SEGrapher: Visualization-based SELinux Policy Analysis

Said Marouf
Department of Software & Information Systems
University of North Carolina at Charlotte
Charlotte, NC, USA
smarouf@uncc.edu

Mohamed Shehab
Department of Software & Information Systems
University of North Carolina at Charlotte
Charlotte, NC, USA
mshehab@uncc.edu

***Abstract*—Performing SELinux policy analyses can be difficult due to the complexity of the policy language and the sheer number of policy rules and attributes involved. For example, the default policy on most SELinux-enabled systems has over $1,500,000$ flat rules, involving over $1,780$ *types*. Simple analyses between *types* can result in a large amount of data, which is poorly presented to administrators in existing analysis tools. We propose and implement a policy analysis tool "SEGrapher" that addresses the above challenges. SEGrapher visually presents analysis results as a simplified directed graph, where nodes are *types*, and edges are corresponding policy rules between *types*. Graphs are generated via a proposed clustering algorithm that clusters *types* based on their accesses. Clusters provide an abstraction layer that removes undesired data, and focuses on analysis attributes specified by the administrator.**

I. Introduction

One of the significant advancements in the security of the Linux operating system, came with the introduction of Security-Enhanced Linux (SELinux) [1], developed by the U.S. National Security Agency. SELinux is a kernel Linux Security Module (LSM) that adds Mandatory Access Control (MAC) to a regular Linux system through a Type-Enforcement model, in which *domains* (subject-types) are used to label processes, and *types* (object-types) are used to label files. Most Linux distributions nowadays, come with support for SELinux, and by default use the SELinux "targeted" policy, which has over $1,500,000$ flat rules and over $1,780$ policy types. With such a large number of rules and types, and the complexity of the policy language, managing and analyzing SELinux policies becomes a daunting task for average administrators (admins) [2], [3], [4]. Existing policy analysis tools [5], [6], [7], assume that admins are highly knowledgable in all aspects of SELinux policies, and are able to easily understand and interpret policy rules. Furthermore, most existing tools output a list of text-based analysis results, that do not provide a clear overview of any inherent relations between types involved in these results. For example, it is difficult to understand relations that are based on multiple policy rules. It is also difficult to discover unnecessary policy rules that might be redundant, or rules that could further be optimized.

Information visualization is a means for enabling users to easily analyze, reason, and explain abstract information using their visual cognition [8]. Information visualization has been adopted in security domains, to better understand and represent collected data related to intrusion detection [9], firewall policies [10], and network attacks [11]. In the case of policy admins, visualizing analysis results can assist them in easily understanding the direct and inherent relations between involved types, and to easily discover new interesting relations that can lead to simpler policy configurations.

Other challenges that can face admins involve the analysis of new policy rules. We believe providing visual cues to admins can simplify the analysis process.In this paper, we propose and implement a visualization-based policy analysis tool, that simplifies policy analyses. The main contributions of this paper are:

- A clustering algorithm that clusters policy *types* based on their allowed accesses, and builds a focus-graph based on identified clusters. Clusters provide an abstraction layer that easily allows admins to discover interesting and inherent relations between types.
- A visualization-based policy analysis tool "SEGrapher", which implements our proposed clustering algorithm. SEGrapher provides a simple, yet powerful interface, that easily allows for analyzing multiple *types* at once, rather than the usual one-to-one analysis.

II. Preliminaries

A. SELinux Policies

SELinux policies are considered quite difficult to manage due to the granular level of controls they provide [2], [3], [4]. Even though this is true, an SELinux policy at its core is no different than other access control policies in which a set of rules are introduced to enforce and achieve an overall security goal. A typical access control policy rule is built around a *subject* which is granted certain *actions* on a certain *object*. For example, John (subject) is allowed to play (action) all mp3 files (object) on a system. The same model is applied in SELinux policy rules but with more elaborate and fine-grained levels of control.

SELinux labels each resource, such as files and processes within an SELinux-enabled system with a *security context*. A security context is a label that usually incorporates three fields: 1) SELinux User, 2) Role, 3) Type, and sometimes includes

a *level* field for applying Multilevel Security (MLS). In this paper we focus on the "Type", which represents the core of access control rules that determine what *subject-types* have what accesses on which *object-types*. Object-types are defined to group file objects, whereas subject-types are defined for processes. Types defined for processes are usually referred to as domains. Objects that fall under the same object-type, are similar in which subjects access them. Subjects or processes that are under the same subject-type, are similar in which objects or files they access. An example of an object-type is the `user_home_t` type, which is used to group files owned by a user and reside in his/her home directory. Grouping here, is achieved by setting the type within each file's security context to `user_home_t`. An example subject-type is the `httpd_t` type, which belongs to the Apache HTTP server process.

In this paper, we focus on the Access Vector (AV) rules within an SELinux policy. There are three main types of AV rules, *allow*, *auditallow*, and *dontaudit*. Our work involves the *allow* type, which is responsible for allowing accesses between types, whereas the latter two are for auditing purposes. A typical AV allow rule specifies how a subject-type is allowed to interact with an object-type. The building blocks of any AV allow rule are the following:

- **Subject-type**: The subject of the access control rule which is granted certain accesses.
- **Object-type**: The object or resource to be accessible by the subject of this rule.
- **Object-class**: Each object within SELinux falls under a certain class (*object-class*). Each object-class has a corresponding set of applicable actions (permissions). For example, *file* and *dir* are object-classes that respectively correspond to files and directories within a system. Having object-classes allows for easier management of permissions on objects. For example, a *read* permission has a different interpretation when applied to files vs. directories, hence having an associated permission set for each object-class allows for easier interpretation of the intended permission, i.e. *read* on object-class *file* is not the same as *read* on object-class *dir*.
- **Permissions**: For each object-class there is an associated set of permissions, i.e. a set of actions that the subject can take on the object. For example, the *file* class has the permissions *read, write, create, rename* an so forth.

Following is an example AV allow rule written in the SELinux AV rule syntax:

```
allow httpd_t httpd_log_files_t:file {read}
```

this reads as: allow the subject-type `httpd_t` to *read* files of object-type `httpd_log_files_t`. Or in a more readable format this reads: Allow the Apache HTTP process to read its log files.

Most Linux distributions come with support for SELinux. The standard SELinux policy provided is called the "Reference Policy" which is currently developed and maintained by Tresys [15]. The reference policy acts as a mutual ground for policy developers and can be tweaked according to the specific security requirements of a system. Two popular variations of the reference policy are the *targeted* policy and the *strict* policy. The targeted policy allows for controlling specific services such as WEB or FTP servers, whereas the strict policy takes full control of a system. In summary, the targeted policy allows for a more permissive system that can be incrementally locked down by admins, whereas the strict policy starts with a nearly locked system that needs to be incrementally opened up appropriately. Our interest in these policy variations comes from the fact that both have quite a different set of defined types, and AV rules. We found the targeted policy to have 1,785 types and 1,517,130 allow rules, vs. 2,321 types and 1,766,729 allow rules for the strict policy. Such large policies emphasize the need for better analysis tools.

III. SELinux Policy Analysis

Let T be the set off all types within a SELinux policy P, O the set of all object-classes, and A the set of all permissions. We propose a policy analysis tool "SEGrapher" which allows for visualizing policy analysis results, by modeling a policy as a directed graph. Given a policy P, SEGrapher builds a directed graph G_p, where a node in G_p maps to a specific SELinux type, and an edge (out-edge) maps to the set of all AV allow rules R_{ij} connecting a type t_i (subject-type) to a type t_j (object-type). Figure 1 illustrates a simple graph of three types, t_1 (subject-type), t_2 (object-type), and t_3 (object-type). The figure shows the corresponding AV rules R_{12} for t_1 and t_2 with two allow rules, and R_{13} for t_1 and t_3 with one allow rule.

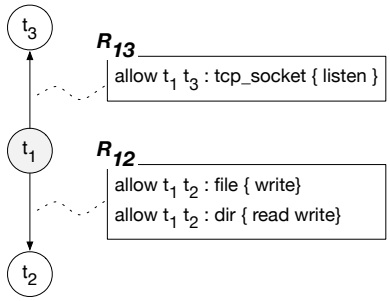


Fig. 1. AV allow rules for subject-type t_1 and object-types t_2 and t_3, represented as edges in G_p

SEGrapher uses G_p to generate a directed focus-graph G_f representing desired analysis results, that is, G_f will indicate the accesses and relations amongst SELinux types analyzed by admins. G_f is driven by a set of inputs that are checked against AV rules (edges) in G_p. These inputs are controlled and provided by admins and include the following:

1) **Focus Types T_f:** A set of types $T_f \subseteq T$ which is the focus of the policy analysis and the basis of extracting the focus-graph G_f from G_p. An out-edge in G_p is added to G_f if the source-node (subject-type) of this out-edge exists in T_f.
2) **Focus Object-Class o_f:** An object-class $o_f \in O$, which is used to filter the out-edges that already satisfy the focus-types T_f.

3) **Focus Permissions** A_f**:** A set of permissions $A_f \subseteq A$, which are used to further filter the out-edges that already satisfy both T_f and the focus object-class o_f.

With the provided T_f, o_f, and A_f, an out-edge in G_p with an AV allow rule set R_{fn} is added to G_f if for any $r_i \in R_{fn}$, *all* of the following conditions are satisfied:

1) The subject-type for r_i exists in T_f.
2) The object-class for r_i = o_f
3) A_f exists within the permissions for r_i.

For example, let $T_f = \{t_1\}$, $o_f = dir$, and $A_f = \{write\}$. When applying these inputs onto the graph in Figure 1, a new focus-graph G_f is generated as illustrated in Figure 2. Note that, only out-edges with AV rule sets fulfilling the above conditions make it to G_f.

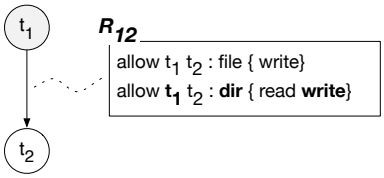


Fig. 2. Filtered AV allow rules for subject-type t_1 and object-type t_2 become an edge in G_f

With 1,517,130 AV allow rules, 1,785 types, 47 object-classes, and 167 different permissions, the full SELinux reference policy graph is infeasible to analyze at once. Even when applying the analysis inputs T_f, o_f, and A_f, a resulting focus-graph G_f can be difficult to analyze. In many cases, simply analyzing a single focus-type can result in a large number of AV allow rules, hence a dense focus-graph G_f. For example, to analyze the *read* accesses of the Samba Server [17] on directories within an SELinux-enabled Linux system, let $T_f = \{smbd\_t\}$, $o_f = \{dir\}$, and $A_f = \{read\}$ where `smbd_t` is the subject-type (domain) corresponding to the Samba Server. This analysis results in 1,048 AV allow rules, hence a dense G_f of 1,048 edges and 1,049 nodes. If we add a second type `ftpd_t` (FTP Server) to T_f, and run a new analysis, we'll find that the number of edges in G_f almost doubles to 2,095, leading to a very dense graph, whereas the number of nodes increases just to 1,052. This is due to the fact that both `smbd_t` and `ftpd_t` have a large overlap in the object-types they access, i.e. their out-edges share a large set of end nodes within G_f.

Observation 1. Many subject-types in SELinux have a large overlap of object-types that they access. In some cases they access the exact set of object-types, and in other cases there is a hierarchical relation between the sets accessed.

Based on Observation 1, we define the following terms and relations between types t_i and t_j in T_f:

Definition 1. (Object-Type Set) The object-type set $To_i \subseteq T$ for type t_i is the set of object-types in all AV allow rules, where an AV rule's subject-type is t_i. That is, the set of all types that t_i can access.

Definition 2. (Matching Types) Types t_i and t_j are *matching* if their respective object-type sets T_{o_i} and T_{o_j} are equal. Formally,

$$t_i \Re_m t_j \iff (T_{o_i} = T_{o_j})$$

Definition 3. (Hierarchical, Parent-Child Types) A *parent-child* relation between types t_i (parent) and t_j (child) exists when t_i's object-type set T_{o_i} is a proper superset of t_j's object-type set T_{o_j}. Formally,

$$t_i \Re_h t_j \iff (T_{o_i} \supset T_{o_j})$$

Definition 4. (Overlapping Types) Types t_i and t_j are *overlapping* if their respective object-type sets T_{o_i} and T_{o_j} overlap and neither $t_i \Re_m t_j$ nor $t_i \Re_h t_j$ holds . Formally,

$$t_i \Re_o t_j \iff (T_{o_i} \cap T_{o_j} \neq \phi) \wedge \overline{t_i \Re_m t_j} \wedge \overline{t_i \Re_h t_j}$$

Definition 5. (Disjoint Types) Types t_i and t_j are *disjoint* if their respective object-type sets T_{o_i} and T_{o_j} are disjoint. Formally,

$$t_i \Re_d t_j \iff (T_{o_i} \cap T_{o_j} = \phi)$$

These relations can assist in discovering other interesting relations between types t_i and t_j in T_f, and help answer a number of analysis questions such as, but not limited to:

- Are t_i and t_j redundant? (Possible when $t_i \Re_m t_j$).
- What accesses does t_i have, but t_j doesn't? (Possible when $t_i \Re_h t_j$, or $t_i \Re_o t_j$).
- Can we simplify a policy by removing overlapping (redundant) AV rules within P? (Possible when $t_i \Re_o t_j$).

Note that our focus is not on one-to-one type relations, i.e. can $t_i \in T_f$ access $t_j \in T_f$, but on more interesting relations that exist between t_i and t_j which can eventually lead to simpler policy configurations and an easier analysis process. One-to-one relations between t_i and t_j can still easily be identified from the relations above. In SEGrapher we uniquely visualize the focus-types T_f, which makes it easy to identify them and to identify any one-to-one relations that may exist between them. Based on the defined relations $\Re_m$, $\Re_h$, $\Re_o$, and $\Re_d$, and our higher goal of discovering new relations, we propose a clustering algorithm in section III-A that utilizes and exposes existing relations between focus types in T_f. By exposing these relations and building a cluster-based focus-graph reflecting these relations, the algorithm is able to visually simplify focus-graphs, hence simplify the policy analysis process.

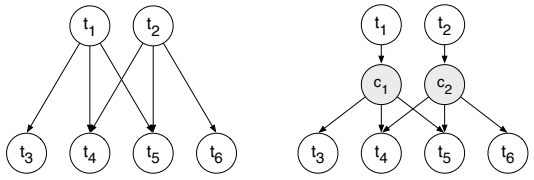
A. Type Clustering

We propose and implement a clustering algorithm that utilizes the relations $\Re_m$, $\Re_h$, $\Re_o$, and $\Re_d$ identified above. Given focus-types T_f, object-class o_f, permissions A_f, and an edge-reduction threshold τ_e, we extract existing relations from a policy graph G_p and generate a set of clusters C where each cluster $C_i \in C$ becomes a node within a new cluster-based focus-graph G_f.

Algorithm 1: Generate Clustered Policy Focus-Graph

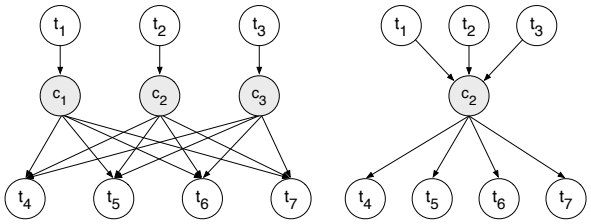
input : Policy graph G_p, focus-types T_f, object-class o_f, permissions A_f, and threshold τ_e
output: Clustered Focus-Graph G_f

1 Initialization: $C \leftarrow \{\}$; `// Candidate Cluster Nodes`
2 **foreach** $t_f \in T_f$ **do**
3 create new cluster node C_c;
4 add edge $e(t_f, C_c)$ to G_f;
5 **foreach** *node* $t_n \in$ `OutNodes`(t_f, G_p) **do**
6 R_{fn} = AV allow rule for edge $e(t_f, t_n)$ in G_p;
7 **if** R_{fn} *satisfies* o_f *and* A_f **then**
8 add edge $e(C_c, t_n)$ to G_f;
9 insert C_c into C;
10 **while** *optimization possible* **do**
11 **for** $i \leftarrow 0$ **to** $size(C)$ **do**
12 **for** $j \leftarrow 0$ **to** $size(C)$ **do**
13 $outnodes_i$ = `OutNodes`(C_i, G_f);
14 $outnodes_j$ = `OutNodes`(C_j, G_f);
15 **if** $outnodes_i = outnodes_j$ **then**
16 `MergeMatching`(C_i, C_j, G_f);
17 **else if** $outnodes_i \subset outnodes_j$ **then**
18 `MergeSuperset`(C_i, C_j, τ_e, G_f);
19 **else if** $outnodes_j \subset outnodes_i$ **then**
20 `MergeSuperset`(C_j, C_i, τ_e, G_f);
21 **else if** $outnodes_i \cap outnodes_j \neq \phi$ **then**
22 `MergeOverlap`(C_i, C_j, τ_e, G_f);



(a) Before Initialization (b) After Initialization

Fig. 3. Initialization of new node clusters for focus-types t_1 and t_2. Note: t_3, t_4, and t_5 are object-types for t_1. t_4, t_5, and t_6 are for t_2.



(a) Before Optimization (b) After Optimization

Fig. 4. Clusters with matching object-types

Algorithm 2: MergeMatching

input: Cluster Nodes C_1 & C_2. Focus-Graph G_f

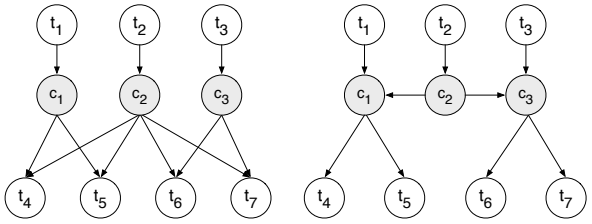
1 **foreach** *edge* $e(t, C_1) \in G_f$ **do**
2 add edge $e(t, C_2)$ to G_f;
3 remove edge $e(t, C_1)$ from G_f;
4 remove C_1 from C

The process of generating G_f is detailed in Algorithm 1. The algorithm starts by initializing a set of cluster nodes from the object-type sets of the focus-types T_f. Lines 3 and 4 create a new cluster node C_c for each of the focus-types $t_f \in T_f$, and a new edge between t_f and C_c is added to G_f. On lines 6 and 7, the AV allow rule corresponding to each edge between t_f and its out-nodes in G_p is evaluated against the given o_f and A_f. If o_f is the same as the AV rule's object-class, and A_f is within the AV rule's permissions, then a new edge from the new cluster C_c and the out-node is created in G_f. Each new cluster is then stored into C, at line 9. Figure 3 shows an example of the initialization process (assuming all AV rules satisfy o_f and A_f).

Lines 11 to 22 of Algorithm 1, involve discovering potential relations between pairs of focus-types, where each focus-type is represented by its corresponding cluster from the initialization phase, that is, each cluster represents a type's object-type set. At line 15 of Algorithm 1, it checks if the relation $\Re_m$ holds. In this scenario, Algorithm 2 is used to merge the object-type sets into one set. Figure 4 illustrates this process. Note that the number of both clusters and edges decreases, hence simplifying the resulting G_f.

At lines 17 and 19 of Algorithm 1, it checks if the relation $\Re_h$ holds. In this scenario, Algorithm 3 is used to establish a parent-child relationship within G_f. This is achieved by removing the out-edges of a parent cluster that point to the object-type set of the child cluster, then pointing the parent cluster to the child cluster. Figure 5 illustrates this process. Note that the edge-reduction threshold τ_e is passed to Algorithm 3, which allows it to measure the feasibility of establishing the parent-child relation. That is, before Algorithm 3 makes any changes to G_f, it checks if the resulting reduction in edge numbers is greater than τ_e. The edge reduction for an $\Re_h$ relation is equal to (the number of out-edges of the child cluster – 1).

At line 21 of Algorithm 1, it checks if the relation $\Re_o$ holds. In this case, Algorithm 4 is used to extract the overlapping object-types, and creates a new cluster that points to the overlap. Figure 6 illustrates this scenario. The edge-reduction threshold τ_e is passed to Algorithm 4, which allows it to measure the feasibility of establishing the $\Re_o$ relation. That is, before Algorithm 4 makes any changes to G_f, it checks if the resulting reduction in edge numbers is greater than τ_e. The edge reduction for an $\Re_o$ relation is equal to (the number of overlapping out-edges – 2). Also note that for this scenario, the



(a) Before Optimization (b) After Optimization

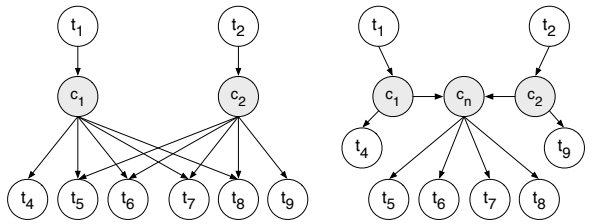
Fig. 5. Clusters with superset object-types (parent-child)

Algorithm 3: MergeSuperset

input: Cluster Nodes C_1 & C_2. Threshold τ_e, and Focus-Graph G_f

1 **if** C_1 *and* C_2 *satisfy* τ_e **then**
2 $T_o \leftarrow$ `OutNodes` (C_1, G_f);
3 **foreach** *node* $t_n \in T_o$ **do**
4 remove edge $e(C_2, t_n)$ from G_f;
5 add edge $e(C_2, C_1)$ to G_f;

number of clusters increases by 1. In our implementation, we find that the increase of clusters for a reasonable edge-reduction τ_e, is effective from a visualization point of view.



(a) Before Optimization (b) After Optimization

Fig. 6. Clusters with overlapping object-types

Algorithm 4: MergeOverlap

input: Candidate Clusters C_1, C_2, and threshold τ_e

1 **if** C_1 *and* C_2 *satisfy* τ_e **then**
2 create new cluster node C_o;
3 $T_o \leftarrow$ `OutNodes` $(C_1, G_f) \cap$ `OutNodes` (C_2, G_f);
4 **foreach** *node* $t_n \in T_o$ **do**
5 add edge $e(C_o, t_n)$ to G_f;
6 remove edge $e(C_1, t_n)$ from G_f;
7 remove edge $e(C_2, t_n)$ from G_f;
8 add edge $e(C_1, C_o)$ to G_f;
9 add edge $e(C_2, C_o)$ to G_f;

Algorithm 1 continues to run until no more feasible relations are discoverable. The run time of the algorithm depends on the size of T_f, the size of the object-type set To_f for each type $t_f \in T_f$, the size of A_f, and the value of the edge-reduction threshold τ_e.

The initialization part of the algorithm, lines 1 to 9, is $O(n^2)$ where n is the number of types in the SELinux policy G_p. Regarding the optimization part, lines 10 to 22, iterations are $O(n^2)$ where n is the number of types in G_p.

The results from applying Algorithm 1 are effective in both discovering interesting relations between focus-types, and in simplifying the visualization of analysis results. Another benefit is the ability to isolate unnecessary analysis data, such as types that are not within the focus-type set. This is achieved by allowing admins to optionally hide/reveal these types within their corresponding cluster nodes. In Section IV, we detail the results of applying the algorithm within SEGrapher.

IV. Design and Implementation

We implement our proposed clustering algorithm in a tool we call "SEGrapher". SEGrapher is based on the Java JDK 1.6, and uses the APIs provided by SETools [15] for parsing SELinux policies. Its graph drawing is based on an extended version of the open source visualization toolkit Prefuse [20]. For the purposes of this paper, we use the SELinux targeted policy binary file `policy.21`.

A. Visualization and Interactivity

SEGrapher's GUI as shown in Figure 7, contains two main panels. First, the left panel which allows the admin to control the analysis attributes, such as focus-types, object-classes, and permissions. It also has the controls for starting the analysis, and searching for types within resulting focus-graphs. Second, a right panel which shows the resulting focus-graphs of the analysis.

1) Focus-Graphs: The components of a focus-graph are visually differentiated to provide for easier policy analysis.

- **Focus-type Nodes:** Focus-types are shown as green nodes within the graph. SEGrapher also creates a new version of a focus-type in cases where it also plays the role of an object-type. The reasoning behind this is to provide a simpler focus-graph with less cycles, in cases where focus-types access other focus-types.
- **Object-type Nodes:** Object-types are shown as orange nodes in the focus-graph. Object-type nodes are hidden by default, as they are not the focus of the analysis. In cases where an object-type is also one of the focus-types, it is by default expanded and visible.
- **Cluster Nodes:** The proposed clustering approach in Section III-A results in cluster nodes that become part of the focus-graph. A cluster node is shown in black color, and shows a label which indicates the number of object-type nodes it points to. Admins can also expand/hide object-type nodes for a cluster node, by double-clicking on the cluster node. Figure 8 shows the cluster node `C1_0` with 13 expanded object-type nodes.

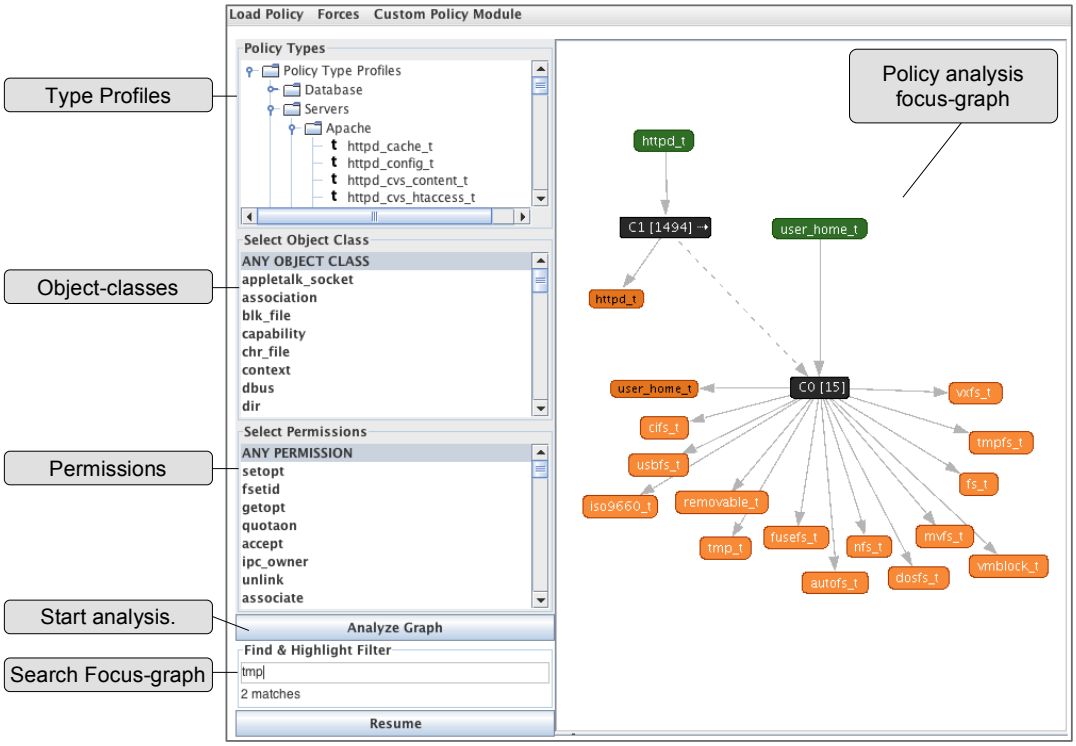


Fig. 7. SEGrapher's User Interface

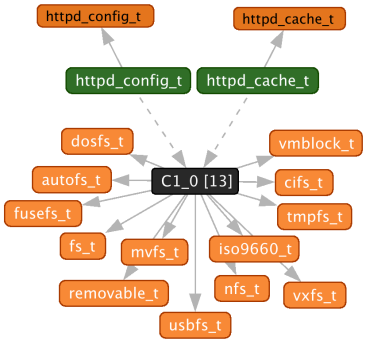


Fig. 8. Cluster C1_0 with 13 object-type nodes expanded. Shows overlapping relation (httpd_config_t $\Re_o$ httpd_cache_t)

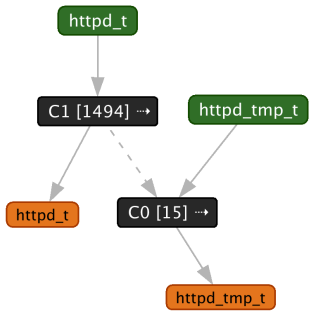


Fig. 9. Hierarchical relation (httpd_t $\Re_h$ httpd_tmp_t)

An out-edge from a node n_i to n_j indicates that n_i can access the type n_j (for the specified object-classes and permissions). If n_j is a cluster node, then n_i can access all the object-type nodes for the cluster node n_j, and all object-type nodes for clusters pointed to by n_j. For example, in Figure 9, the type `httpd_t` can access all object-type nodes for cluster `C1` and `C0`, whereas the type `httpd_tmp_t` can only access nodes of `C0`. Note that edges between clusters are visually differentiated as a dashed line.

B. Policy Analysis

To start a policy analysis, first, the admin loads a policy into SEGrapher, which is then parsed into a graph and stored into memory for future analysis. Second, the admin needs to select a set of focus-types to be analyzed, and can optionally select which object-class, and permissions to be used for filtering policy AV rules. The left panel of SEGrapher, as seen in Figure 7, shows some of the object-classes and permissions provided.

1) Focus-Types: SEGrapher allows admins to select a set of focus-types from a set of profiles we define. These profiles allow for a more intuitive method of selecting types according to their functionality, rather than searching for a specific type from within a large list of types (e.g. SELinux targeted-policy has over 1,780 types). For example, an admin can easily find the type `httpd_t` within the profile *Apache* which itself is within the profile *Servers*. Other profile examples include *Databases*, *Mail*, *Intrusion Detection*, etc. Figure 7 shows some of the profiles SEGrapher provides.

Once the admin decides on the analysis attributes, she/he can start the analysis. Following our proposed clustering algorithm

in Section III-A, SEGrapher produces a focus-graph reflecting the analysis results.

Figure 9 shows a focus-graph for focus-types `httpd_t` and `httpd_tmp_t`. This focus-graph illustrates a hierarchical relation (`httpd_t` $\Re_h$ `httpd_tmp_t`), that is, `httpd_t` has access to all object-types that {`httpd_tmp_t` has access to. This is reflected through the cluster nodes `C1` and `C0`, where `httpd_t` points to `C1` which in turn points to `C0`, whereas `httpd_tmp_t` only points to `C0`.

Another example of a focus-graph is shown in Figure 8, which shows an overlapping relation (`httpd_config_t` $\Re_o$ `httpd_cache_t`). The overlapping accesses between `httpd_config_t` and `httpd_cache_t` are clearly captured within the cluster `C1_0`.

Discussion: SEGrapher does not directly use type *attributes* (used to group similar types within SELinux policies), but rather generates its own clusters. SEGrapher already has support for attributes, because attributes are handled identically to types within AV rules. Hence, attributes are eventually represented via our clusters. Attributes are also not as reliable, because not all types are associated to an attribute, hence clustering is required.

V. Related Work

Well known SELinux policy analysis tools include APOL [5], SLAT [6], PAL [7], and Gokyo [21]. Tresys Technology developed the APOL tool, which is used to analyze SELinux policies. It provides a wide range of features including domain transition analysis, direct and transitive information flow analysis, and type relationship analysis. APOL requires a strong understanding of SELinux policies and the involved attributes, and requires a fair set of skills to perform proper policy analyses. Results in APOL are text-based, and in many cases unmanageable due to large result sets. Our tool, SEGrapher, provides easy mechanisms for analyzing policies, and visualizes the result set in a simple manner.

SLAT (Security Enhanced Linux Analysis Tool), follows an approach similar to our's, in that it represents a policy as a directed graph. SLAT represents nodes as security-contexts, and edges as the permissions on certain object-classes. The focus of SLAT is on information flow, which can be detected by traversing the policy graph. SEGrapher extends on this, by generating a *cluster-based* graph, representing analysis results, and dramatically reducing the complexity of the graph. It then is able to present simplified analysis results.

PAL (Policy Analysis using Logic-Programming), uses a logic-programming approach for analyzing SELinux policies. It follows the same model as SLAT, but provides a more extensive query set to admins. Similar to SLAT, PAL does not provide visualized analysis results, and is not able to discover inherent relations between multiple types, but is rather limited to answering direct queries.Both SLAT and PAL require a strong understanding of SELinux to generate strong queries that result in meaningful results.

Jaeger et al. [21], developed a tool called Gokyo, mainly used for checking the integrity of a proposed trusted computing base (TCB) for SELinux. Integrity checks ensure that no types outside the TCB can write to types within the TCB, and no types inside the TCB can read from those outside of it. Gokyo uses a graphical access control model for representing policies. Gokyo is limited to the proposed TCB, and does not provide "on the fly" policy analysis, nor does it allow admins to interact with the resulting analysis results.

Xu et al. [22], proposed a visualization-based policy analysis framework for analyzing security policies using semantic substrates and adjacency matrices. The framework allows admins to run visualization-based queries on a policy base to find possible policy violations. However, their framework is limited to a small set of queries, and the visualization results can be difficult to interpret and understand.

VI. Conclusion and Future Work

In this paper, we presented our visualization-based SELinux policy analysis tool, SEGrapher. SEGrapher implements a proposed clustering algorithm that generates cluster-based focus-graphs. Focus-graphs represent policy analysis results, and are dramatically simplified by the use of clusters. Clusters represent sets of object-types that are accessed by certain subject-types. Using SEGrapher, we are able to discover new interesting relations between multiple types at once, such as hierarchical, matching, and overlapping relations.

Future work will include further focus on information flow aspects of a policy, additional methods of visualization, customizable type profiles, and the ability to analyze rules on types that do not exist within the SELinux policy.

Further more, we plan on introducing and implementing a risk model which will be incorporated into SEGrapher. This model will allow administrators to assess the risks tied to new policy rules based on existing ones. We can achieve this via a similarity-based approach that's used to identify sets of nearest-neighbors for subject-types within newly introduced policy rules, i.e. types that are most similar to the types within the new rules. Similarities can be measured by investigating the object-types, object-classes, and permissions accessed by various subject-types. The nearest-neighbor rules are then used to assist admins in identifying potential risks attached to the new rules. To continue with our visualization-based approach, we will introduce visual cues that assist admins in easily assessing the risks of new rules.

Acknowledgements

This work was funded by the National Science Foundation (NSF-CNS-0831360, NSF-CNS-1117411) and Google Research Award.

References

[1] Security Enhanced Linux, "http://www.nsa.gov/research/selinux."

[2] Vincent Danen, "Introduction to SELinux: Don't let complexity scare you off," *http://www.techrepublic.com/blog/opensource/introduction-to-selinux-dont-let-complexity-scare-you-off/2447*.

[3] LWN.net, "Quotes of the week," *http://lwn.net/Articles/179829/*.

[4] Kernel Trap, "SELinux vs. OpenBSD's Default Security," *http://kerneltrap.org/OpenBSD/SELinux_vs_OpenBSDs_Default_Security*.

[5] Tresys Technology, "APOL," *http://oss.tresys.com/projects/setools*.

[6] MITRE , "SELinux Analysis Tools (SLAT)," *http://www.mitre.org/tech/selinux/*.

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

[8] I. Herman, I. C. Society, G. Melancon, and M. S. Marshall, "Graph visualization and navigation in information visualization: a survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, pp. 24–43, 2000.

[9] R. F. Erbacher, "Intrusion behavior detection through visualization," *SMC03 Conference Proceedings 2003 IEEE International Conference on Systems Man and Cybernetics Conference Theme System Security and Assurance Cat No03CH37483*, vol. 3, pp. 2507–2513, 2003. [Online]. Available: \url{http://ieeexplore.ieee.org/iel5/8811/27877/01244260.pdf?tp=&arnumber=1244260&isnumber=27877}

[10] T. Tran, E. Al-Shaer, and R. Boutaba, "Policyvis: firewall security policy visualization and inspection," in *Proceedings of the 21st conference on Large Installation System Administration Conference*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1:1–1:16. [Online]. Available: http://portal.acm.org/citation.cfm?id=1349426.1349427

[11] W. Yurcik, "Tool update: visflowconnect-ip with advanced filtering from usability testing," in *Proceedings of the 3rd international workshop on Visualization for computer security*, ser. VizSEC '06. New York, NY, USA: ACM, 2006, pp. 63–64. [Online]. Available: http://doi.acm.org/10.1145/1179576.1179588

[12] Justin R. Smith, Yuichi Nakamura, and Dan Walsh, "audit2allow," *http://linux.die.net/man/1/audit2allow*.

[13] Yuichi Nakamura, "SELinux Policy Editor(SEEdit) Administration Guide 2.1," *http://seedit.sourceforge.net/doc/2.1/tutorial/node9.html*, February 2007.

[14] Red Hat, Inc., "Red Hat SELinux Guide, Chapter 8. Customizing and Writing Policy," *http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/4/html/SELinux_Guide/selg-section-0120.html*.

[15] Tresys Technology, "Setools: Policy analysis tools for selinux http://oss.tresys.com/projects/setools."

[16] Hitachi Software, "Seedit: Selinux policy editor http://seedit.sourceforge.net."

[17] Samb , "Samba Server," *http://www.samba.org/samba*.

[18] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '99. New York, NY, USA: ACM, 1999, pp. 230–237. [Online]. Available: http://doi.acm.org/10.1145/312624.312682

[19] M. R. McLaughlin and J. L. Herlocker, "A collaborative filtering algorithm and evaluation metric that accurately model the user experience," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '04. New York, NY, USA: ACM, 2004, pp. 329–336. [Online]. Available: http://doi.acm.org/10.1145/1008992.1009050

[20] Jeffrey Heer , "Prefuse (Java)," *http://prefuse.org*.

[21] 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 - Volume 12*. Berkeley, CA, USA: USENIX Association, 2003, pp. 5–5. [Online]. Available: http://portal.acm.org/citation.cfm?id=1251353.1251358

[22] W. Xu, M. Shehab, and G.-J. Ahn, "Visualization based policy analysis: case study in selinux," in *Proceedings of the 13th ACM symposium on Access control models and technologies*, ser. SACMAT '08. New York, NY, USA: ACM, 2008, pp. 165–174. [Online]. Available: http://doi.acm.org/10.1145/1377836.1377863

[23] MITRE, "Polgen: Guided auto-mated policy development. http://www.mitre.org/tech/selinux."