses SLAT information flow model to implement a framework for analyzing SELinux policies. All these tools try to provide a way for querying policies. However, they all display policies and policy query results in text-based expressions, which are difficult to understand for policy developers or security administrators. Other policy analysis methods are also proposed. For example, in [23], they propose to analyze the policies with information-flow based method. For another example, in [22], they try to analyze the policies with graph-based model. However, non these approaches are applicable in our scenarios since they are specific to certain access control model, and none of them are realized with certain tools.

To overcome these issues, we have developed a graph-based methodology for identifying and expressing interested information flows in SELinux policies [21]. We also

have proposed a policy analysis mechanism using Petri Nets is proposed in [3]. However, this work does not have the capability of policy query, thus is limited in identifying system TCB and other domain TCBs.

3 Graph-based Policy Analysis

To help a policy administrator better understand security policies and perform policy analysis tasks, we first propose a graph-based policy analysis methodology in this section. Graphs leverage highly-developed human visual systems to achieve rapid uptake of abstract information [10]. In our methodology, we have two parallel building blocks: basic policy analysis and graph-based policy analysis. The basic policy analysis is composed of security policy definitions, integrity model for identifying policy violations, and methods for resolving policy violations. Graph-based analysis is built according to basic policy analysis and expresses corresponding components with graphical mechanisms and algorithms. We elaborate the details of our methodology in the remainder of this section.

3.1 Basic Policy Analysis

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 $read\_like$ [9] and operations between subjects can be expressed as $calls$. 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*, 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 $read(y, x)$. 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 $call(y, x)$. 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 subject s_1 to s_2, denoted as $flowtrans(s_1, s_2)$. Also, if 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*.

Integrity Model Retrospecting the integrity models introduced in Section 2, 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 boundary. With that reason, we develop a *domain-based integrity 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 paper. 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.

Information Domain As mentioned in Section 1, 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 in this domain.
- *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 have 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 other subjects should be added to domain TCB or dependencies should be removed. Based on these principles, we first identify an 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 this domain. Here, we identify the initial TCB(d) subjects based on these subject launching relationships and the number of subjects that a subject can launch. For example, for web server domain, *httpd* launches all other processes like *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 identified 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 (or integrity violations described shortly in this subsection), we adjust the identified TCB(d) with wrongly included or excluded subjects. For example, initially subject *awstats_script* (web server statistics script) is excluded from TCB(d). After identifying policy violations caused from *awstats_script* to web server TCB(d), we found that these violations can be ignored. Hence, the TCB(d) should be adjusted to include *awstats_script*.

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

Definition 1. *Domain-based integrity 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.*

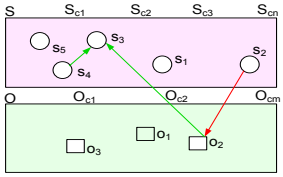
Through this definition, domain-based integrity model achieves the integrity of an information domain by isolating information flow to TCB(d). This model requires that any information flow happening in a domain *d* adheres within the TCB(d), from system TCB to the TCB(d), or from another domain TCB via filter(s). In this paper we do not discuss the integrity of filters, which can be ensured with other mechanisms such as formal verification or integrity measurement and attestation [14]. Filters can be processes or interfaces that normally is a distinct input information channel and is created by, e.g., a particular open(), accept() or other call that enables data input. For example, linux *su* process allows a low integrity process (e.g., staff) changes to be high integrity process (e.g., root) through calling *passwd* process. 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*. Normally, it is the developer's tasks to build filtering interfaces and prove effectiveness to the community [13]. 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 [17]. 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 filter subject to resolve policy violations.

Policy Violation Detection Based on domain-based integrity 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).*

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.



(a) Design 1- 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

Algorithm 1: [Building 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.*

(b) Algorithm for building policy graph

Fig. 1. Designation and algorithm for expressing policies in graph.

3.2 Graph-based Analysis

Semantic substrates [5] is a visualization methodology for laying out a graph, in which graph nodes are displayed in non-overlapping regions based on node attributes. Through this way, the location of a node conveys its information. Also, the visibility of graphic links used to describe node relationships is available depending on user control. In our work, we use semantic substrates to display *policy graph* and *policy violation graph* which are defined based on the previously stated security policies and policy violations. A graphical query-based violation identification method is introduced based on domain-based integrity model. Also, we illustrate how we can apply policy violation resolution methods to policy violation graphs.

Policy Graph A security policy consists of a set of subjects, objects, and operations including *write, read* and *call*. 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* $call(s1, s2)$*.*

As concluded in [8], humans perceive data coded in spatial dimensions far more easily than those coded in non-spatial ones. 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 1 (a) 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 subject s_2 and object o_2 is expressed with a red link.

Different security policies have different formats and components. To give a uniform way for policy analysis, we need to preprocess a primitive policy. Figure 1 (b) 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 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.

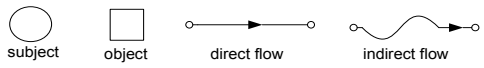


Fig. 2. Policy query building blocks.

Violation Identification with Graphical Queries According to our domain-based integrity model, we need to investigate information domain, domain TCB, and then identify policy violations. To complete these tasks, we design a graphical user interface to create and run queries against a policy graph, and then get required information such as which subject should be included in the information domain. A query formulation is built with four basic components–*Subject*, *Object*, *Direct flow* and *Indirect flow*. Figure 2 summarizes these visual components and details are elaborated as follows.

– *Subject* is shaped as a labelled circle node to indicate policy subject(s) in a query. A user can define the Subject node as a single subject or a set of subjects based

on different query requirements. The label under the Subject node specifies user defined subject names. For instance, a user can query to find a policy subject *httpd*, a set of policy subjects *httpd, php*, and *ssh*, or any sequence of policy subjects "*" from a policy graph. Color filled in the Subject node represents user defined policy subject attribute for a query. An example of the attribute can be the name of an information domain which is expressed in green color. Therefore, a user can create an example query with green color in the Subject node which is labelled with *httpd*. The meaning of this query is to find a subject which belongs to the information domain and has name *httpd*. After the subject is found, it is changed to an expected color (e.g., red), which is defined by the user through specifying the Subject node line color. Also, if the query purpose is to identify policy subject(s), then wildcard "?" should be marked on the Subject node.

- *Object* is represented as a labelled rectangle node to denote policy object(s) in a query. Similarly, the label under the Object node illustrates a single policy object (e.g., *etc*) or a set of policy object (e.g.,*etc* or *bin*). Also, a user can specify policy object attribute (e.g., domain name) for a query and the color of identified policy object in the result graph through coloring the Object node rectangle and rectangle line, respectively. In the situation that the query purpose is to identify objects, "?" is specified on the Object node.
- *Direct flow* is drawn with a link to connect Subject node and Object node and provides a way for direct information flow-based query. The label of a link represents the intended information flow. For example, a user can query to find if a write operation exists between a policy subject and a policy object. To specify that the intended query is to identify a direct flow, a user can denote the Direct flow link with "?".
- *Indirect flow* is expressed in a curved link to connect Subject node and Object node. The main purpose of indirect flow is to specify the intended information flow paths between subjects and objects. A user can find the shortest path, all paths, or any path. The wildcard "*" denotes all paths that can be found between subjects and objects. If the intended query is to identify indirect flow paths, "?" should be drawn on the Indirect flow link.

Information Domain Queries Corresponding to two steps of information domain identification presented in Section 3.1, we propose two principles as follows.

- *Name-based policy subjects or objects query* To identify domain subjects and objects based on keyword, we construct subject and object queries by using their names. Figure 3 (a) shows an example query: *Identifying the subjects or objects whose names have prefix "prefix", and painting result nodes with green color*. The query result is shown in Figure 3 (a'), where subjects S_1, S_3 and object O_1 are identified to be included in the information domain.
- *Direct flow-based subjects or objects query* To investigate domain subjects and objects based on direct flows, we construct an example query shown in Figure 3 (b). The meaning of this query is: *Finding the subjects or objects that can directly flow to the initial identified domain subjects and objects (green colored), and painting result nodes with green color*. The query result in Figure 3 (b') indicates that subjects S_2, S_4 and object O_2 should be added into the information domain.

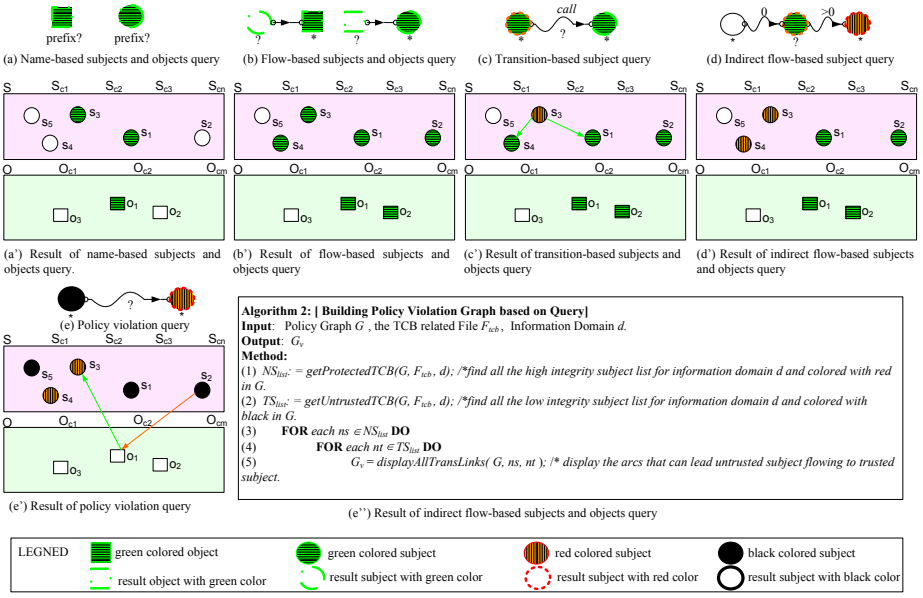


Fig. 3. Policy query examples.

Domain TCB Queries For domain TCB identification steps described in Section 3.1, we query TCB(d) for an information domain with following principles. After TCB(d) queries are completed, TCB(d) subject information is saved into a file F_{tcb}.

- *Transition-based subjects query* To query a TCB(d), we first identify TCB(d) based on subject launching relationships. The example query in Figure 3 (c) states: *Identifying and displaying the direct call flow links between domain subjects*. Example result is shown in Figure 3 (c'), which displays the call transition relationships between subjects S_1, S_2 and S_3. Hence, subject S_3 belongs to TCB(d).
- *Indirect flow-based subjects query* To identify the subjects that can flow only to initial TCB(d) as shown in Figure 3 (d), a query is constructed as follows: *Identifying the subjects that belong to the information domain (green colored) can flow only to the initial TCB(d) (red colored) with red color in result nodes*. Figure 3 (d') indicates the example result that S_4 should be added into TCB(d).

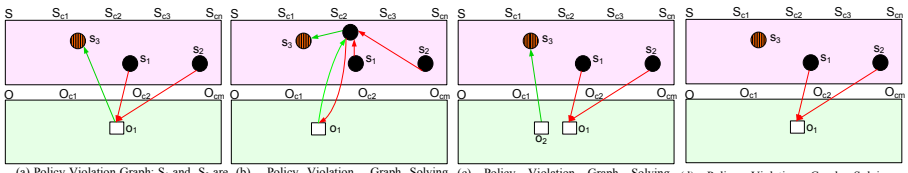
Policy Violation Queries Before introducing our methodology for policy violation queries, we first define a violation policy graph based on our proposed integrity model.

Definition 3. *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 ∂ is a subgraph of G where*

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

Figure 3 (e') shows a violation path with a flow transition from a low integrity subject (black) to a high integrity subject (red). To generate the policy violation graph, a query is constructed as shown in Figure 3 (e), where we draw in the Subject node with black color (NON-TCB) and red color (TCB), trying to identify the policy violations from NON-TCB to TCB caused by indirect information flow. Figure 3 (e") shows the details of policy violation graph generation based on query. Through the query operations performed earlier, 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 "*".

Policy Violation Resolutions in Graph With a generated policy violation graph, we introduce different approaches to modify the policy graph and remove policy violations and illustrate the expected graph result after the modification. Based on the policy violation resolution strategies discussed in Section 3.1, 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 Figure 4 (a) with violation resolved graph in Figure 4 (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 (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. Example result of the modification is shown in Figure 4 (d), where the read operation between object O_1 and TCB subject S_3 is removed to resolve policy violations between NON-TCB and TCB.



(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

Fig. 4. Policy violation and modification example graphs

4 Graph-based Policy Analysis

Our previous policy visualization tool [21] has shown the feasibility and benefits to visualize security policies with *semantic substrates* [5]. Extended from this, a more comprehensive policy analysis tool (GPA) is developed in our work, which implements the

query functionality for achieving high system assurance with information flow control. In this section we use an example SELinux reference policy to show the flexibility and efficiency of policy analysis with our tool. We use JDK1.6 and other necessary Java libraries to develop the main analysis components. We implemented graph drawing with graph package Piccolo [6].

Applying the reference monitor-based TCB identification method to SELinux, subjects functioning as reference monitor such as checking policy and loading policy belong to system TCB. Also, processes used to support reference monitor such as kernel and init, are included into system TCB. After reference monitor-based system TCB identification is completed, other subject types such as *restorecon* are required to be included into system TCB based on their flow transition relationship with the initial system TCB. Table 1 shows the TCB domains that we have recognized.

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.

Apache Domain Identification To identify subjects and objects for Apache domain, we first apply keyword-based identification principle. As discussed earlier, we use *http* as a keyword prefix. As a result, subjects and objects such as `httpd_t`, `httpd_php_t` are included into Apache domain. With the flow-based identification principle, 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 1 shows a selected list of subjects and objects that we detected. Also, we use graph-based query functions implemented in GPA to automatically identify Apache information domain. Figures 5.I (a) and 5.I (b) show how we use graph queries to identify subjects and objects corresponding to Apache domain identification steps.

Apache TCB(d) Identification Based on the initial TCB(d) identification principle in Section 3, 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 labelled by `httpd_t` is a predominant subject which launches other subjects in Apache server. Similarly, a subject labelled as `httpd_suexec_t` is also a predominant subject since this domain can transit to most of other `httpd` domains. Naturally, `httpd_t` and `httpd_suexec_t` are included into Apache TCB(d). Secondly, we construct a query to find all subjects that can transit only to the initially identified TCB(d) (shown in Figure 5.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.

Violations and Resolutions for Apache Based on the example query drawn in Figure 5.I (e), we automatically identify possible policy violations from NON-TCB subjects to TCB(d) subjects in Apache service. Figure 5(II) shows the identified policy

violations, which implies that TCB(d) integrity is violated because NON-TCB subjects have write_like operations on objects that can be read by TCB(d) subjects. Due to possible high number of violations, our GPA tool can selectively show certain violations, allowing a policy administrator to solve them based on the priority.

Table 1. Apache information domain

System TCB				
kernel_t	load_policy_t	initrc_t	bootloader_t	quota_t
mount_t	ipsec_mgmt_t	useradd_t	automount_t	passwd_t
hwclock_t	admin_passwd_exec_t	cardmgr_t	checkpolicy_t	fsadm_t
kudzu_t	sshd_login_t	restorecon_t	newrole_t	klogd_t
syslogd_t	sysadm_t	getty_t	apt_t	sshd_t
dpkg_t	logrotate_t	snmpd_t	ldconfig_t	init_t
lvm_t	local_login_t	setfiles_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_helper_t
httpd_unconfined_script_t	httpd_php_t	httpd_sysadm_script_t	httpd_sys_script_t	httpd_suexec_t
httpd_prewikka_script_t	httpd_apcupsd_cgi_script_t	httpd_user_script_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_php_t
httpd_sysadm_script_t	httpd_prewikka_script_t	httpd_rotatelogs_t	httpd_apcupsd_cgi_script_t	

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 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

Table 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

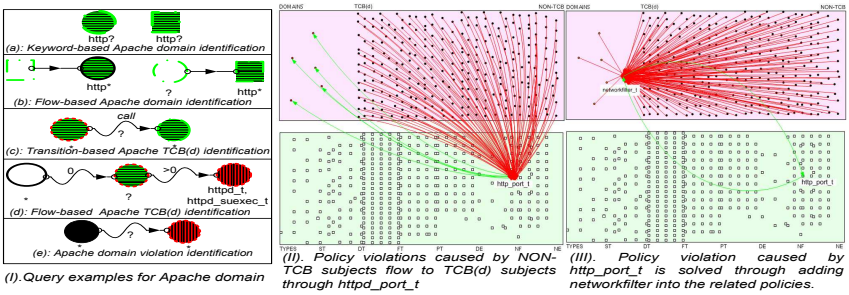


Fig. 5. Policy violations and solving

be included into TCB(d) since it would lead unlimited file access for the staff services such as `staff_t`, `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 web server information. To resolve the policy violations caused due to the write access to `httpd_sys_content_t`

by webalizer_t, we remove the policy rule stating write_like operation 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 can be 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 the policy violation related domains and types. Comparing Figure 5 (c) with Figure 5 (b), 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 Conclusion

In this paper, we have proposed a graph-based policy analysis framework to effectively verify complex security policies for integrity protection, based on a proposed domain-based integrity model. We develop an intuitive visualization tool to demonstrate the feasibility of our framework. Additionally, we discuss how we can use our framework to analyze SELinux policies and the results demonstrate the effectiveness and efficiency of our methodology. We believe that this is the first effort to formulate a general policy analysis framework with graph-based approach. We are going to develop a violation graph based ranking schema, which can be used to help resolving the policy violations. Also, the userability of the graph-based policy analysis tool will be studied. In addition, we plan to enhance the flexibility of our approach and investigate how our policy analysis framework and tool can be utilized with other integrity models.

References

1. Tresys Technology Apol. http://www.tresys.com/selinux/.
2. *Trusted Computer System Evaluation Criteria*. United States Government Department of Defense (DOD), Profile Books, 1985.
3. G. Ahn, W. Xu, and X. Zhang. Systematic policy analysis for high-assurance services in selinux. In *Proc. of IEEE Workshop on Policies for Distributed Systems and Networks*, 2008.
4. A. P. Anderson. Computer security technology planning study. *Technical Report ESD-TR-73-51*, II, 1972.
5. A. Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733–740, 2006. Senior Member-Ben Shneiderman.
6. H. C. I. L. at University of Maryland. Piccolo. *Available from http://www.cs.umd.edu/hcil/jazz/download/index.shtml*.
7. K. J. Biba. Integrity consideration for secure compuer system. Technical report, Mitre Corp. Report TR-3153, Bedford, Mass., 1977.
8. M. Green. Toward a perceptual science of multidimensional data visualization: Bertin and beyond. *Available from http://www.ergogero.com/dataviz/dviz2.html*, 1998.
9. 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.
10. I. Herman, G. Melancon, and M. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.
11. T. Jaeger, R. Sailer, and X. Zhang. Analyzing integrity protection in the selinux example policy. In *Proc. of USENIX Security Symposium*, 2003.
12. P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *USENIX Annual Technical Conference, FREENIX Track*, 2001.
13. N. Provos, M. Friedl, and P. Honeyman. Preventing privilege escalation. *12th USENIX Security Symposium*, page 11, August 2003.
14. 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.
15. R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.
16. B. Sarna-Starosta and S. D. Stoller. Policy analysis for security-enhanced linux. In *Proceedings of the 2004 Workshop on Issues in the Theory of Security*, 2004.
17. U. Shankar, T. Jaeger, and R. Sailer. Toward automated information-flow integrity verification for security-critical applications. In *NDSS*. The Internet Society, 2006.
18. S. Smalley. Configuring the selinux policy. http://www.nsa.gov/SELinux/docs.html, 2003.
19. T.Fraser. Lomac: Low water-mark integrity protection for cots environment. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2000.
20. WIKIPEDIA. Trusted computing base, http://en.wikipedia.org/wiki/Tusted_Computing_Base.
21. W. Xu, M. Shehab, and G. Ahn. Visualization based policy analysis: case study in selinux. In *Proc. of ACM Symposium of Access Control Models and Technologies*, 2008.
22. H. Wang and S. Osborn. Discretionary access control with the administrative role graph model. In *Proc. of ACM Symposium of Access Control Models and Technologies*, 2007.
23. S. Osborn. Information flow analysis of an RBAC system. In *Proc. of ACM Symposium of Access Control Models and Technologies*, 2002.