Visualization Based Policy Analysis: Case Study in SELinux*

Wenjuan Xu, Mohamed Shehab, Gail-Joon Ahn
Dept. of Software and Information Systems
University of North Carolina at Charlotte
Charlotte, NC, USA
{wxu2,mshehab,gahn}@uncc.edu

ABSTRACT

Determining whether a given policy meets a site's high-level security goals can be difficult, due to the low-level nature and complexity of the policy language, and the multiple policy violation patterns. In this paper, we propose a visualization-based policy analysis framework that enables system administrators to visually query and visualize SELinux security policies and to easily identify the policy violations. We propose and formalize both a semantic substrate and adjacency matrix visualization techniques for policy visualization. Furthermore, we propose a visual query language for expressing policy queries in a visual form. Our framework is targeted towards enabling the average administrator by providing an intuitive cognitive sense about the policy, policy queries and policy violations. We also describe our implementation of a visualization-based policy analysis tool that provides the functionalities discussed in our framework.

Categories and Subject Descriptors

D.1.7 [**Programming Techniques**]: Visual Programming; D.4.6 [**Operating Systems**]: Security and Protection—*Information flow controls*

General Terms

Security, Verification

1. INTRODUCTION

In computing systems security policies are specified to implement security goals such as access to protected resources, information flow to and from protected resources, resource isolation and separation of duty. Policy administration is a challenging task due to the complexity, and interdependence of policy rules. This is further exacerbated by the large policy size, for example, the Secure-Enhanced Linux (SELinux) policy includes over 30,000 statements [22]. Access control systems can become significantly ineffective if the implemented policies are not representative to targeted security goals. Simple policy misconfigurations might allow an unprivileged process A to write to some resource that can be read by a privileged process B, causing information flow from process A to B leading to an integrity violation. System administrators use policy analysis tools to locate and correct policy violations. Several policy analysis frameworks have focused on information flow models [35, 19, 17, 18, 28, 16, 13, 30] to enable policy verification and testing. Policy analysis frameworks assume that the policy administrator is a security expert that completely understands and interprets all the policy rules. Policy analysis requires the administrator to be proficient in a custom text based policy analysis expressions. Furthermore, such analysis would locate policy violations; however it would not go further to indicate the effect of such violations. The output of policy analysis tools is list of possible violations, which does not give the system administrator a clear view of how the violation originated and how it might propagate in the system. Information visualization [14] enables users to explore, analyze, reason and explain abstract information by taking advantage of their visual cognition. Several disciplines have adopted information visualization mechanisms to better understand and reason about the collected data. For example, visualization techniques have been adopted in bio-informatics, networks, data mining, information retrieval, social networks and several other areas. In the security arena, visualization has been used to better understand and present data related to network attacks [39, 40, 23, 41], intrusion detection [11, 25, 15, 36] , firewall policies [21, 24, 37], and trust negotiations [38]. In this paper, we propose a policy analysis framework that is based on information visualization principles to simplify policy analysis and to provide a better understanding to the policy administrator.

A policy visualization framework should provide mechanisms to both display and query the policy base. Our framework models the security policy as a policy graph and adopts both the *semantic-substrates* [12, 3] and *adjacency-matrix* [32, 20] mechanisms to generate policy layouts for displaying policy portions. In the semantic substrates mechanism, the nodes and links expressing policy statements are arranged based on the semantic classifications, which provides a systematic approach to trace policy rules. The adjacency-matrix mechanism provides an intuitive approach to trace the read and write relationships between subjects and ob-

*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).

SACMAT'08, June 11–13, 2008, Estes Park, Colorado, USA.

jects. By providing simple and descriptive policy graph layouts it enables the policy administrator to intuitively examine and understand the policy. Another novel module in our framework is the visual query formulation, that enables the administrator to build queries against the policy base by simply dragging and connecting provided query components. This mechanism follows an approach similar to the query by example mechanism used for relational databases [26, 27]. Using a graphical query platform enables the average administrator to easily probe the policy for violations by specifying graphical queries, without the need to write any script or learn a new query language. We implemented our policy visualization framework and developed a Policy Visualization Analysis (PVA) tool. Then we applied it to visualize and query SELinux policies.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 provides an overview of SELinux, trusted computing base and information flow models. In Section 4, we introduce our visualization-based policy analysis framework. The policy visualization approaches are discussed in Section 5. The policy query classification and query execution are presented in Section 6. Our policy visualization tool (PVA) is presented in Section 7. The conclusion and future work are discussed in Section 8.

2. RELATED WORK

Previous typical methods and tools developed to analyze SELinux policies include Gokyo [19, 28, 17, 16], SLAT [10], PAL [31] and APOL [35]. Gokyo was used to check integrity of a proposed trusted computing base (TCB) for SELinux. Integrity of the TCB holds if there is no type 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. Because Gokyo only identifies one common TCB in SELinux and SELinux has multiple security goals with obviously different kinds of trust relationship, Gokyo can not cover all the aspects of policy violations. SLAT (Security Enhanced Linux Analysis Tool) defines an information flow model and the SELinux policies are analyzed based on this model. In the information flow model, SLAT characterizes information flow caused by allowed operations for a given policy. It defines the information flow relation (write operation transfer information from process to resource; read operation transfer information from resource to process) as the flow transition. Then through this flow transition relationship, a path is defined to reflect a sequence of events through which some causal effects are transmitted from the first process to the last. SLAT also contains an implementation using information flow model checking. Sarna-Sota et al. [10] used the SLAT information flow model to implement a framework for analyzing configuration policies in SELinux; it is called PAL (Policy Analysis using Logic Programming). PAL creates a logic program based on an SELinux policy to make it possible to run queries to analyze the policy. APOL [35] is a tool developed by Tresys Technology to analyze SELinux configuration policies. Its main features include forward and reverse domain transition analysis, direct and transitive information flow analysis, relabel analysis and type relationship analysis based on user request.

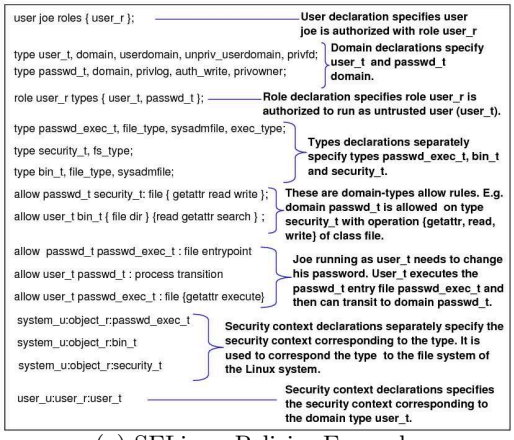
SLAT, PAL and APOL tools require the administrator to be well versed in SELinux policies to generate meaningful queries against the policy base to ultimately extract meaningful information. Furthermore, some of these tools provide graph based visualization mechanisms to aid in policy analysis, however these mechanisms are not intuitive to the average administrator. Our policy analysis framework provides visualization techniques that are usable and intuitive to the average administrator. Two works in information visualization are related to our works: Semantic Substrates and Adjacency Matrices.

Semantic Substrates [3] is a visualization method that generates graph layouts that are based on user-defined semantic substrates, which 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. Of course semantic substrates are effective only if there is some categorical attribute or if a numerical attribute can be binned to form categories. Although there are limitations in the implementation, but the utility of semantic substrates shows apparent for coping with the complexity of large numbers of nodes and links. Also, as the node-link based diagram, semantic substrates method shows strong advantages in small graphs. However, in many situations, the graph maybe very big and dense. Adjacency matrices [20] are widely used in graph visualization because they can effectively display a big and dense graph through interpreting the structural information buried in a matrix view of a graph. Although adjacency matrices can be used to visualize both directed and undirected graphs, it is argued as bad ability in finding the path from one node to another node in the directed graph.

3. PRELIMINARIES

3.1 SELinux Overview

Security-Enhanced Linux [22] implements the mandatory access control (MAC) based policies, which are . The MAC mechanisms are implemented through the Type-Enforcement model, in which domains are used to label processes, and types are used to label files and other resources. The policy rule set specifies how domains can access different types. For example, a policy defines a domain `passwd_t` and assigns it to processes running a specific set of executables used for password. The policy would allow `passwd_t` domain to operate on resources with type `security_t`. The operation is identified by two pieces of information: a class (e.g., file, directory, process, socket) and a permission (e.g., read, unlink, signal, sendto). SELinux defines 28 classes and 120 permissions. For the sake of simplicity SELinux uses type to interchangeably describe both domain and type. In addition to Type-Enforcement, SELinux also provides a role based access control (RBAC) model [22]. A user is assigned to a role which is an abstraction designed to make policy rules more concise. Policy rules are introduced to state the user to role assignments and the role to permission assignments. The set of permissions associated with a role are specified using 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 1(a) shows an example SELinux policy showing the type, domain and role declarations, a user `jdoe` operating in the untrusted domain `user_t`, the domain-type allow rules, and the security context declarations.



(a) SELinux Policies Example

SELinux Domains Classification		
Domain Class.	**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
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 config. 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

(b) SELinux Type Characteristics

Figure 1: SELinux Example Policy and Classifications

3.1.1 SELinux Type Characteristics

The SELinux types are classified into different categorizations corresponding to the functions performed by processes and the operations performed on the different objects [33]. The domain and type classifications are defined as follows:

- **Domain Classification:** According to the SELinux policy configuration from NSA [33], 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 Figure 1(b).

3.1.2 SELinux Policy Security Goals

Loscocco et al. [2] outlined six critical security goals to be achieved by SELinux security policies, these goals are summarized as follows: (G1) Limiting raw access to data, (G2) Protecting kernel integrity, (G3) Protecting system file integrity, (G4) Confining privileged process, (G5) Separating processes, and (G6) Protecting the administrator domain. Goals G2, G3 and G6 are focused on integrity protection of resources that include the boot files, proc files and security policy related objects. Goal G1 protects both the integrity and confidentiality of the system device resources, for example, the *write* operation to the fixed disk devices is restricted to the *fsck* labeled programs for file system consistency checking. Goals G4 and G5 target the implementation of the principle of the least privilege by restricting access to certain domains [29]. For example, a mail server process should only have access to certain resources such as the mail spool file. These goals are implemented in SELinux policies by limiting access using the allow/deny rules targeting specific domains and types. *Goal related rules* can be identified by checking the policy allow/deny rules and the affected resources. For example, the policies related to G1, G2, and G3 can be identified by locating rules affecting raw data, kernel files and systems files respectively. Later, we use the classification of goal related policies to analyze the security policies against these security goals and locate security violations.

3.2 Trusted Computing Base (TCB)

The early understanding of trust was that hardware and software that had to be trusted was generally equated to the operating system and the supporting hardware. Then the concept of the reference monitor was introduced in system architectures to validate all references by programs against information security policies [8]. This consequently lead to the introduction of the Trusted Computing Base (TCB), which is defined as the part of the system that is responsible for enforcing the information security policies of the system [1]. TCB includes not only the reference validation mechanism, but also encompasses all other functionality that directly or indirectly affects the correct operation of the reference validation mechanism. Using the operating system as the example, the TCB of the system includes the object management and access control functions. The object management function is responsible for creating objects, processing requests and the access control contains both the rules and the security attributes that support the access control decision-making process. TCB partitions the hardware and software into two parts: the part inside the Trusted Computing Base is referred to as trusted (TCB) and the part outside the Trusting Computing Base is referred to as untrusted (N-TCB).

3.3 Information Flow Model

In an operating system, the operations between subjects and objects can be classified as *write_like* or *read_like* [10] and the operations between subjects can be expressed as *calls*. If a subject s_1 can write to an object o ($write(s_1, o)$), which can be read by another subject s_2 ($read(o, s_2)$), we say there is a *flow transition* from subject s_1 to subject s_2

($flowtrans(s_1, s_2)$). The subject to subject calling relationship is considered as a flow transition from subject s_1 to s_2 if s_1 can call s_2.

DEFINITION 1. *The Flow Transition $flowtrans(s_i, s_j)$ specifies that information flows from subject s_i to subject s_j. We say there is a flow transition from subject s_i to subject s_j if:* $(\exists o \in O : write(s_i, o) \wedge read(s_j, o)) \vee call(s_1, s_2)$.

The flow transition describes the direct information flow between subjects. Suppose there is a sequence of flow transitions in which $flowtrans(s_{i-1}, s_i)$ for subjects $i = 1, \ldots, n$, then without loss of generality there is an information flow path from subject s_0 to subject s_n.

DEFINITION 2. *The Information Flow Path $flowpath(s_0, s_n)$, specifies sequence of flow transitions from subject s_i to subject s_j. Assume there is a flow transition $flowtrans(s_{i-1}, s_i)$ for $i = 1, \ldots, n$ then $flowpath(s_0, s_n)$ is represented as:* $\bigwedge_{i=1}^{n} flowtrans(s_{i-1}, s_i)$.

Traditional models describing information flow related to integrity and confidentiality include Lattice [9], Bella- LaPadula [5], Biba [6] and Clark-Wilson [7]. The Biba and Clark-Wilson are related to integrity, the Bella-LaPadula model is concerned with confidentiality and Lattice is the combination of the Biba and the Bella-LaPadula models. The Biba integrity property is fulfilled if a high integrity process cannot read lower-integrity data, execute lower-integrity programs, or otherwise obtain lower-integrity data in any other manner. Clark-Wilson provides a different view of dependence, where low integrity data can flow to high integrity only through a particular information flow channel referred to by *Filter*. Since in there is no *Filter* mechanism in SELinux, we later adopt Biba or BLP models in checking information flow paths and finding possible policy violations against security goals.

4. FRAMEWORK OVERVIEW

In this section, we present our framework for enabling policy visualization with emphasis on SELinux policies. Our framework is divided in the following major modules:

- **Policy Files:** The policy files include the *security policy*, *role permission mappings*, *TCB and N-TCB definitions* and the *roal related rules labeling*. These provide information related to policy statements, mappings of the operations between the subjects and objects, the initial TCB/N-TCB classification, and types targeted by the different security goals (G1 to G6).
- **Policy Parser:** This module involves the parsing of policies and the mapping of policies into goals and TCB definitions. This information is used to compile the policy graph, which is discussed in the next section.
- **User Input:** This module is composed of the *overview module* which provides a general view of the policy graph, the *content view module* which is used for viewing the policy statements, the *detailed view module* which is used for exploring detailed portions of the policy graph, the *policy analysis module* provides the GUI used for analyzing and finding the policy violations.
- **Query:** This module enables the user to specify, translate and execute queries against the policy graph. The *query writer* provides the graphical tools used by the user to specify the query, this query is then translated into path queries on the policy graph by the *query translator* and finally the *query executor* applies path finding algorithms on the policy graph to execute the query.
- **Policy Visualization:** This module provides the visualization capabilities. It provides several graph visualization layouts for the query computed policy graphs such as the *semantic substrates* and the *adjacency matrix*. It also enables the user to perform several operations on the visual layouts such as *zoom, pan, annotation, rearrangement* and *clock-wise*.

5. POLICY VISUALIZATION

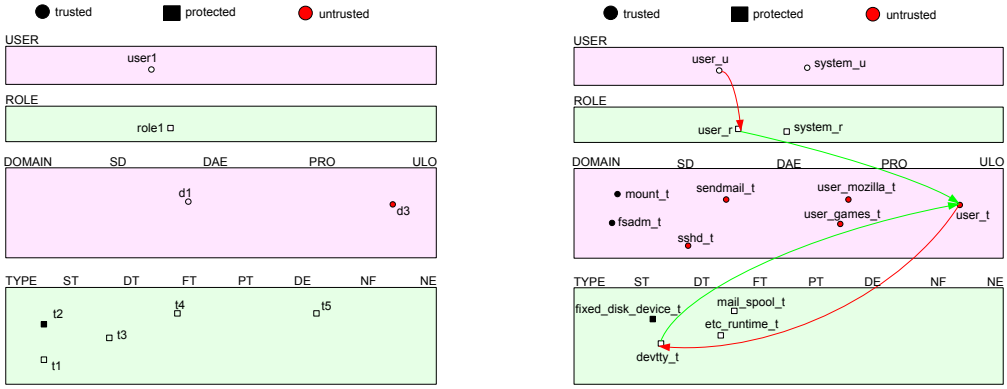
Information visualization leverages highly-developed human visual system to achieve rapid uptake of abstract information. In our framework we use information visualization techniques to visualize the policy to enable the system administrator to better understand the configured policy. In this section, first we define the policy graphs, then we present our proposed semantic substrates and adjacency matrix policy visualization techniques. A policy graph is defined as:

DEFINITION 3. *Policy Graph is a directed categorized graph $G = (V, E)$, where the set of vertices V and the set of edges E represent the types of entities and the flow transitions between them respectively.*

- *$V = V_o \cup V_s \cup V_r \cup V_u$ is the set of nodes representing different entities. V_o, V_s, V_r, and V_u is the set of nodes that represent objects, subjects, roles, and users respectively. The objects are assigned types and the subjects are assigned domains.*
- *$E = E_r \cup E_w \cup E_c$ is the set of edges describing information flow between the different vertices. Given subject vertices $v_{s_i}, v_{s_k} \in V_s$ and object vertex $V_o \in V_o$:*
  - *$(V_{s_i}, V_o) \in E_w$ if there is a $write(s_i, o)$.*
  - *$(V_o, V_{s_k}) \in E_r$ if there is a $read(s_k, o)$.*
  - *$(V_{s_i}, V_{s_k}) \in E_c$ if there is a $call(s_i, s_k)$.*

5.1 Semantic Substrates

Several visualization studies concluded [12, 3] that humans perceive data coded in spatial dimensions far more easily than those coded in non-spatial ones. Building on these results, we propose the use of semantic substrates based on node attributes to layout nodes in non-overlapping screen regions. We also make use of non-spacial cues, such as color or shape to emphasis certain nodes or group of nodes. An SELinux policy graph consists of mainly four node categories, namely *User*, *Role*, *Domain* and *Type*. Furthermore, domains and types can be further classified, for example *administration domain* and *user program domain*. Based on this semantic classification of nodes, the policy graph can be displayed spatially by distributing nodes into non-overlapping regions. Figure 2, shows the semantic substrate template used. The Y-axis is divided into regions, where each region contains nodes representing a certain entity. Furthermore, in each region nodes representing entities with different classification are placed in different districts on the X-axis. Different colors and shapes are used to help the identification of different nodes, for example, black circles, red circles and black squares are used to represent



(a) Policy Display Template (b) Example Node-Link Path

Figure 2: Semantic Substrate Template and Example

(a) Policy Display Template (b) Example Adj-Matrix Path

Figure 3: Adjacency Matrix Template and Example

trusted domains, untrusted domains and protected types respectively. Based on the policy graph definition, we distinguish the transitions between different nodes by assigning different colors to the different transition classes. For example, the *user* to *role* assignment is represented by a red arc, and similarly the *role* to *domain*, *domain* to *type* and *type* to *domain* are assigned different colors. One advantage of semantic substrates is that the administrator can easily visualize links that cross from one category (region) to another region [3].

5.2 Adjacency Matrix

The semantic substrates is a very good choice for path finding given 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 which is more compact and is free of visual clutter [32, 20]. We further enhance the path visualization capabilities of the adjacency matrices approach by adding direction characteristics. We also develop a direction based approach that enables the administrator to intuitively trace the visualized paths.

Figure 3(a) shows the 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 3(b) shows the visualization of path $P = \{d_2, t_2, d_4, t_1\}$. The series of arcs carry all information of the original path. In our template the types and domains are arranged on both the X-axis and the Y-axis. Furthermore, the grid is divided into four quadrants:

- **Quadrant 1:** This is the *write quadrant*, a slot (d_i, t_j) signifies that domain d_i can write to type t_j.
- **Quadrant 2:** Slot (d_i, d_j) signifies that domain d_i can call domain d_j.
- **Quadrant 3:** This is the *read quadrant*. A slot (t_i, d_j) signifies that type t_i can read by domain d_j.
- **Quadrant 4:** Slot (t_i, t_i) is used to enable transition.

For example, a path $P = \{d_2, t_2, d_4, t_1\}$ represents information flow $write(d_2, t_2)$, $read(d_4, t_2)$ and $write(d_4, t_1)$. In our proposed adjacency matrix template this requires the path to visit the *write quadrant* then the *read quadrant*. Therefore, information flow paths will 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, non-trusted and goal protected entities in the adjacency matrix.

6. SECURITY POLICY QUERYING

Users have difficulty writing or formulating a query [34]. The idea of the visual query formulation is to help system administrators to specify precise queries on the policy base using an interactive visual querying technique. Using an ap-

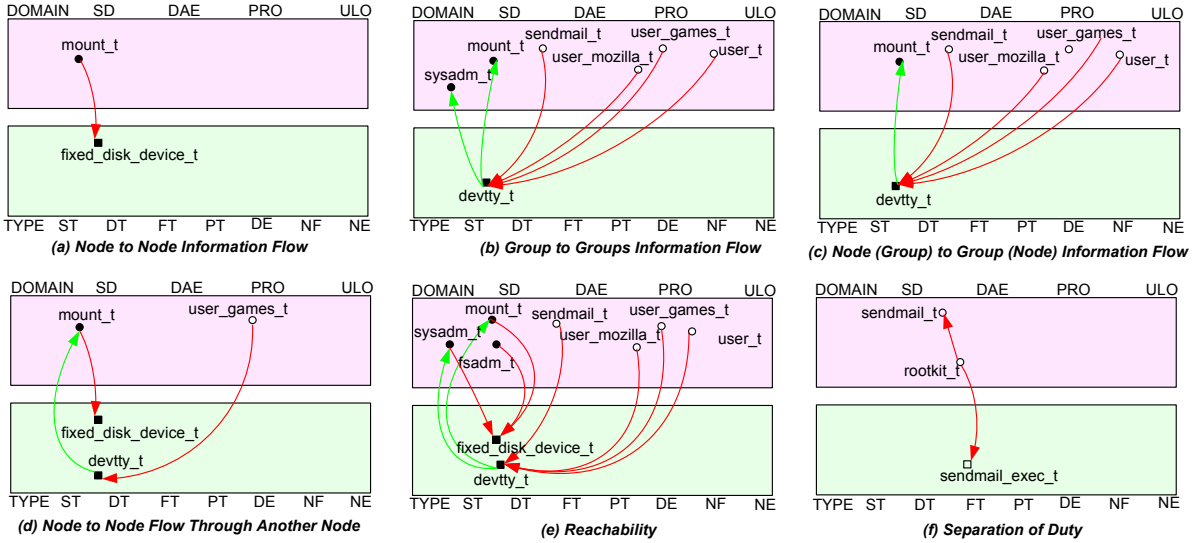


Figure 4: Example Query Results

proach similar to the Query-by-Example (QBE) for querying relational data [26, 27], in having a graphical user interface that allows users to write queries by creating example tables. 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 classification and operators were designed to provide functionalities adopted by the previous policy analysis mechanisms [35, 31]. In general, there are two classes of queries:

Q1. Identify policy integrity violations based on information flow against security goals.

Q2. Identify other policy violations like separation of duty and incompleteness.

6.1 Query Classification

Integrity checking is based on performing reachability analysis on the policy graph. For example, PAL [31] focused on finding information flow paths from N-TCB to TCB. In addition to the TCB and N-TCB classification, our framework provides a goal related policy classification, which enables querying for information flow paths affecting resources protected by certain goals. In what follows we provide set of basic query classes that are supported by our framework to enable the administrator to query the policy base. A node represents a user, role, type or domain, and a group represents set of nodes, available groups are TCB, N-TCB, goal related nodes, and user defined groups.

C1. *Node to Node information flow paths.* This enables the querying for information flow from a specific domain to a specific type. The example in Figure 4(a) shows the query result in the form of the information flow path from domain *mount_t* to type *fixed_disk_device_t*.

C2. *Group to Groups information flow paths.* This enables the querying for information flow from N-TCB to TCB, or from a N-TCB to a set of goal related domains or types. The example in Figure 4(b) shows the paths query result from N-TCB to TCB.

C3. *Node (Group) to Group (Node) information flow paths.* This enables the querying for information flow from one domain to the goal protected types, or from the N-TCB to a certain domain. The example in Figure 4(c) shows the result of finding information flow paths from all N-TCB to the *mount_t* domain.

C4. *Node to Node information flow paths through another Node.* Finds information flow from one type to another type through a certain type, where types can be domains or types. The example in Figure 4(d) shows the result of finding information flow path from domain *user_games_t* to type *fixed_disk_device_t* through type *devtty_t*.

C5. *Reachability.* Finds all possible information flows from or to a certain type. For example, find all information flows to *fixed_disk_device_t*, or the information flows from *user_t*. The example in Figure 4(e) shows the result of finding the information flow paths flowing to *fixed_disk_device_t*.

C6. *Separation of Duty (SoD).* Checks constraints on authorizations to types. For example in the context of SELinux, the separation of duty can be interpreted as separation of the domains allowed to modify(e.g., write or create) executable files from the domains allowed to execute those executables. In PAL [31], these queries are restricted to the direct access. We consider direct and indirect by examining the information flow path. In the SELinux example policies, to test this query we introduce policies that enable the *rootkit_t* domain to have write access on *sendmail_exec_t* type and transition operation on *sendmail_t*. By querying the policy graph we are able to locate this SoD violation as depicted in Figure 4(f).

6.2 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 compared to the current policy the current policy analysis frameworks [35, 31] which are based on scripting. Figure 5 summarizes the basic visual components.

- *Element Nodes (E-Nodes)* are shaped as labeled circles;

their label represents the attributes of the element. e.g. using SELinux policy as the example, the element nodes include *USER, ROLE, DOMAIN, TYPE, TCB, NON-TCB* and *Goal*. The character # is used to help the attribute specification. For example, Goal# can be customized to be Goal1, Goal 2 etc.

- *Operator Edge (O-Edge)* is represented as the curve that connect 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 or a set. When the policy administrator draws the query, this value 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 required path properties. 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, any path or the paths that can be found in the time limitation. The value "*" denotes all paths.

Figure 5(e) shows example composed queries that specify how to query policy graph nodes relationship like have, information flow path.

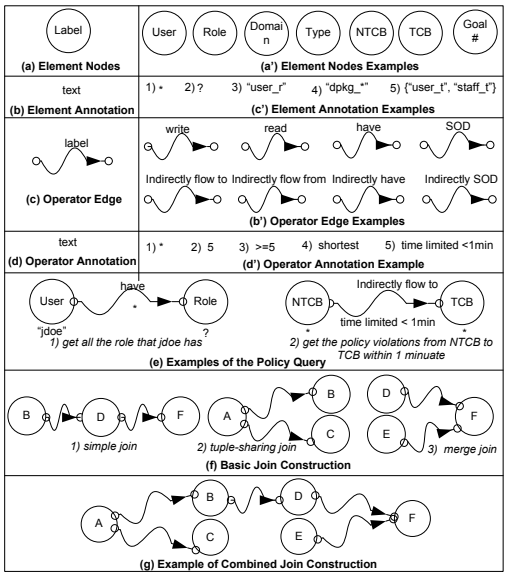


Figure 5: Query Construction

6.2.1 Join Query Construction

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

- *Simple Join* specifies that a set of E-Nodes are 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. An example join query is shown in Figures 5(f).
- *Tuple-sharing Join*specifies that two or more E-Nodes are connected out 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 sort-merge 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 $o_j(n_j, n_k)$, then we say there is a merge join.

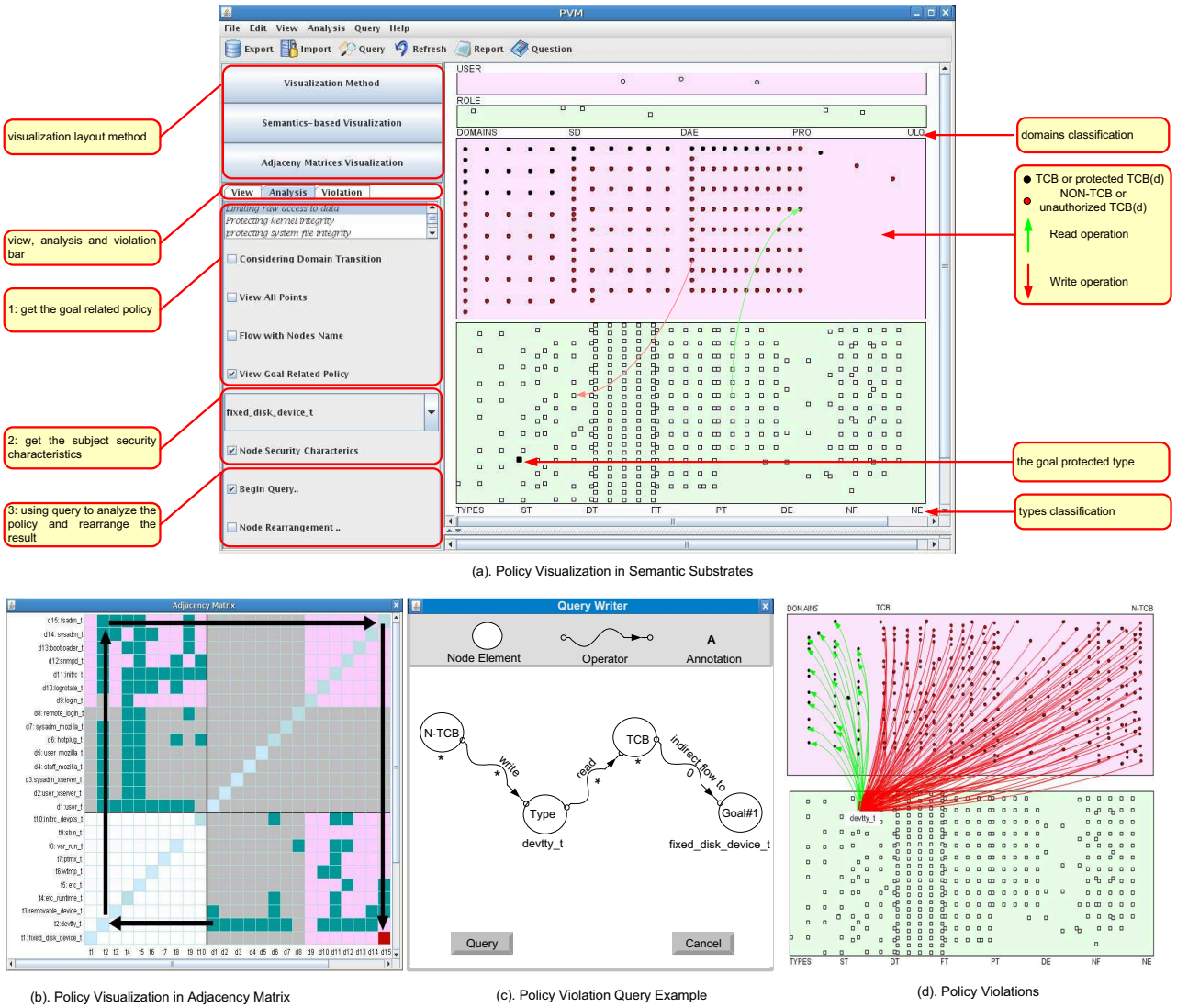
6.3 Query Execution

Based on the definitions of the join query construction, the identification of the different join format can facilitate the query execution. The paths computed during the query executions are based on the OE-Annotations associated with operator edges which include the shortest path, any path or the paths found given a execution time limit. The query execution makes use of the shared nodes between group nodes. For example, in the tuple-sharing join (shown in Figure 5), suppose A is $NTCB$, B is TCB, C is $fsadm\_t$ and the O-Edges having 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, the query will evaluate paths from D to F then the paths from E - $(E \cap D)$ to F.

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 6: Query Execution Algorithm

Referring to the algorithm in Figure 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 nodes connected to it identified by 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



(a). Policy Visualization in Semantic Substrates

(b). Policy Visualization in Adjacency Matrix

(c). Policy Violation Query Example

(d). Policy Violations

Figure 7: Policy Visualization

$n_a - n_b.getMergeNodes(n_b, e)$, the merge join nodes information are stored in the n_b attribute and can be retrieved using $n_b.getMergeNodes(n_b, e)$. On the other hand, if n_a and n_b are part of tuple-sharing, the duplication of n_b using $n_b - n_a.getTupleNode(n_a, e)$, where the tuple-sharing nodes information is maintained in the n_a attribute and can be retrieved using $n_a.getTupleNode(n_a, e)$. After the information duplication is removed, the query from n_a to n_b with operator e is executed. Finally, the executed queries are added to result graph by adding the nodes and edge information into n_a and n_b respectively using $n_a.addTupleNode(n_b)$ and $n_b.addMergeNode(n_a)$.

7. SELINUX CASE STUDY

In this section we discuss the implementation details of our proposed framework, we give design snapshots of our policy visualization analysis tool (PVA) and we discuss how the tool is used to identify policy violations in SELinux policies.

7.1 Policy Visualization Analysis Tool (PVA)

The PVA tool is presented to the user via a self explanatory graphical user interface. To enhance the cognition and understanding of the policy information, we provide implementations of both the 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.

Our implementation is based on the Java JDK1.6 and supporting libraries. The graph drawing modules were based on our extensions to the open source graphing package Piccollo [4]. Our parsing tool is based on the policy structure adopted by the APOL [35] tool. In this case study the SELinux policy binary file `policy.19` was used. Figure 7(a) shows a snapshot of the our tool. The policy administrator can import, analyze, query and modify the policy through the menu. 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 e.g. viewing the whole policy graph through zoom in, zoom

Table 1: Policy Violation Examples

Example Policy Violations			
Subjects	Type:Class	Subject	Resolution
200	network	fsadm_t	Filter
rhgb_t	mnt_t:dir	fsadm_t	Modify
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
3	scsi_generic_device_t:chr_file	fsadm_t	Modify
200	devlog_t:sock_file	fsadm_t	Ignore

out etc. 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 a security violations. Furthermore, in this tab the policy administrator can directly modify the policy in using the text editor or by directly editing the policy graph. In the main window the policy graph, query results, goal related policy graphs and the policy violation graph can displayed.

7.2 Policy Graph

The main window in Figure 7(a) shows the visualized SELinux policy based on semantic substrate design proposed in Section 5. The policy is composed of 308 domains, 1092 types and 31604 links. The Y-axis is divided into four regions including USERS, ROLES, DOMAINS and TYPES. The X-axis is labeled using the domain and type classifications discussed in Section 5. 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), NE (network types). To help the policy administrator to easily identify the different regions, the elements in non-neighboring regions are represented 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 7(b), shows the adjacency matrix-based policy visualization method, which was compiled by selecting a subset of the nodes in the semantic substrates overlay.

7.3 Policy Query and Violation Detection

Figure 7(c) shows the graphical query interface and a query designed to discover the paths N-TCB to resources related to goal G1 (limiting raw access to data) such as `fixed_disk_device_t` through specific type `devtty_t` and TCB resources. Starting from left to right (Figure 7(c)), the first node selects the N-TCB resources and finds the paths to type `devtty_t`, then finds the paths from `devtty_t` to the TCB resources. Finally, the query builds the paths from the TCB resources to the goal G1 `fixed_disk_device_t` device. Figure 7(d) shows the identified policy violations by this query. Note, that the display divides the TCB and N-TCB to provide a better understanding to the system administrator. Running the visualization tool on 1.4GHz Intel Pentium CPU with 512Mbytes of memory, the query loading and parsing takes 15s, and the query execution and display 21s. Another example query that investigates information flow paths from N-TCB to `fsadm_t` (TCB) without the constraint of passing through a specific intermediate node executes and displays in 88s. Table 1 shows identified policy violations caused by information flow from NON-TCB to TCB `fsadm_t`. The query execution and display is dependent on the query type and the results size.

8. CONCLUSION

In this paper, we have proposed a visualization-based policy analysis framework to analyze the security policies. We have provided both semantic substrates and adjacency matrix approaches for policy visualization. We presented our visualization-based query mechanism that enables the administrator to query the policy base by simply connecting query components similar to the query by example approach. Our main methodology is to use visualization-based queries to identify the possible policy violations. We have developed a Policy Visualization Analysis (PVA) tool to implement our framework. Additionally, we discussed how to use our framework to analyze SELinux policies and the results confirmed the feasibility and applicability of our methodology. We believe that this is the first attempt to formulate a general visualization-based policy analysis framework and the first attempt to use the visualization-based query to analyze the policies.

Our current future work includes developing usability studies to evaluate the usability of our policy visualization framework. For example, from using our tool we noticed that building SoD queries is not intuitive as it requires the user to clearly understand to constraints behind the SoD rules. Furthermore, we plan on investigating node and link reordering mechanisms that minimize the link crossings and entanglement to provide more appealing policy visualizations. Other possible areas of future work include applying more visualization mechanism into our work to improve the effect of the visualization and optimize the algorithm of policy query. Also, the application of our framework for visualizing and analyzing the XACML policy will be investigated in the near future.

9. REFERENCES

[1] *System Management Concepts: Operating System and Devices*. IBM Corporation, 1 Ed., 1999.

[2] P. Loscocco and S. Smalley. Meeting critical security objectives with security-enhanced linux. In *Proceedings of the Ottawa Linux Symposium*, 2001.

[3] A. Aris. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):733–740, 2006.

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

[5] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. *MITRE Corporation*, 1976.

[6] K. J. Biba. Integrity considerations for secure computer systems. *MTR-3153, MITRE Corporation*, April 1977.

[7] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1987.

[8] M. Abrams and M. Joyce. Trusted computing update. *Computers & Security*, 14(1):57–68, 1995.

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

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

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

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

[13] 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.

[14] 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.

[15] T. Itoh, H. Takakura, A. Sawada, and K. Koyamada. Hierarchical visualization of network intrusion detection data. *IEEE Computer Graphics and Applications*, 26(2):40–47, 2006.

[16] T. Jaeger, R. Sailer, and U. 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.

[17] 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*, pages 5–5, Berkeley, CA, USA, 2003.

[18] T. Jaeger, R. Sailer, and X. Zhang. Resolving constraint conflicts. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 105–114, New York, NY, USA, 2004.

[19] T. Jaeger, X. Zhang, and A. Edwards. Policy management using access control spaces. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):327–364, 2003.

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

[21] 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.

[22] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the linux operating system. In *USENIX Annual Technical Conference, FREENIX Track*, pages 29–42, Berkeley, CA, USA, 2001.

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

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

[25] S. Noel and S. Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 109–118, New York, NY, USA, 2004.

[26] 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.

[27] 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.

[28] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of the 11th ACM conference on computer and communications security*, pages 308–317, New York, NY, USA, 2004.

[29] J. Saltzer and M. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sept 1975.

[30] B. S.-Starosta and S. 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.

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

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

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

[34] 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.

[35] Tresys Technology. Apol. *Available from http://www.tresys.com/selinux/*.

[36] R. Su Thompson, E. M. Rantanen, W. Yurcik, and B. P. Bailey. Command line or pretty lines?: comparing textual and visual interfaces for intrusion detection. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 1205, New York, NY, USA, 2007.

[37] T. Tran, E. S. Al-Shaer, and R. Boutaba. Policyvis: Firewall security policy visualization and inspection. In *LISA*, pages 1–16, 2007.

[38] D. Yao, M. Shin, R. Tamassia, and W. H. Winsborough. Visualization of automated trust negotiation. In *VizSEC 05: IEEE Workshop on Visualization for Computer Security*, Oct 2005.

[39] X. Yin, W. Yurcik, M. Treaster, Y. Li, and K. Lakkaraju. Visflowconnect: netflow visualizations of link relationships for security situational awareness. In *Proceedings of the ACM workshop on Visualization and data mining for computer security*, pages 26–34, New York, NY, USA, 2004.

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

[41] W. Yurcik. Tool update: visflowconnect-ip with advanced filtering from usability testing. In *Proceedings of the 3rd international workshop on Visualization for computer security*, pages 63–64, New York, NY, USA, 2006.