Int. J. Inf. Secur. (2013) 12:155–171
DOI 10.1007/s10207-012-0180-7

REGULAR CONTRIBUTION

Visualization-based policy analysis for SELinux: framework and user study

Wenjuan Xu · Mohamed Shehab · Gail-Joon Ahn

Published online: 8 November 2012

Abstract In this paper, we propose a visualization-based policy analysis framework that enables system administrators to query and visualize security policies and to easily identify the policy violations, especially focused on SELinux. Furthermore, we propose a visual query language for expressing policy queries in a visual form. Our framework provides 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 supports the functionalities discussed in our framework. In addition, we discuss our study on usability of our tool with evaluation criteria and experimental results.

Keywords Policy analysis · Visualization-based · SELinux

1 Introduction

In computing systems, security policies are specified to meet security goals such as access to protected resources, information flow to and from protected resources, and 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 large policy sizes, for example, the Security-Enhanced Linux (SELinux) policy includes over 30,000 statements [18]. Access control systems can become significantly ineffective if the implemented policies are not representative of targeted security goals. Simple policy misconfigurations might allow an unprivileged process α to write to some resource that can be read by a privileged process β, causing information flow from α to β 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 [7,11–14,27,28,36] 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. Furthermore, such policy analyses would help locate policy violations.The output of policy analysis tools may list possible violations, which does not necessarily give the system administrator a clear view on how the violation was originated and how it might propagate in the systems. Information visualization [8] 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 understand and present data related to network attacks [20,38–40], intrusion detection [5,10,22,34], firewall policies [16,21,35], and trust negotiations [37]. 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.

W. Xu
Frostburg State University, 101 Braddock Rd.,
Frostburg, MD 21532, USA
e-mail: wxu@frostburg.edu

M. Shehab
UNC Charlotte, 9201 University City Boulevard,
Charlotte, NC 22087, USA
e-mail: mshehab@uncc.edu

G.-J. Ahn (✉)
Arizona State University, 660 South Mill Avenue,
Tempe, AZ 85287, USA
e-mail: gahn@asu.edu

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* [2,6] and *adjacency matrix* [15,29] mechanisms to generate policy layouts for representing policy portions. In the semantic substrates mechanism, the nodes and links expressing policy statements are arranged based on semantic classifications, which provide 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 objects. Providing simple and descriptive policy graph layouts enables the policy administrator to easily 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 [24,25]. Using a graphical query platform enables the average administrator to easily probe a policy for identifying violations by specifying graphical queries, without the need to write any script or learn a new query language. Also, our policy visualization framework is realized as a policy visualization analysis (PVA) tool and we attempt to visualize and query SELinux policies. In addition, we conduct a user study with two aspects: comparison between PVA and an existing tool, and comparison between semantic substrates approach and adjacency matrix approach. In this study, we examine the usability of our framework with participants who browse and analyze security policies with tools and two visualization methods based on the designed instruction. The results are collected and analyzed with paired samples t-tests [23] based on the participants' feedback.

The rest of the paper is organized as follows. Section 2 provides an overview of SELinux, trusted computing base and information flow models. In Sect. 3, we introduce our visualization-based policy analysis framework and policy visualization approaches. The policy query classification and query execution are presented in Sect. 4. Our policy visualization tool, PVA is presented in Sect. 5 followed by the evaluation through user study in Sect. 6. Section 7 describes the related work. Section 8 concludes the paper along with the future work.

2 Preliminaries

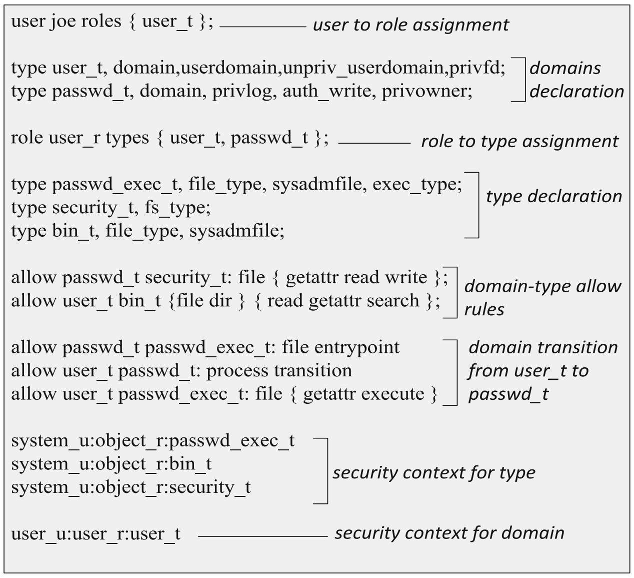
2.1 SELinux overview

Security-Enhanced Linux [18] implements mandatory access control (MAC)-based policies. 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 a password. The policy would also allow the `passwd_t` domain to operate on resources with a type `security_t`. The operation is identified by two pieces of information: a class (e.g., file, directory, process, and socket) and a permission (e.g., read, unlink, signal, and sendto). SELinux defines 28 classes and 120 permissions. For the sake of simplicity, SELinux uses the notion of *type* to interchangeably describe both domain and type. In addition to Type Enforcement, SELinux also provides a role-based access control (RBAC) model [18]. 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 is specified using types. Correspondingly, all the resources in SELinux are labeled with a set composed of user, role and domain or type. This kind of set is called security context. 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 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.

2.1.1 SELinux type characteristics

The SELinux types are classified based on the functions performed by processes and the operations performed on the different objects [30]. The domain and type classifications are defined as follows:

– **Domain Classification:** According to the SELinux policy configuration from NSA [30], 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.



```
user joe roles { user_t };

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

role user_r types { user_t, passwd_t };

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

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

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

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

user_u:user_r:user_t
```

(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	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

Fig. 1 SELinux example policy and classifications

– **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 summarized in Fig. 1.

2.1.2 SELinux policy security goals

Loscocco et al. [19] 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 checking file system consistency. Goals G4 and G5 target the implementation of the principle of the least privilege by restricting access to certain domains [26]. For example, a mail server process should only access certain resources such as the mail spool file. These goals are implemented in SELinux policies by limiting access with the allow/deny rules targeting specific domains and types. *Goal-related rules* can be identified by checking the 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.

2.2 Trusted computing base (TCB)

The early understanding of trust perceived that hardware and software, which need to be trusted, should be generally equated to operating systems and the supporting hardware. Then, the concept of the reference monitor was introduced in system architectures to validate all access requests by programs against information security policies [1]. This consequently led to the introduction of the Trusted Computing Base, which is defined as part of a system that is responsible for enforcing security policies of the system [33]. The Trusted Computing Base not only includes the reference validation mechanism, but also encompasses all other functionalities that directly or indirectly affect the correct operation of the reference validation mechanism. Using a operating system as an example, the Trusted Computing Base of the system includes the object management and access control functions. The object management function is responsible for creating objects and processing requests. The access control function contains both rules and security attributes that support the access control decision-making process. The Trusted Computing Base 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).

2.3 Information flow model

In an operating system, the operations between subjects and objects can be classified as $write\_like$ or $read\_like$ [7] 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 a subject s_1 to a subject s_2 ($flowtrans(s_1, s_2)$). The subject to subject calling relationship is considered as a flow transition from a subject s_1 to a subject s_2 if s_1 can call s_2.

Definition 1 The Flow Transition. $flowtrans(s_i, s_j)$ specifies that information flows from a subject s_i to a subject s_j. We say there is a flow transition from a subject s_i to a 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 $flowtrans(s_{i-1}, s_i)$ for subjects $i = 1, \ldots, n$, then without loss of generality there is an information flow path from a subject s_0 to a subject s_n.

Definition 2 The Information Flow Path. $flowpath(s_0, s_n)$ specifies a sequence of flow transitions from a subject s_i to a 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 [4], Bell-LaPadula [32], and Biba [3] models. The Biba model is related to integrity, the Bell-LaPadula model is concerned with confidentiality and the Lattice-based approach is the combination of Biba and Bell-LaPadula models. The integrity property in Biba model is fulfilled if a high integrity process cannot read lower-integrity data, execute lower-integrity programs, nor otherwise obtain lower-integrity data in any other manner. In SELinux policy analysis, we later adopt Biba or Bell-LaPadula models in checking information flow paths and finding possible policy violations against security goals.

3 Framework overview: visualization-based policy analysis

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

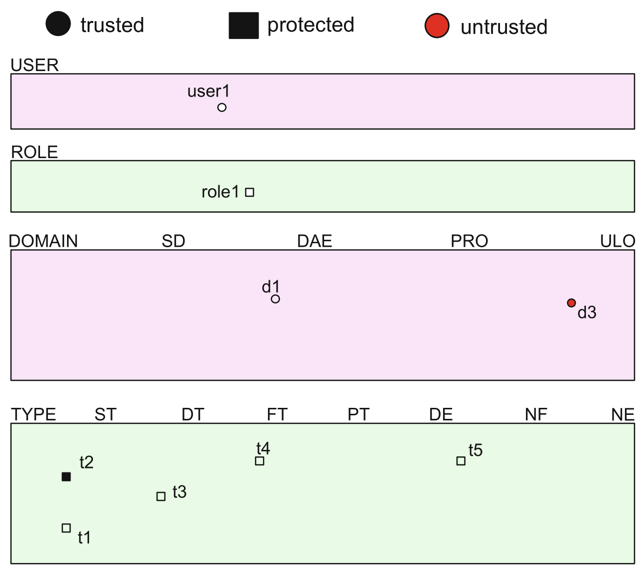
- **Policy Files:** The policy files include the *security policy*, *role permission mappings*, *TCB and N-TCB definitions* and the *goal-related rule labeling*. These provide information related to policy statements, mappings of the operations between the subjects and objects, the initial TCB/N-TCB 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 subsequent 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, and the *policy analysis module* which provides interactive interfaces 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* helps the user specify the query. The 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 on 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 *clockwise*.

3.1 Policy visualization

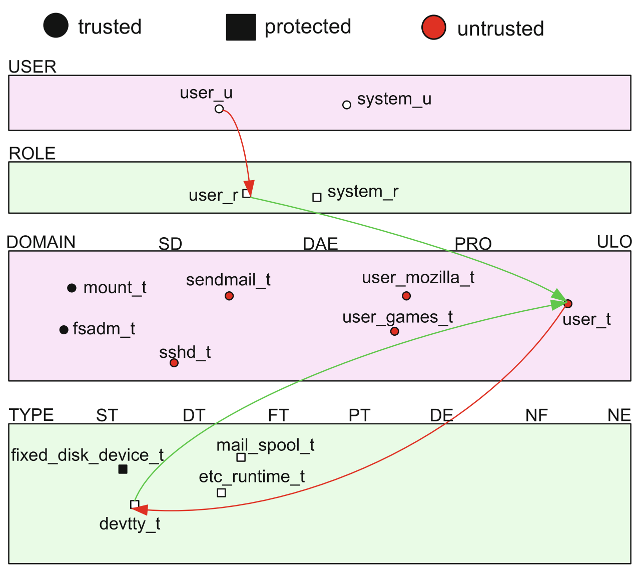
In our framework we use information visualization techniques to visualize the policy so that the system administrator is empowered to better understand the configured policy. In this section, first we define the policy graph, then we present our proposed semantic substrates and adjacency matrix policy visualization techniques. A policy graph is defined as:

Definition 3 A 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 are the set of nodes that represent objects, subjects, roles, and users, respec-



(a) Policy Display Template



(b) Example Node-Link Path

Fig. 2 Semantic substrate template and example

tively. The objects are the assigned types and the subjects are the assigned domains.

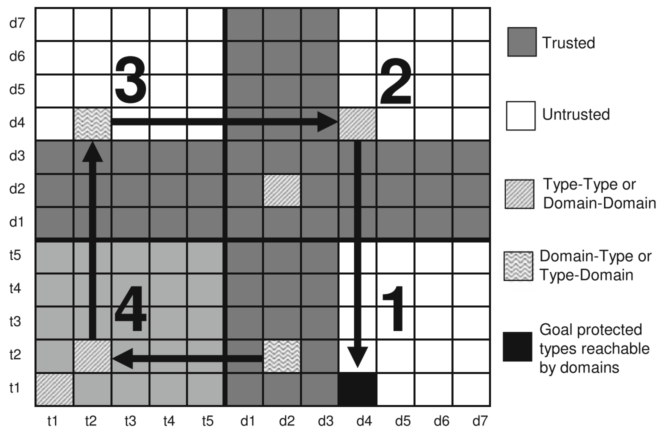
– $E = E_r \cup E_w \cup E_c$ is the set of edges describing information flow between the different vertices. Given subject vertices $v_{s_i}, v_{s_k} \in V_s$ and object vertex $V_{obj} \in V_o$:

  – $(V_{s_i}, V_{obj}) \in E_w$ if there is a $write(s_i, obj)$.
  – $(V_{obj}, V_{s_k}) \in E_r$ if there is a $read(s_k, obj)$.
  – $(V_{s_i}, V_{s_k}) \in E_c$ if there is a $call(s_i, s_k)$.

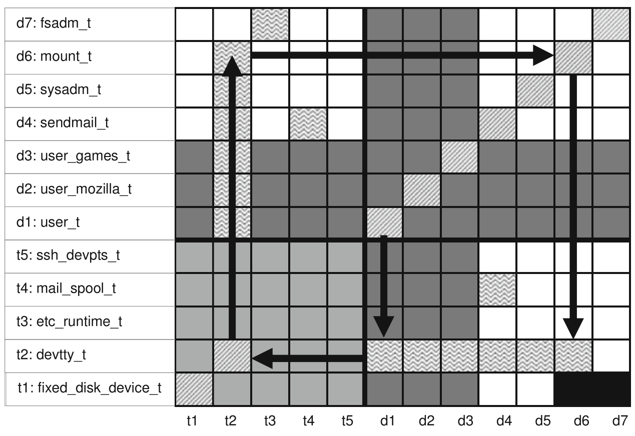
Two information visualization techniques are related to visualization work: Semantic Substrates and Adjacency Matrix. Semantic Substrates [2] is a visualization method that generates graph layouts that are based on user-defined semantic substrates, which are non-overlapping regions in which node placement 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 exist some categorical attributes or if a numerical attribute can be used to form categories. Although there are limitations in the implementation, the utility of semantic substrates apparently copes with a large number of nodes and links. Also, as the node-link-based diagram, the semantic substrates method shows strong advantages in small graphs. However, in many situations, the graph may be very big and dense. Adjacency matrix [15] is widely used in graph visualization because it can effectively display a big and dense graph by interpreting the structural information embedded in a matrix view of a graph. Although adjacency matrices can be used to visualize both directed and undirected graphs, it is argued that finding the path from one node to another node in the directed graph might not be an essential ability. In the following, we will explain how to use these techniques to visualize security policies.

3.1.1 Semantic substrates

Several visualization studies concluded [2,6] 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-spatial cues, such as color or shape to emphasize certain nodes or a 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 and examples. 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 classifications are placed in different districts on the X-axis. Different colors and shapes are used to aid the identification of nodes, for example, black circles, red circles and black squares are used to represent trusted domains, untrusted domains and protected types, respectively. Based on the policy graph definition, we distinguish the transitions between 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*


d7
d6
d5
d4
d3
d2
d1
t5
t4
t3
t2
t1
3
2
4
1
t1 t2 t3 t4 t5 d1 d2 d3 d4 d5 d6 d7
Trusted
Untrusted
Type-Type or Domain-Domain
Domain-Type or Type-Domain
Goal protected types reachable by domains

(a) Policy Display Template



(b) Example Adj-Matrix Path

Fig. 3 Adjacency matrix template and example

to *domain* are denoted with other colors. One advantage of semantic substrates is that the administrator can easily visualize links that cross from one category (region) to another region [2].

3.1.2 Adjacency matrix

The semantic substrates is a very good choice for finding a path, 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 free of visual clutter [15,29]. 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 3a shows our proposed adjacency matrix visualization template. The nodes are arranged on both the X-axis and the Y-axis. To visualize a path $P = \{v_0, v_1, \ldots, v_n\}$ in the adjacency matrix, we highlight entries (v_i, v_i) and (v_i, v_{i+1}), for $i = 0, \ldots, n-1$. We draw an arc from entries (v_i, v_i) and (v_i, v_{i+1}) for $i = 0, \ldots, n-1$, and we draw an arc from entries (v_{i-1}, v_i) and (v_i, v_{i+1}) for $i = 0, \ldots, n-1$. Figure 3b 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:

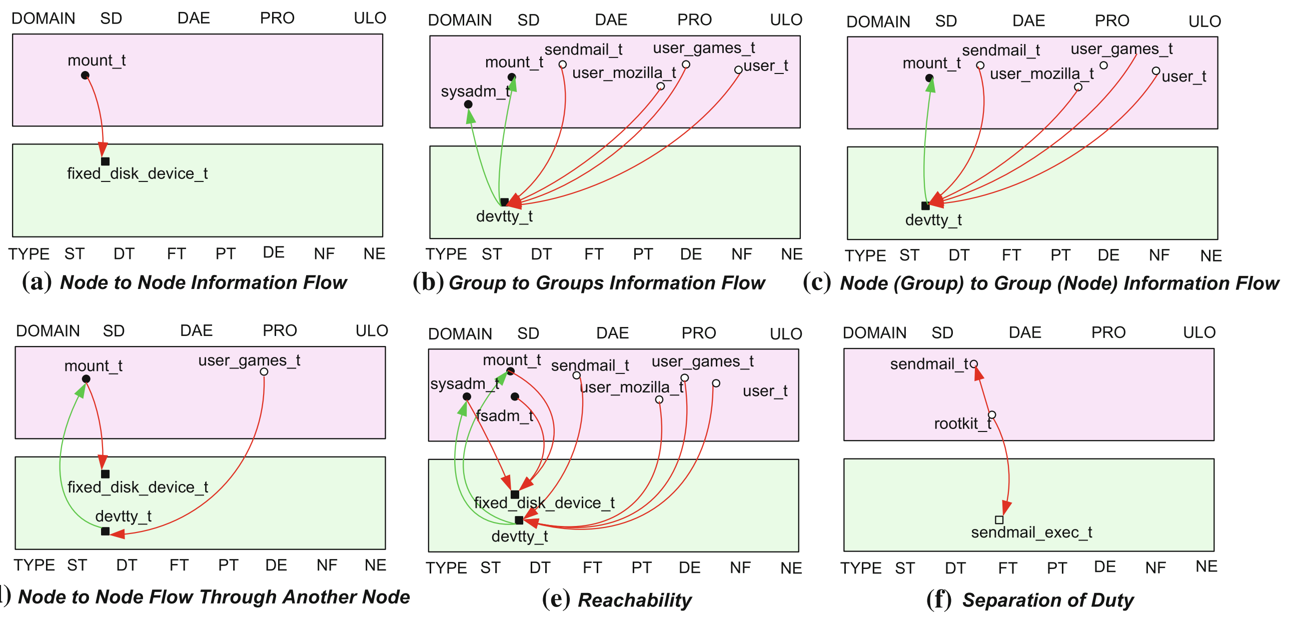
- **Quadrant 1:** This is the *write quadrant*, a slot (d_i, t_j) signifies that a domain d_i can write to a type t_j.
- **Quadrant 2:** Slot (d_i, d_j) signifies that a domain d_i can call a domain d_j.
- **Quadrant 3:** This is the *read quadrant*. A slot (t_i, d_j) signifies that a type t_i can read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

For example, a path $P = \{d_2, t_2, d_4, t_1\}$ represents information flow $\mathtt{write}(d_2, t_2)$, $\mathtt{read}(d_4, t_2)$ and $\mathtt{write}(d_4, t_1)$. 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 *clockwise* 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.

4 Security policy querying

Users may have difficulties in writing or formulating a query [31]. The idea of the visual query formulation is to help system administrators specify precise queries on the policy base using an interactive visual querying technique. Using an approach similar to the Query by Example (QBE) for querying relational data [24,25], 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 are designed to provide functionalities adopted from the previous policy analysis mechanisms [36,27]. 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.



(a) *Node to Node Information Flow* (b) *Group to Groups Information Flow* (c) *Node (Group) to Group (Node) Information Flow*

(d) *Node to Node Flow Through Another Node* (e) *Reachability* (f) *Separation of Duty*

Fig. 4 Example query results

4.1 Query classification

Integrity checking is based on performing reachability analysis on the policy graph. For example, PAL [27] 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 us to query information flow paths affecting resources protected by certain goals. Also, we provide a set of basic query classes that are supported by our framework. A node represents a user, role, type or domain, and a group represents a set of nodes. Groups include 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. Figure 4a shows the query result in the form of the information flow path from a domain *mount_t* to a 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 an N-TCB to a set of goal-related domains or types. Figure 4b shows the 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 N-TCB to a certain domain. Figure 4c 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.* It helps finding information flow from one type to another type through a certain type, where types can be domains or types. Figure 4d shows the result of finding information flow path from a domain *user_games_t* to a type *fixed_disk_device_t* through a type *devtty_t*.

C5. *Reachability.* It enables to find all possible information flows from or to a certain type. For example, it finds all information flows to *fixed_disk_device_t*, or the information flows from *user_t*. Figure 4e shows the result of finding the information flow paths to *fixed_disk_device_t*.

C6. *Separation of Duty (SoD).* This helps check constraints on authorizations to types. For example, in the context of SELinux, separation of duty can be interpreted as separation of domains allowed to modify (e.g., write or create) executable files from the domains allowed to execute those executables. In PAL [27], these queries are restricted to direct access. In the SELinux example policies, 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 Fig. 4f.

4.2 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

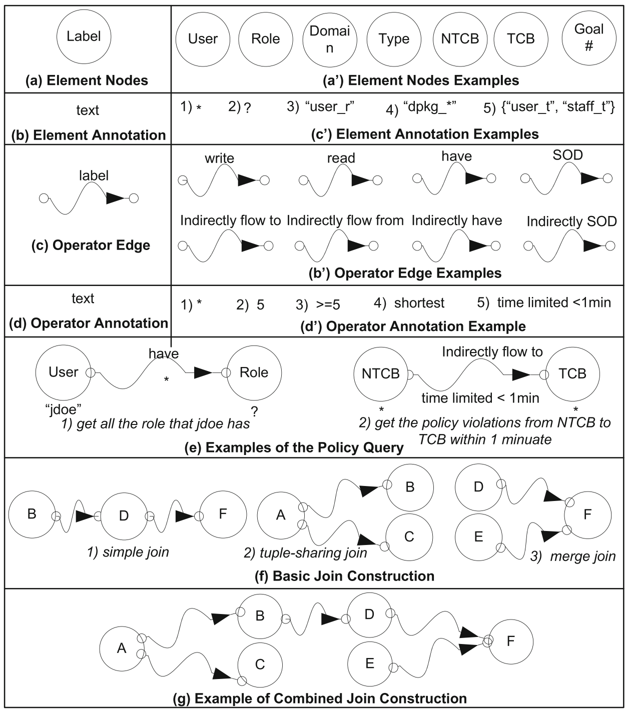

Label
(a) Element Nodes
User
Role
Domain
Type
NTCB
TCB
Goal #
(a') Element Nodes Examples
text
(b) Element Annotation
1) * 2) ? 3) "user_r" 4) "dpkg_*" 5) {"user_t", "staff_t"}
(c') Element Annotation Examples
label
(c) Operator Edge
write
read
have
SOD
Indirectly flow to
Indirectly flow from
Indirectly have
Indirectly SOD
(b') Operator Edge Examples
text
(d) Operator Annotation
1) * 2) 5 3) >=5 4) shortest 5) time limited <1min
(d') Operator Annotation Example
have
"jdoe"
1) get all the role that jdoe has
Indirectly flow to
time limited < 1min
2) get the policy violations from NTCB to TCB within 1 minuate
(e) Examples of the Policy Query
1) simple join
2) tuple-sharing join
3) merge join
(f) Basic Join Construction
(g) Example of Combined Join Construction

Fig. 5 Query construction

compared to current policy analysis frameworks [36,27] 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. For example, using SELinux policy as an 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 G1, G2, etc.
– *Operator Edge (O-Edge)* is represented as the curve that connects the element nodes to another element nodes. The label of the operator edges represents the query classification of the query. Based on the query classification, the operator edges include *write, read, call, have*, *indirect have, indirect flow to, SOD* and *indirect SOD*.
– *Element Nodes Annotation (EN-Annotation)* is to specify the element nodes value. It can be a single value 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 another node, we can specify a query to find the shortest path, all the paths, any path or the paths that can be found in the time limitation. The value "*" denotes all paths.

Figure 5e shows composed queries that specify how to query policy graph node relationships such as *have* and *flow path*.

4.2.1 Join query construction

The policy administrator can use a join query to construct more complex queries such as finding the domain that can both write and read the goal protected objects. Also, using our join query the policy administrator can accumulate several query results on a single graph. Based on the different ways of sharing E-Nodes, we define three basic joins: *Simple Join, Merge Join* and *Tuple-sharing Join*. More complex join can also be constructed by combining the three basic joins.

– *Simple Join* specifies that a set of E-Nodes is sequentially connected 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 Fig. 5f.
– *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 $o_j(n_i, n_k)$ and $o_k(n_i, n_k)$, 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 $o_i(n_i, n_k)$ and $o_j(n_j, n_k)$, then we say there is a merge join.

4.3 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. Query execution makes use of the shared nodes between group nodes. For example, in the tuple-sharing join (shown in Fig. 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*.

Referring to the algorithm in Fig. 6, the policy query execution algorithm is mainly composed of two 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 from node n_a using $getConnectEdges(n_a, G_q)$. In the second part, for each of the edges identified e in the previ-

```
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 duplications 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
```

Fig. 6 Query execution algorithm

ous step, the algorithm identifies the nodes connected to the edge identified by n_b which are retrieved by the function $findConnectNode(e, n_a, G_q)$. Since merge query and tuple-sharing are built based on the shared E-nodes, there will be duplicated nodes retrieved from the function $findConnectNode(e, n_a, G_q)$. If n_a and n_b are part of the merge query, the duplicated nodes are removed from n_a by using the expression $n_a - n_b.getMergeNodes(n_b,e)$, 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 information of tuple-sharing nodes 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 an operator e is executed. Finally, the executed queries are leveraged by adding the nodes and edge information into n_a and n_b respectively using $n_a.addTupleNode(n_b)$ and $n_b.addMergeNode(n_a)$.

5 Case study with SELinux policies

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

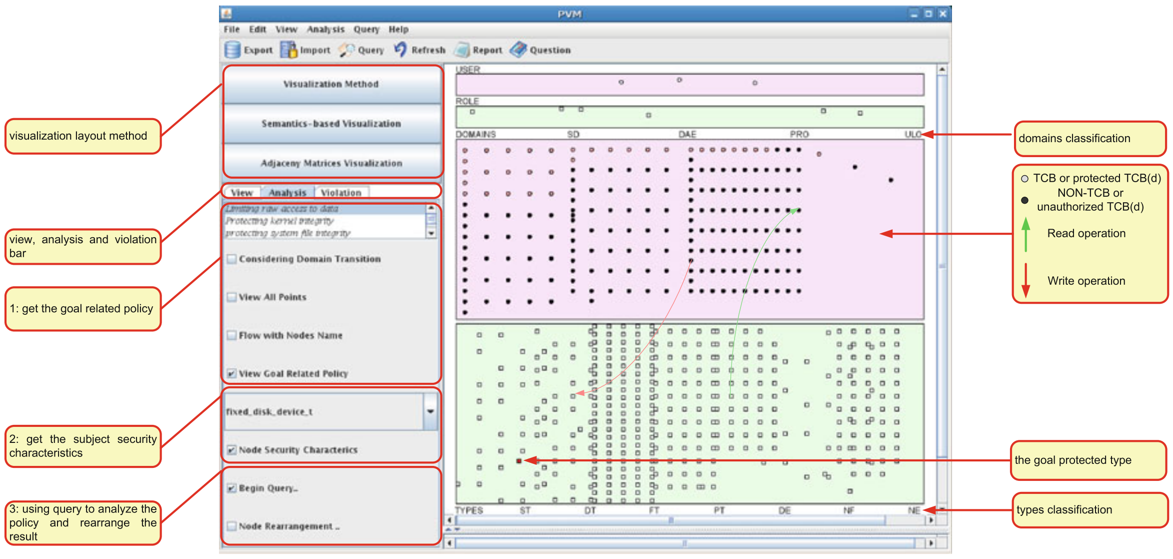
5.1 Policy visualization analysis tool (PVA)

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- 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.

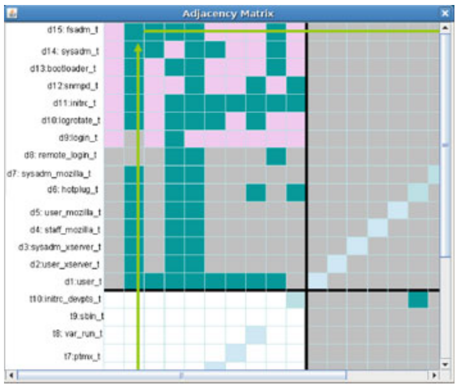
Our implementation is based on the Java JDK1.6 and supporting libraries. The graph drawing modules were based on our extensions to the open source package Piccollo [9]. Our parsing tool is based on the policy structure adopted by the APOL [36] tool. In this case study the SELinux policy binary file `policy.19` was used. Figure 7a shows a snapshot of 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 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 policy violations 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 by using the text editor or directly editing the policy graph. In the main window, the policy graph, query results, goal-related policy graphs and the policy violation graph are displayed.

5.2 Policy graph

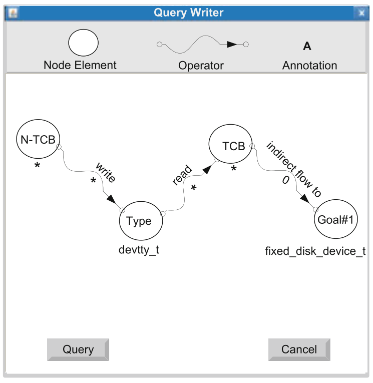
The main window in Fig. 7a shows the visualized SELinux policy based on semantic substrate design proposed in Sect. 3.1. The policy is composed of 308 domains, 1092 types and 31604 links. The Y-axis is divided into four regions including USERS, ROLES, DOMAINS and TYPES. The X-axis is labeled using the domain and type classifications discussed in Sect. 3.1. 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 (device types), FT (file types), PT (procfs types), DE (devpts types), NF (nfs types), and NE (network types). To help the policy administrator to easily identify the different regions, the elements in non-neighboring regions are represented as different shapes, for example users and domains are expressed with circle, and


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

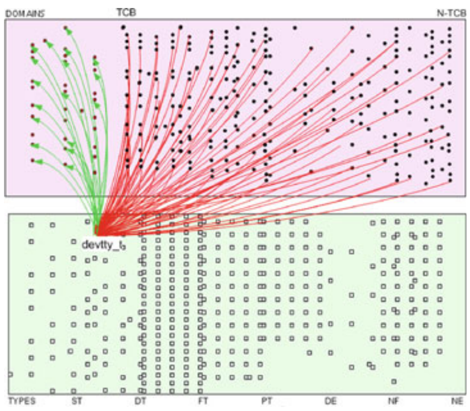
(a) Policy Visualization in Semantic Substrates



(b) Policy Visualization in Adjacency Matrix



(c) Policy Violation Query Example



(d) Policy Violations

Fig. 7 PVA tool introduction

roles and types are expressed with rectangle. The edges between different regions are represented by different colored lines, for instance the write operation between a domain and a type is 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 7b shows the adjacency matrix-based policy visualization method, which was compiled by selecting a subset of the nodes in the semantic substrates overlay.

5.3 Policy query and violation detection

Figure 7c shows the graphical query interface and a query designed to discover the paths from N-TCB to resources related to the goal G1 (limiting raw access to data) such as `fixed_disk_device_t` through a specific type `devtty_t` and TCB resources. Starting from left to right (Fig. 7c), the first node selects the N-TCB resources and finds the paths to a 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 7d 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 a 1.4 GHz Intel Pentium CPU with 512 MB of memory, query loading and parsing take 15 s and query execution and display take 21 s. Another example query, which investigates information flow paths from N-TCB to `fsadm_t` (TCB) without the constraint of passing through a specific intermediate node, was executed and displayed in 88 s due to the large number of policy violations. However, if we only query the information flow path from `user_t` (N-TCB) to `fsadm_t` (TCB), it takes about 0.5 s. Hence, the time cost depends on the size of the query set and query results. Table 1 shows identified policy

Table 1 Policy violation examples

Subjects	Type : Class	Subject	Resolution
Example policy violations			
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

violations caused by information flow from N-TCB to TCB `fsadm_t`.

6 User study

This section describes our user study that was conducted to evaluate the usability of PVA. Participants in our study browse and analyze SELinux security policies using PVA and APOL [36] that is a GUI-based tool developed by Tresys Technology to analyze SELinux policies. Currently, the APOL analysis tool is issued together with other Linux packages such as Fedora Core. APOL helps browse and search policy components (e.g., types, attributes, object classes, roles, users, and booleans), searching type enforcement and other rules, and viewing file contexts from a filesystem. In addition, APOL allows policy administrators to perform automated, complex analysis of a policy, which include domain transition, file relabel, types relationship, and information flow analysis. In our evaluation work, we mainly focus on comparison between PVA and APOL. Also, participants are required to compare semantic substrates method and adjacency matrix method in displaying of SELinux security policies and policy violations.

6.1 Participant enrollment and general feedback

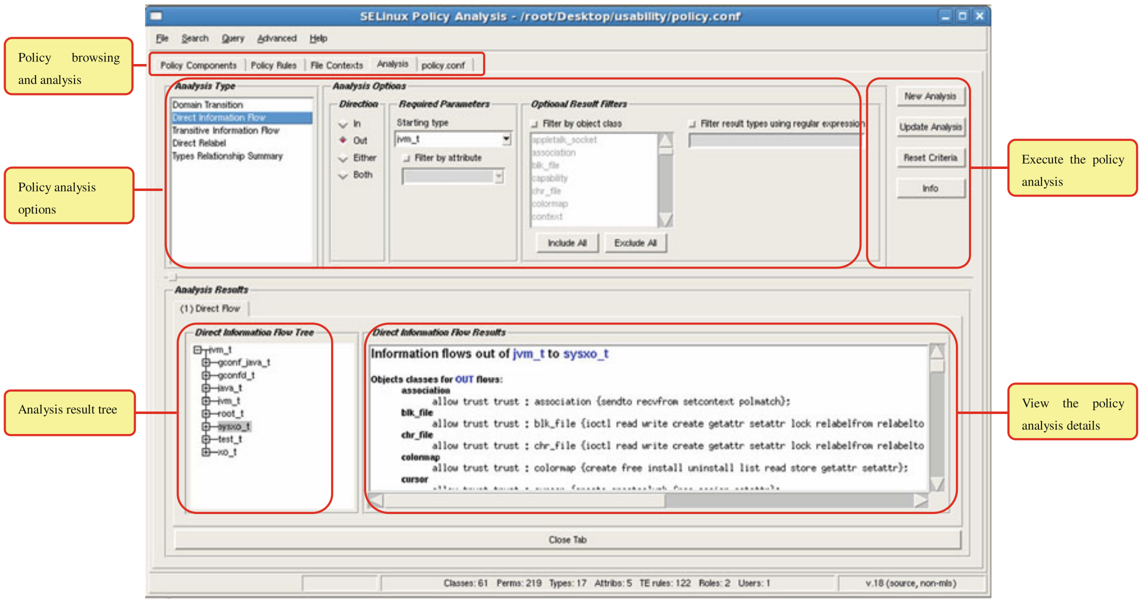
To enroll the participants, we first interviewed some student participants after giving guest lectures. Other participants were recruited from several research laboratories and the system administrator office of the college. In addition, we contacted some participants who were recommended as either Linux expert or system administrators and had a total of 60 enrolled users.

The sample for our study consists of system administrators (15.2 %), graduate students (72.7 %), and undergraduate students (12.1 %) with different specialities within computer science discipline (69.7 % information security; 6.1 % computer networking; 6.1 % database systems; 6.1 % computer graphics and visualization; and 12.1 % other) of various ages [12.1 % were 18–22 years old (y. o.), 42.4 % were 22–26 y. o., 30.3 % were 26–30 y. o., and 15.2 % were 30–40 y. o.]. Participants differed in terms of their most frequently used OS (15.2 % Linux; 72.7 % Windows; and 12.1 % Mac OS), the frequency with which they change OS configuration settings (27.3 % never; 54.5 % monthly; 12.1 % weekly; and 6.1 % daily), the frequency with which they configure or check their OS security policies (30.3 % never; 54.5 % monthly; 6.1 % weekly; and 9.1 % daily), whether or not they use special software tools to manage their OS security policies (21.2 % use special software tools; 63.6 % do not use special software tools; and 15.2 % do not know whether or not they use special software tools), the extent to which they agree that configuring security policies is an important task (3 % strongly disagree; 0 % disagree; 0 % neither agree not disagree; 45.5 % agree; and 51.5 % strongly agree), and the amount of time they would be willing to spend on configuring a security policy (6.1 % no time; 33.3 % up to 15 min; 30.3 % up to 30 min; 6.1 % up to 1 h; 3 % up to 2 h; and 21.2 % more than 2 h).

6.2 User study preparation

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

- This real policy set is representative of security policies that the policy administrator might encounter in practice.
- The size of policy set is about 200 KB that covers core policies and sufficiently helps measure usability of tools. Although there might be a larger SELinux policy set



```
Information flows out of jvm_t to sysxo_t

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

Fig. 8 APOL tool introduction

available, analyzing such policies will bring difficulty for participants in both evaluating APOL and PVA. On the other hand, in case the security policy size is too small, the benefit of visualization-based approach will be probably downgraded.

First, in order to give ideas about how to use APOL and PVA and how to compare the two visualization mechanisms in PVA, we introduced SELinux policy overview to the participants and an instruction was provided with screenshots as shown in Figs. 7 and 8.

The first part of the instruction explains how to use APOL and PVA for browsing and analyzing security policies. The participants are instructed to load the security policy set and browse any contents they are interested in. Then, the participants are required to follow the instructions for completing the following tasks: (1) identifying the information types containing `gcon`; (2) identifying all the direct information flows from `jvm_t` ; (3) identifying direct information flows from `jvm_t` to `java_t`; (4) identifying indirect information flows from `test_t` to `user_install_t` ; (5) identifying direct information flows from `jvm_t, test_t` to `root_t` ; (6) identifying information flows from `jvm_t` through `root_t` to `sysxo_t` ; (7) identifying direct information flows from group 1 = `{test_t, jvm_t}` to group 2 = `{xo_t, root_t}` and group 3 = `{user_install_var_t, user_install_exec_t}`; and (8) repeating steps 6 and 7 to examine the result of analysis composition. In addition, the participants are asked to test the tool freely and provide their feedback on how the interface is designed, how the security policy is composed, and other information that would be necessary for our user study.

The second part of the instruction mainly elaborates how to use the semantic substrates approach and adjacency matrix approach separately to visualize the security policy set, read information from the visualized policy, generate and display security policy violations against predefined security goals, and identify the root causes of the existing policy violation such as why there exists an information flow from an untrusted domain `user_install_t` to a trusted domain `sysxo_t`.

6.3 User study procedures and data collection

To perform the user study, we installed APOL and PVA tools on our lab machine in an independent room where the participant cannot be interrupted during their session. The participants first performed the five classified policy queries with example scenarios using APOL and PVA based on our instruction. Then, the participants were required to visualize and identify security policy violations with the Node-Link and Adjacency Matrix approach. For conducting these steps, the participants are required to use the same example SELinux policy.

After completing the policy analysis, participants were asked to complete a post-session questionnaire assessing their attitudes toward two tools and two approaches, their experiences with and attitudes toward security policy analysis, their general control inclinations, and their demographic characteristics.

In the questionnaire, we design totally 20 items to measure participant attitude including the ease of constructing queries, the ease of understanding policy analysis results and the overall ease of using the tool interface. Among these measures, the first two items are to measure PVA and APOL in the policy analysis aspect and the latter items focus on policy browsing, understanding and the overall experience of the tool interface. For each measure, we design separate questionnaires for each tool but with semantically equivalent items. The following summarizes what we intend to measure.

- *Ease of constructing queries* using APOL and PVA for policy analysis was measured with 4 items. Participants were asked to rate the easiness of the processes described in the items using APOL and PVA functions on a 5-point rating scale (1 = very complicated to 5 = very easy). A sample item is "browsing the security policy using APOL was."
- *Ease of understanding query results* using APOL and PVA for policy analysis was measured with 3 items rated on a Likert scale (1 = strongly disagree to 5 = strongly agree). A sample item is "PVA can give you some general knowledge about the loaded SELinux policy."
- *Overall ease of using the interface* using APOL and PVA for policy analysis was measured with 3 items rated on a Likert scale (1 = strongly disagree to 5 = strongly agree). A sample item is "The APOL interface is easy to combine and compose policy queries."
- *Ease of identifying policy violations* using policy visualization functions of Node-Link and the Adjacency Matrix methods was measured with 3 items rated on a Likert scale (1 = strongly disagree to 5 = strongly agree). A sample item is "It is easy to identify trusted and untrusted domains."
- *Ease of tracing policy violation elements* using policy visualization functions of Node-Link and the Adjacency Matrix methods was measured with 3 items rated on a Likert scale (1 = strongly disagree to 5 = strongly agree). A sample item is "This visualization is clear, not crowded and not cluttered."
- *Satisfaction with the visualization policy* using policy visualization functions of Node-Link and the Adjacency Matrix methods was measured with 3 items. Participants were asked to rate the degree to which they liked different aspects of the two policy visualization functions on a 5-point rating scale (1 = do not like at all to 5 = like it very much). A sample item is "Viewing specific policy element relationships (domains and types)."
- *Scale scores* were calculated by computing participants' mean responses to the items included in each scale.
- *Participant characteristics* were aggregated based on their feedback such as their age range, specialization, occupation, and their general attitudes and behaviors regarding the configuration of OS settings, such as the security policies.

Table 2 PVA versus APOL (number of participants = 32)

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

For the questionnaire design, we implemented the questionnaire using the lime survey tool [17] on our lab server. For each participant, our lab sever recorded their answers to the questions and helped determine whether or not they finished the questions. Their answers were stored and can be exported through a database for subsequent analysis.

6.4 User study results

PVA versus APOL: Paired samples *t*-tests [23] were conducted to compare participants' attitudes toward two approaches in policy analysis. Corresponding to each measure, we were able to produce the following results:

- *Ease of constructing queries:* Through calculating the answers for this measure, the satisfaction tendency for APOL was ($M = 2.78$, SD = .94)[1] compared to PVA ($M = 4.36$, SD = .41). Therefore, we concluded that constructing queries with PVA is perceived to be significantly easier than APOL, where $t(32) = -9.67$ and $p < .001$.
- *Ease of understanding query results:* Based on the the answers for this measure, the satisfaction tendency was calculated. APOL's satisfaction tendency was ($M = 3.13$, SD = .78) compared to PVA ($M = 4.32$, SD = .40). Hence, understanding query results with PVA is also perceived to be relatively easier than APOL, where $t(32) = -9.07$ and $p < .001$.
- *Overall ease of using the interface:* Through examining the answers for this measure, the satisfaction tendency for APOL was ($M = 2.41$, SD = .86) compared to PVA ($M = 4.42$, SD = .43). Also, the overall interface of PVA is perceived to be significantly easier to use than the APOL interface, where $t(32) = -11.82$ and $p < .001$.

In summary, participants indicated that all measurements of the PVA approach were superior to the corresponding aspects of the APOL approach in performing policy analysis as shown in Table 2.

[1] M and SD denote *mean* and *standard deviation*, respectively.

Table 3 Semantic substrates versus adjacency matrix (number of participants = 32)

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

Semantic Substrates versus Adjacency Matrix: With a method similar to that we used to compare PVA with APOL, paired samples t-tests [23] were conducted to compare participants' attitude toward the two visualization approaches for policy analysis. For each measure, our results were as follows:

– *Ease of identifying policy violations:* Through calculating the answers for this measure, the satisfaction tendency for semantic substrates was ($M = 4.40$, SD = .40) compared to adjacency matrix ($M = 4.17$, SD = .64). It implies that identifying policy violations with the semantic substrates visualization is perceived to be easier than adjacency matrix visualization, where $t(32) = 2.21$ and $p < .05$.
– *Satisfaction with the visualization policy:* Based on the answers for this measure, although there was a tendency for participants to be more satisfied with the Node-Link visualization compared to the adjacency matrix visualization ($M = 4.11$, SD = .65), this tendency is not statistically significant, where $t(32) = 1.98$ and $p = .056$.
– *Interpretability of visualization results:* Based on the answers for this measure, the satisfaction tendency for semantic substrates was ($M = 4.23$, SD = .64) compared to adjacency matrix ($M = 3.99$, SD = .70). Therefore, we could conclude that interpreting the visualization results with the semantic substrates visualization is perceived to be not significantly easier than adjacency matrix visualization, where $t(32) = 1.98$ and $p < .056$.

In summary, although there is a tendency for participants' perceptions of the semantic substrates visualization to be more favorable, compared to their perceptions of the adjacency matrix visualization, this tendency is only significant with respect to the ease of identifying policy violations (Table 3).

6.5 Lessons learned from user study

This section describes some lessons that we learned through the evaluation of PVA. Although the evaluation itself may have diverse goals and claims for different types of security policies, we share our lessons so that other researchers can consider these lessons for their policy analysis work. The lessons learned are summarized as follows:

Lesson 1: Ensure that participants understand the assigned task. It is necessary to clearly communicate with participants and confirm participants' understanding before initiating the survey. We found out that the best way to achieve this objective was to ask the participants to explain back to us what they understood and they needed to do. These participants had less problems in executing the assigned tasks than the participants who simply nodded to passively indicated that they understood.

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

Lesson 3: Use as little paper work as possible. Initially, we designed a hard-copy questionnaire which was perceived to be too exhaustive and boring to the participants. Also the paper work caused additional problem such as bringing overhead to the data analysis, difficulty for data backup, and so on. Hence, we prepared online-version of the questionnaire which tremendously helped us leverage the benefits of web-based application such as interactive question & answer, data collection, analysis, and backup.

Lesson 4: Minimize the chances that participants may make mistake. It was a challenging task to minimize the chances that participants could make mistakes unrelated to the claims. In other words, we should pay attention not to introduce unnecessary complications to the tasks. For example, in our instructions, we stated that PVA allows users to perform the same kind of security policy analysis through two different ways. Having both mechanisms to perform the analysis, some participants felt that they could choose one mechanism based on their first impression of GUIs. Consequently, it would introduce unnecessary confusion to the participants. For another example, in our initially designed questionnaire, it had a "no answer" option for some items, which resulted in data collected to be invalid and required us to re-invite participants.

There are several directions that we may consider to improve our user study for the future work. First, the lack of policy analysis tool was one of critical obstacles in our user study. Due to such limitations, we could select only the most sophisticated tool APOL for our user study. Therefore, we could not have a chance to test PVA with more generic evaluation criteria. Also, APOL does not consider *trusted* and *untrusted* concepts in the tool. With such a distinction, we can formulate and detect information flows. Hence, we were not able to examine corresponding functionalities of PVA compared to APOL. Second, some naive participants have never used any policy analysis tool before so they were not able to provide more constructive feedback on each questionnaire–even though their unbiased feedback was significantly helpful to evaluate our tool. The participants with different background are critical for user study but it is also crucial to be balanced. Third, both APOL and PVA worked a bit slow when analyzing a large security policy set to identify policy violations against a large volume of information flow. However, some participants were intolerant of such delays. Especially, if the policy size reaches gigabyte, APOL and PVA consumed most of system resources. Hence, we had to reduce the size of policy set as we discussed before. The evaluation with diverse data sets would provide some potential clues to improve our tools in terms of interface design and functional aspects. Fourth, although our current user sample can evaluate the user study well, enrolling more users into the user study will make the study more meaningful.

7 Related work

Previous typical methods and tools developed to analyze SELinux policies include Gokyo [12,28,14,13], SLAT [7], PAL [27] and APOL [36]. Gokyo was used to check integrity of a proposed 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 cannot 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 transfers information from process to resource; read operation transfers 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 by analyzing an information flow model. Sarna-Sota et al. [7] used SLAT's 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 run queries for analyzing the policy. APOL [36] 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 familiar with SELinux policies for generating queries against the policy base which ultimately leads to extracting meaningful information. Furthermore, APOL tool provides graphical interface to aid policy analysis and supports policy query. However, our PVA has the following differences and advantages compared to APOL. First, PVA is built based on a policy visualization framework. All the policy rules can be expressed with graphs and their relationships are automatically displayed. APOL only provides text-based expressions for the policy rules. A policy administrator has to read the policy rules one by one for identifying the relationship between rules. Second, we support the visual query formulation that enables a user to easily construct the query. Especially, the administrator can construct complex queries to examine policies for their need. APOL only supports a simple query construction based on clicking buttons and choosing items from the list provided by the graphical interface. Our user study clearly indicates users found our policy analysis framework more intuitive to perform tasks. In addition, PVA tool supports N-TCB and TCB concepts for policy violation detection while APOL does not support such concepts and corresponding functions.

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, which is similar to the query by example approach. Our main methodology also facilitates visualization-based queries to identify the possible policy violations. We have developed a 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. In addition, we have conducted a user study to evaluate the usability of our policy visualization framework.

For the future work, we plan to investigate node and link reordering mechanisms that minimize the link crossings and entanglement for more appealing policy visualizations. Also, the application of our framework for visualizing and analyzing the web-based access control policy such XACML policies will be investigated.

Acknowledgments The work of Gail-Joon Ahn and Wenjuan Xu was partially supported by the grants from National Science Foundation and Department of Energy.

References

1. Anderson, A.P.: Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, II (1972)
2. Aris, A.: Network visualization by semantic substrates. IEEE Trans. Vis. Comput. Graph. **12**(5), 733–740 (2006). Senior Member-Ben Shneiderman
3. Biba, K.J.: Integrity Consideration for Secure Compuer System. Technical report, Mitre Corp. Report TR-3153, Bedford, Mass (1977)
4. Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
5. Erbacher, R.: Intrusion behavior detection through visualization. In: IEEE International Conference on Systems, Man and Cybernetics, pp. 2507–2513 (Oct 2003)
6. Green, M.: Toward a perceptual science of multidimensional data visualization: Bertin and beyond. Available from http://www.ergogero.com/dataviz/dviz2.html, 1998
7. Guttman, J., Herzog, A., Ramsdell, J.: Information flow in operating systems: Eager formal methods. In: Workshop on Issues in the Theory of Security (WITS) (2003)
8. Herman, I., Melancon, G., Marshall, M.: Graph visualization and navigation in information visualization: A survey. IEEE Trans. Vis. Comput. Graph. **6**(1), 24–43 (2000)
9. H.C. I. L. at University of Maryland. Piccolo. Available from http://www.cs.umd.edu/hcil/jazz/download/index.shtml
10. Itoh, T., Takakura, H., Sawada, A., Koyamada, K.: Hierarchical visualization of network intrusion detection data. IEEE Comput. Graph. Appl. **26**(2), 40–47 (2006)
11. Jaeger, R.S.T., Zhang, X.: Resolving Constraint Conflicts. In: Sacmat '04: Proceedings of the Ninth Acm Symposium on Access Control Models And Technologies, pp. 105–114 (2004)
12. Jaeger, X.Z.T., Edwards, A.: Policy management using access control spaces. ACM Trans. Inf. Syst. Secur. (TISSEC) **6**, 327–364 (2003)
13. Jaeger, T., Sailer, R., Shankar, U.: Prima: policy-reduced integrity measurement architecture. In: SACMAT '06: Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies, pp. 19–28. ACM, New York, NY, USA (2006)
14. Jaeger, T., Sailer, R., Zhang, X.: Analyzing integrity protection in the selinux example policy. In: SSYM'03: Proceedings of the 12th Conference on USENIX Security Symposium, pp. 59–74. USENIX Association, Berkeley, CA, USA (2003)
15. Keller, R., Eckert, C.M., Clarkson, P.J.: Matrices or node-link diagrams: which visual representation is better for visualising connectivity models? Inf. Vis. **5**(1), 62–76 (2006)
16. Lee, C., Trost, J., Raheem, N.G.B., Copeland, J.: Visual firewall: Real-time network security monitor. In: IEEE Workshops Visualization for Computer, Security, pp. 129–136 (2005)
17. Lime Survey Tool http://www.limesurvey.org/
18. Loscocco, P., Smalley, S.: Integrating flexible support for security policies into the linux operating system. In: USENIX Annual Technical Conference, FREENIX Track, pp. 29–42 (2001)
19. Loscocco, P.A., Smalley, S.D.: Meeting critical security objectives with security-enhanced linux. In: Proceedings of the Ottawa Linux Symposium (2001)
20. Mathew, S., Giomundo, R., Upadhyaya, S., Sudit, M., Stotz, A.: Understanding multistage attacks by attack-track based visualization of heterogeneous event streams. In: VizSEC '06: Proceedings of the 3rd International Workshop on Visualization for Computer Security, pp. 1–6. ACM, New York, NY, USA (2006)
21. Nidhi, S.: Fireviz: A personal firewall visualizing tool. In: Thesis (M. Eng.), Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science (2005)
22. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, pp. 109–118. ACM, New York, NY, USA (2004)
23. Paired Samples T-tests. http://www.statisticssolutions.com/methods-chapter/statistical-tests/paired-sample-t-test/
24. Reiterer, H., Muler, G.: A visual information seeking system for web search. In: Proceedings of the Oberquelle, H., Oppermann, R., Krause, J. (eds) Mensch & Computer Conference, pp. 297–306, (March 2001)
25. Reiterer, H., Tullius, G., Mann, T.: Insyder: A content-based visual-informationseeking system for the web. Springer-Verlag GmbH, International Journal on Digital Libraries (2005)
26. Saltzer, J., Schroeder, M., (1975) The protection of information in computer systems. In: Proceedings of the IEEE, pp. 1278–1308.
27. Sarna-Starosta, B., Stoller, S.D.: Policy analysis for security-enhanced linux. In: Proceedings of the 2004 Workshop on Issues in the Theory of Security (WITS), pp. 1–12 (April 2004)
28. Shankar, U., Jaeger, T., Sailer, R.: Toward automated information-flow integrity verification for security-critical applications. In: NDSS, The Internet Society (2006)
29. Shen, Z., Ma, K.: Path visualization for adjacency matrices. In: Proceedings of Eurographics/IEEE Symposium on Visualization (EuroVis), May 2007
30. Smalley, S.: Configuring the SELinux policy. http://www.nsa.gov/SELinux/docs.html, 2003
31. Sutcliffe, A.G., Ennis, M., Watkinson, S.J.: Empirical studies of end-user information searching. J. Am. Soc. Inf. Sci. **51**(13), 1211–1231 (2000)
32. Secure computer systems: Unified exposition and multics interpretation. MITRE Corporation, 1976
33. System management concepts: Operating system and devices, 1 ed., (1999)
34. Thompson, R.S., Rantanen, E.M., Yurcik, W., Bailey, B.P.: Command line or pretty lines?: comparing textual and visual interfaces for intrusion detection. In: CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1205. ACM, New York, NY, USA (2007)
35. Tran, T., Al-Shaer, E.S., Boutaba, R.: Policyvis: Firewall security policy visualization and inspection. In: Lisa, pp. 1–16 (2007)
36. Tresys Technology Apol. http://www.tresys.com/selinux/
37. Yao, D., Shin, M., Tamassia, R., Winsborough, W.H.: Visualization of automated trust negotiation. In: VizSEC 05: IEEE Workshop on Visualization for Computer, Security, Oct 2005
38. Yin, X., Yurcik, W., Treaster, M., Li, Y., Lakkaraju, K.: Visflowconnect: netflow visualizations of link relationships for security situational awareness. In: VizSEC/DMSEC '04: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 26–34. ACM, New York, NY, USA (2004)

39. Yurcik, W.: Visualizing netflows for security at line speed: the sift tool suite. In: LISA'05: Proceedings of the 19th Conference on Large Installation System Administration Conference, pp. 169–176. USENIX Association, Berkeley, CA, USA (2005)
40. Yurcik, W.: Tool update: visflowconnect-ip with advanced filtering from usability testing. In: VizSEC '06: Proceedings of the 3rd International Workshop on Visualization for Computer Security, pp. 63–64. ACM, New York, NY, USA (2006)