CASPR: Context-Aware Security Policy Recommendation

Lifang Xiao*, †, Hanyu Wang*, †, ✉, Aimin Yu*, †, Lixin Zhao*, Dan Meng*, †
*Institute of Information Engineering, Chinese Academy of Sciences, China
†School of Cyber Security, University of Chinese Academy of Sciences, China
{xiaolifang, wanghanyu1999, yuaimin, zhaolixin, mengdan}@iie.ac.cn

***Abstract*—Nowadays, SELinux has been widely used to provide flexible mandatory access control and security policies are critical to maintain the security of operating systems. Strictly speaking, all access requests must be restricted by appropriate policy rules to satisfy the functional requirements of the software or application. However, manually configuring security policy rules is an error-prone and time-consuming task that often requires expert knowledge. Therefore, it is a challenging task to recommend policy rules without anomalies effectively due to the numerous policy rules and the complexity of semantics. The majority of previous research mined information from policies to recommend rules but did not apply to the newly defined types without any rules. In this paper, we propose a context-aware security policy recommendation (CASPR) method that can automatically analyze and refine security policy rules. Context-aware information in CASPR includes policy rules, file locations, audit logs, and attribute information. According to these context-aware information, multiple features are extracted to calculate the similarity of privilege sets. Based on the calculation results, CASPR clusters types by the K-means model and then recommends rules automatically. The method automatically detects anomalies in security policy, namely, constraint conflicts, policy inconsistencies, and permission incompleteness. Further, the detected anomalous policies are refined so that the authorization rules can be effectively enforced.**

The experiment results confirm the feasibility of the proposed method for recommending effective rules for different versions of policies. We demonstrate the effectiveness of clustering by CASPR and calculate the contribution of each context-aware feature based on SHAP. CASPR not only recommends rules for newly defined types based on context-aware information but also enhances the accuracy of security policy recommendations for existing types, compared to other rule recommendation models. CASPR has an average accuracy of 91.582% and F1-score of 93.761% in recommending rules. Further, three kinds of anomalies in the policies can be detected and automatically repaired. We employ CASPR in multiple operating systems to illustrate the universality. The research has significant implications for security policy recommendation and provides a novel method for policy analysis with great potential.

I. Introduction

With the rapid development of information technology, system security issues are becoming increasingly prominent, and various attacks are constantly occurring, causing significant losses to individuals and organizations. Therefore, it is crucial to strengthen the security of operating systems. Security Enhanced Linux (SELinux) [3], an effective kernel security enforcement module designed to enhance the security of Linux systems, provides a robust and flexible mechanism for resources and operations. SELinux is implemented based on security labels, defined for each user, application, process, and file in systems. It controls interactions between entities through security policies to achieve the least privilege to prevent unauthorized access. SELinux policy has complex semantics and numerous rules, containing approximately one hundred thousand rules to implement fine-grained access control [30].

However, the large number of policy rules and complex semantics make policy customization in SELinux time-consuming and error-prone. Currently, most policies are configured manually, which requires not only expertise and experience but also an in-depth understanding of access behaviors, which undoubtedly increases the time cost of policy customization. More importantly, inappropriate policies will cause security problems. The access control policy determines the operating authority of the subject to the object. If the policy rules are over-privileging, it will lead to unauthorized access, data leakage, and malicious attacks. Conversely, if they authorize inadequately, it will affect the normal operations of the system. Therefore, configuring rules with reasonable privileges is an urgent and vital task. Undoubtedly, policy anomaly detection and rule recommendation [45], [52], [55] help to optimize policies and automatically generate appropriate security policies.

To cope with the challenges, a large number of studies are related to policy analysis, anomaly detection, and automatic recommendation of policy rules. Previous approaches of policy analysis mainly detect anomalies such as misconfiguration [12], [58], over-granting [43], inconsistency [40], [57] and policy conflicts [17], [53]. Additionally, a significant number of approaches to generating policy rules have been proposed, such as natural language processing and pattern learning to mine important information from logs or rules and recommend policy rules automatically [21], [24], [29].

While previous policy analysis can automate the iterative process of policy recommendation, it is not suitable for newly defined types that lack applicable policy rules. When new software is installed on the system or a new type is configured by the system administrator, it is difficult to generate rules for it without current policy rules to use as a reference. To address this issue, we incorporate supplementary context-aware infor-

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240418
www.ndss-symposium.org

mation, which is used as features to analyze privilege similarity for recommending policy rules. The integration of multiple features not only enables the automation of recommending rules for newly defined types but also enhances the precision of recommending rules for existing types.

In this paper, we propose a context-aware security policy rule recommendation method, termed as CASPR, which recommends rules based on the privilege calculation and detects policy anomalies automatically. The context-aware information includes policy rules, audit logs, file locations, and attribute information. Our observation is that context-aware information plays an important role in rule recommendation. In particular, domains or object types with related context-aware information are granted the same privileges. To this end, we design CASPR for privilege computation and policy recommendation. CASPR uses context-aware information as features and domains and object types as samples. We train a K-means model with a feature matrix of privilege similarity. To address the issue of a huge number of object types in SELinux, we adopt a secondary clustering approach when clustering the objects of the policy. CASPR not only recommends rules but also performs anomaly detection to ensure effective implementation of the policy.

To evaluate the performance of CASPR, we capture and analyze context-aware information in SELinux. In addition to the types associated with the rules, we also add new types as samples. We demonstrate the clustering effect of domains and object types and the contribution value of each context-aware feature in the clustering using SHapley Additive exPlanations(SHAP) [26], which explains model output by assigning importance values to features. It indicates that the security context-aware information is critical, as it has a favorable impact on the policy recommendation. The average accuracy of CASPR to recommend policy rules is 91.582% on average, which exceeds other rule recommendation methods. Besides, CASPR achieves the F1-score of 93.761% on average, which indicates that CASPR has few false positives and false negatives. We also employ CASPR in multiple operating systems to illustrate its universality. The results confirm the accuracy of the recommendation rules based on the clustering results obtained from measuring privilege similarity.

In summary, the contributions of this paper are summarized as follows:

- *Privilege calculation based on context-aware information.* We innovatively introduce context-aware information for privilege calculation and integrate them, including policy rules, file locations, audit logs, and attribute information.
- *Rule recommendation and anomaly detection.* This paper proposes CASPR, a rule recommendation and anomaly detection method, which establishes a privilege similarity matrix based on context-aware information to cluster the domains and object types and automatically recommends policy rules. Additionally, the method can detect and refine anomalies, including constraint conflicts, policy inconsistencies, and permission incompleteness.
- *Experimental effectiveness.* In this paper, we perform experiments based on large-scale context-aware information. The experiments prove that the method improves the accuracy of policy recommendations with higher adaptability and feasibility.

II. Background & Definition

A. Overview of SELinux

Access control is a security mechanism typically implemented using the Linux Security Module (LSM) framework [48], which manages and restricts access to resources by users and processes in the system to avoid unauthorized access, destruction, and tampering of the system. Classical access control models are Discretionary Access Control (DAC) [33], [37], Mandatory Access Control (MAC), Role-Based Access Control (RBAC) [15], [35], [38] and Attribute-Based Access Control (ABAC) [33], [54]. SELinux [2] is a kernel-level MAC security mechanism that defines the access privileges of each entity in the system and uses security policies to control interactions between these entities. When an access request is initiated, the kernel interface is invoked and a database of policies in the kernel determines whether it is allowed. If the mode of SELinux is enforcing, all access is denied by default except when the relevant rules are defined in the policy.

The access control policy in SELinux is based on the security labels of subjects and objects. Each subject and object has an associated security label. The security label *<USER:ROLE:TYPE[LEVEL]>* contains three fields, user, role, and type identifier. Among them, type is the core of Type Enforcement (TE) [5], [20] rules and the privileges of security policy rules are assigned based on type. Types include subject types (domains) and object types. Type, role, and user are legal security labels only when associated. In security policies, declarations and rules define type enforcement policies together. The rules include massive access vector rules and type rules. Type rules make labels legal and facilitate policy writing. SELinux policy is large-scale and complex, composed of numerous policy modules. The policy modules form the policy source files, which are compiled into a binary file by the policy compiler and loaded into the kernel through the policy loading function to implement access control. The security policy implements privilege changes by tuning the mandatory rules, and each access request must have explicit rules.

B. Definition of Context-aware Information and Privilege Similarity

In this paper, the context-aware information related to policy recommendations includes security policy rules, file locations, audit logs, and attribute information. This section introduces the definition of these context-aware information and privilege similarities.

1) Attribute Information: If the type identifier is used to reference a subject, it specifies its subject type, that is a domain. If the type identifier is used to reference an object, it specifies its object type. Attributes represent sets of domains or object types. The mapping relationship can be expressed as $attr_s = \{d_i\}$ where $attr_s$ denotes a subject attribute including domains and d_i means a domain in the attribute. Similarly, $attr_o = \{t_i\}$ where $attr_o$ represents an object attribute consisting of various object types t_i. This abstraction based on attributes facilitates the management of complex security policies by grouping types with identical privileges together, thereby simplifying the assignment and enforcement of privileges. Attributes provide a more manageable extensible policy framework for large systems with numerous types.

2) Security Policy Rules: We collect the rules in the type enforcement policy in SELinux. The rules can be represented as a five-tuple $r_e =< a, d, t, c, \{p_i\} >$, where a indicates allow or neverallow, d is the domain, t is the object type, c is the object class, $p_i \epsilon p$ represents permissions. In this paper, we focus on analyzing the allow rules, and when a indicates allow, r_e indicates that the subject of domain d is allowed to have p_i permission on the c class of the object of type t. The type enforcement policy in SELinux can be represented as a set of rules. For example, *allow passwd_t passwd_file_t : file { ioctl read write create getattr setattr lock append map unlink link rename open } ;* . This rule indicates that subjects with *passwd_t* domain are assigned these permissions to the *passwd_file_t* object type with *file* class.

The SELinux policy contains rules defined based on attributes and rules defined based on types. According to the definition of attributes, the attribute-based rules in the policy can be converted to type-based rules, which form is $r =< a, attr_s, attr_o, c, \{p_i\} >$ to $r =< a, d|attr_s, t|attr_o, c, \{p_i\} >$. In addition to attributes, we also expand the set of permissions. We make each rule $r =< a, d, t, c, \{p_i\} >$ represented in the form of multiple rules $r_a =< a, d, t, c, p >$, where $p \epsilon \{p_i\}$. This rule specifies one permission for a domain, which facilitates our analysis. Figure 1 represents a schema of rules expansion and statistical privileges.

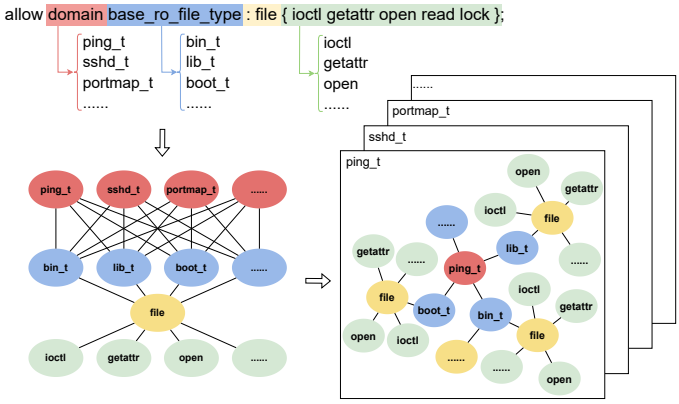

Fig. 1: Process of SELinux Policy.

3) File Locations: In SELinux, there is a file mapping the relationship between security labels and locations. The relationship can be expressed as $label \rightarrow \{location_i\}$. A label corresponds to at least one location. The mapping relationship between labels and file locations is the implementation of associating specific files with types, which serves as the basis for fine-grained access control.

4) Audit Logs: Audit logs [43], [50] record the operating behavior. By analyzing the logs to extract important information, it can audit and trace the violation of user or software operations. In addition, access control logs are important evidence for policy customization and optimization to guarantee the normal operation of software and applications and prevent anomalies caused by under-privilege. For example, the logs of type Access Vector Cache (AVC) record detail information including the domains, object types, object classes, and permissions. As shown in Figure 2, the logs record two violation operations, the subject *user_t* was rejected when performing open and read operation on *auditd_log_t*. Through processing, the access requests are extracted in logs.

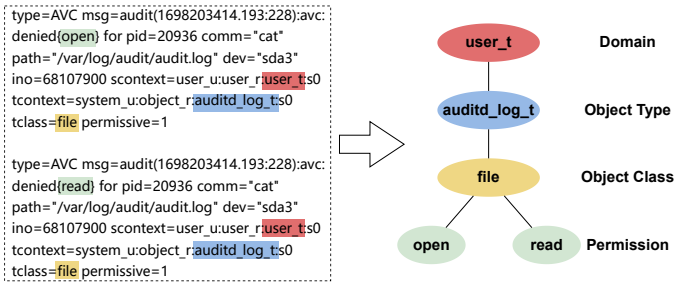

Fig. 2: An Example of Extracting Policy Rules from Logs.

5) Privilege Similarity: A parsed rule, of the form $r_a =< a, d, t, c, p >$, is regarded as a privilege. Different subjects that have the same permission on the same object have identical privileges. Similarly, different objects performed the same operations by the same subjects have identical privileges. It is worth noting that having identical privileges is not equivalent to having the same privilege set. The more identical the privileges they contain, the greater the similarity of the privilege sets is. The granted privileges are included in the rules. To simplify the description, we refer to the similarity of privilege sets as privilege similarity. In addition to policy rules, there is other context-aware information related to privileges, including file locations, audit logs, and attribute information. The detailed reasons are explained in Section IV-A1.

C. K-means Clustering Model

The K-means model is a widely used unsupervised learning algorithm for clustering analysis. Its main objective is to partition a dataset into K predefined clusters, maximizing the similarity among data points within the same cluster while minimizing the similarity between data points in different clusters. K-means algorithm needs to select K initial centroids randomly. Afterward, it updates the centroids by calculating the mean of all data points in each cluster. K-means repeats the assignment and update phases until the centroids do not change significantly or a specified number of iterations is reached.

The K-means algorithm is computationally efficient and suitable for handling large-scale datasets and can accomplish a large number of data clustering tasks in a short period of time. For policies with massive domains and types, it is necessary to efficiently compute their privilege similarity and recommend policy rules as soon as possible to prevent attackers from performing illegal operations before fixing the policies. Therefore, the K-means model is particularly suitable in this case.

III. Problem Statement and Threat Model

Improper configuration of SELinux policy, including over-privileging and under-privileging, poses significant security risks to the SELinux-enforced system. Over-privileged policy rules allow attackers to execute malicious operations, as demonstrated by the exploitation of vulnerability CVE-2019-13272 [1]. The vulnerability arises from improper handling of the *ptrace_traceme* function in the parent-child process relationship, enabling attackers to hijack a child process by

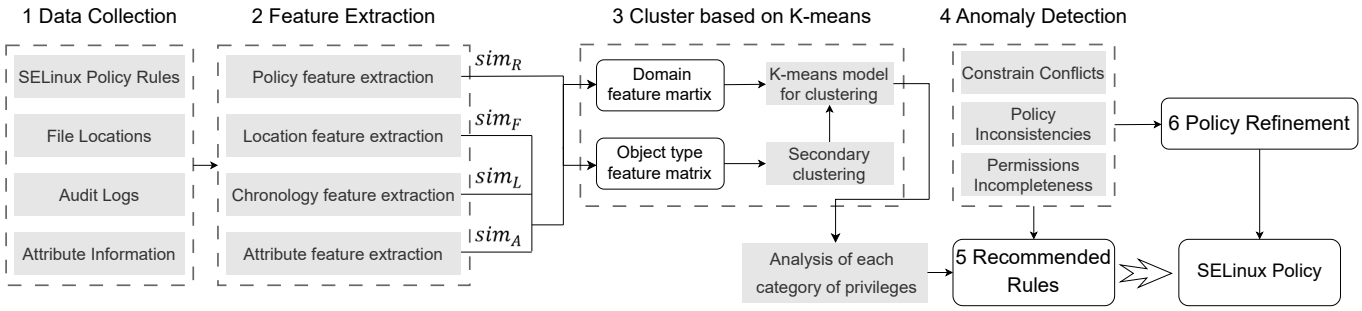

Fig. 3: The workflow of CASPR.

gaining control of the parent process. They can then manipulate the child process through the *ptrace* trace function, which is typically denied by standard SELinux policies. However, because the *deny_ptrace* boolean was not enabled, leaving the system vulnerable to attack.

On the other hand, overly strict policies can disrupt normal operations. For example, in a few cases, the administrator changes the default port of the Apache HTTP server from port 80 to port 3131 or stores content in a non-standard directory. It causes the new directory and port have not been appropriately labeled for *httpd* access. As a result, the server may fail to start or return a "*403 Forbidden*" error due to SELinux policies denying access. [41]

Given these challenges, this paper focuses on the potential risks associated with the recommendation of policy rules in SELinux. CASPR is designed to identify the over-privileging and under-privileging rules in policy.

Threat Model. We consider that an attacker attempts to use the expanded attack surface to perform illegal operations if the policy is over-privileged. However, if it is under-privileged, the operation of normal function is limited. We assume that an attacker is enforced with security policies and cannot bypass the security measures. To cope with these potential problems, a powerful policy recommendation mechanism is needed.

IV. CASPR Design

CASPR is a large-scaled policy recommendation architecture using security context-aware information in SELinux. CASPR analyzes each context-aware feature and clusters the domains and object types with identical privileges to recommend rules. Additionally, CASPR refines policy rules after detecting anomalies. Figure 3 shows the workflow of CASPR. We describe the three components in the architecture of CASPR, that is, privilege computation based on context-aware information, rule recommendation based on secondary clustering, and policy refinement after detecting anomalies.

A. Context-aware Feature Computation

In this section, we analyze the rationality for the selection of context-aware information for computing privilege similarity. Then, we explain how the privilege similarity is calculated for each feature and how the feature matrix is constructed based on the similarity computation for these features.

1) Feature Selection: CASPR extracts features including policy feature, location feature, chronology feature, and attribute feature from context-aware information. The following part explains the reasons for selecting these features as the judgments of identical privileges.

Policy Feature. Due to the fact that privileges are granted by policy implementation in SELinux, the most significant indicator for identical privileges is the security policy rules.

Location Feature. The path dependency of files indicates that they have the identical privileges. For example, subjects of domain $secadm\_t$ are authorized to read $audit.log$ of type $auditd\_log\_t$. Meanwhile, they are also authorized to search parent directories with path dependencies, including $\{getattr, search, open\}$ permissions on directory $/var$ of type var_t, directory $/var/log$ of type var_log_t and directory $/var/log/audit$ of type $auditd\_log\_t$.

Chronology Feature. In the audit logs, each of the logs has a timestamp field indicating when the behaviors occurred. The behaviors which occurred simultaneously over a while have been generated by the same event. These behaviors are allowed or denied at the same time. For example, when a network activity occurs on a host, this event consists of multiple behaviors such as inbound and outbound network connections to the system, transmission and receiving of network packets, and authentication of web credentials. If access is required, it is not possible to authorize only part of these behaviors. Therefore, we consider the chronological information on behavior in audit logs as a feature for identical privileges.

Attribute Feature. All types that belong to the same attribute have the privileges granted by this attribute. Definitely, types contained in the same attribute have identical privileges. For example, $su\_domain\ \_type$ is an attribute with system administrative privileges. It contains three domains, $auditadm\_su\_t$, $secadm\_su\_t$, and $sysadm\_su\_t$, which are used for monitoring, authorization, and system management, respectively. When the attribute su_domain_type is authorized to read files of type $security\_t$, the domains in this attribute can also read these files, so they have identical privileges. The analysis results indicate that other domains that are not in the same attribute do not possess this privilege. For instance, subjects of domain $systemd\_passwd\_agent\_t$ cannot read files of type $security\_t$, even if it is also a label for managing system activities.

It is worth noting that some samples do not have all the context-aware features. For example, not all types belong to

an attribute and CASPR cannot determine identical privileges based on the feature of attributes. In this case, CASPR calculates privileges based on other features. In addition, aimed at the newly defined types that have no related rules in the system, we can recommend rules based on other security context-aware features besides rules to recommend policy rules.

2) Feature Computation: When constructing the feature matrix A, we arrange the types as rows and columns, and the value in the row i and column j of the matrix a_{ij} denotes the similarity of privileges between the i and j types. The feature matrix built based on these features is the input of the clustering algorithm. We compute the policy features before considering other features.

For the similarity of security policies, taking the clustering of domains as an example, when a subject is granted permission for a particular object, the corresponding value in the feature vector is 1. If the subject does not have permission to the object, the corresponding value in the feature vector is 0. We calculate the value corresponding to two samples in the feature matrix according to the similarity of feature vectors by the Jaccard index. The similarity of vector a and vector b is calculated as follows:

$$sim(a,b) = \frac{len(a\&b)}{len(a|b)} \tag{1}$$

It is similar to the clustering for object types. We use the subjects that can be accessed and the permissions they are authorized to perform as feature vectors. Especially, for the newly defined types, the contribution of policy rules to the value in the feature matrix is small, but other features indicate identical privileges. In this condition, CASPR recommends policy modifications based on the differences in their rules to improve the policy.

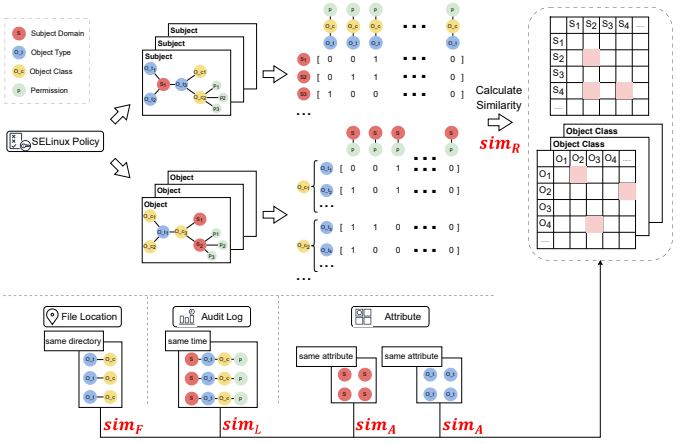

Fig. 4: Feature Extraction.

When calculating the privilege similarity about the features including location feature, chronology feature, and attribute information, we consider the privilege similarity in a simpler computational manner. The value of sim is set to 1 if the collected data indicates that the two samples exhibit similarity in the dimensions of the features, analyzed in Section IV-A1, including path dependency or frequent simultaneous access or belonging to the same attribute. Conversely, if such indicators are absent, sim is set to 0.

We use the similarity of domains and object types on different features as the criterion for determining the privilege similarity. CASPR adopts a rigorous approach to creating the feature matrix, which is shown in Figure 4. Since SELinux implements fine-grained access control, of which there are numerous domains and object types, the feature matrix computed based on privileges is very sparse, which affects the performance of clustering. Therefore, it is important to perform feature scaling, which standardizes and normalizes the feature matrix when generating it. It involves converting the privilege similarity on each feature to a distribution with an arithmetic mean of 0 and a standard deviation of 1 as follows:

$$x_{standard} = \frac{x - \mu}{\sigma} \tag{2}$$

where μ is the arithmetic mean and σ is the standard deviation of the dataset. The standardization eliminates the difference in scale between different features. Afterwards, the data is scaled to the range of 0 to 1 for K-means computation as follows:

$$x_{normal} = \frac{x - min(x)}{max(x) - min(x)} \tag{3}$$

B. Rule Recommendation Based on Clustering

In this section, we introduce the clustering of domains and object types with identical privileges to determine the recommended rules based on the clustering results.

1) Cluster Algorithm: After generating the feature matrix according to the privilege similarity calculated in the dimensions of selected context-aware features, CASPR uses the K-means model for clustering the domains and object types respectively. The clustering is shown in the algorithm 1.

Subjects Clustering. For clustering the domains, we first generate a feature matrix according to the privilege similarity calculated by the policy feature. Afterwards, we iteratively update this feature matrix to superimpose the similarity in the dimensions of other features and perform feature scaling. Taking the feature matrix as input, CASPR uses K-means model, which is a cluster analysis algorithm with iterative solving, to cluster samples with identical privileges. The initial clustering center is determined by the privilege sets defined in the known policy rules. By clustering the data and calculating the cluster centers, samples are assigned clusters by the distance from the sample points to the cluster centers. The K-means model calculates that the feature vectors of the privilege sets of subjects that are classified into the same category have higher similarity, which means that they have more identical privileges to the objects.

Secondary Clustering of Objects. The clustering of objects is similar to subjects. The difference is that the clustering of objects is more complex. Specifically, if two objects share the same object type but are contained in different object classes, they will have different privileges. For example, for an object of the same object type bin_t, the subjects of $user\_t$ have the permissions of map and $execute$ on $file$ class but have no permission on directories of the dir class. Therefore, it is vital to analyze the privileges of each object type and class. This leads to a huge sample number in object clustering. Therefore, we set a secondary cluster. The large-scale dataset is divided into several subsets. The clustering computation is

Algorithm 1: Domains and object types clustering.

Input: Policy, Location, Log, User, Attribute
Output: Clustering Results of subjects and objects.

```
// capture features
rules← GetPrivilege(policy);
relationship_location← GetNearFile(Location);
relationship_chronology← GetEvent(Log);
relationship_type← GetAttribute(Attribute);
// clustering of domains
subject← GetDomain(policy+added_domain);
matrix1[:] = 0;
for rule ϵ rules do
    subject_vector[subject].update(rule[object_type]+
     rule[object_class]+rule[permission]);
end
for s1 ϵ subjects do
    for s2 ϵ subjects do
        matrix1[s1][s2].add(sim(s1_vector,
         s2_vector));
    end
end
for relationship(s1, s2) in relationship do
    matrix1[s1][s2].add(simi);
end
matrix1.scaling();
results_subject← K-means(martrix1);
// clustering of object types
objects← GetObjectType(rules+added_type) ;
matrix2[:] = 0;
for class ϵ rules[object_class] do
    for rule ϵ rules do
        object_vector[object].update(rule[domain]+
         rule[permission]);
    end
    for o1 ϵ objects do
        for o2 ϵ objects do
            matrix2[o1][o2].add(sim(o1_vector,
             o2_vector));
        end
    end
    for relationship(o1, o2) in relationship do
        matrix2[o1][o2].add(simi);
    end
    matrix2.scaling();
    results_object ← K-means(martrix2);
end
return results_subject, results_object;
```

performed on different nodes so as to reduce time complexity and computational load. In the primary cluster, we cluster the objects according to object class, such as directory, file, process, and so on. In the secondary cluster, based on the result of the primary cluster. We cluster object types in the same object class. In this way, objects that are clustered into the same category are accessed by more identical subjects and granted more identical privileges.

2) Rule Recommendation: We statistic the privilege sets for each category of domains and object types respectively after clustering, and the majority of samples in the same category have identical privileges. For domains, their privileges can be expressed as permissions, object types, and object classes, which can be accessed by most of the subjects of the domain in this category. For object clustering, it is the set of domains and the permissions performed by the subjects. The subjects are authorized to perform operations on most of the object types in this class. Figure 5 depicts the procedure of rule recommendation.

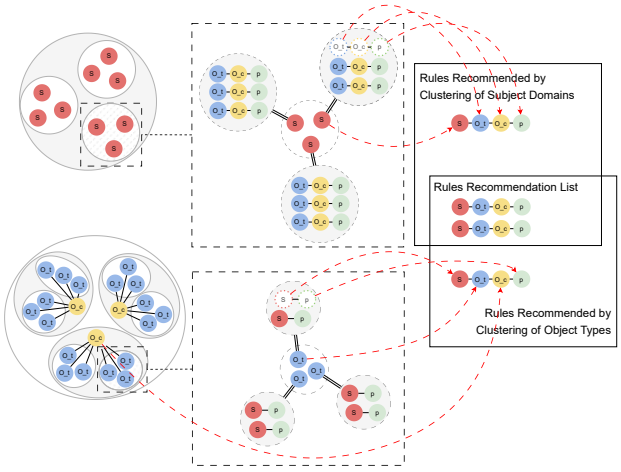

Fig. 5: Rule Recommendation.

After obtaining the results of the clustering, samples in the same category have identical privileges. However, taking domain clustering as an example, domains clustered in a category just have multiple identical privileges, not the same privilege sets. Thus there is a case where one sample does not have the privileges granted in the policy that the majority of the other samples in the category have. However, it is clustered with other domains due to the identical privileges in other feature representations. In this case, this domain should be granted based on the calculation of CASPR. To complete the authorization, it is necessary to recommend the missing rules. For example, among multiple domains, only domain s_1 does not have read permission and the rest of the domains do. At this point, we generate a rule that grants read permission to domain s_1. Similarly, if the object type lacks the relevant rules, we recommend them in the same way.

Theoretically, all samples in the same category cannot have exactly identified privilege sets. Because SELinux performs the principle of least authorization, if two types have exactly identical privileges, it will cause privilege confusion and potential security vulnerabilities. Therefore, it is not appropriate to blindly add the recommended rules to the policy based on the similarity of privilege sets while ignoring the privilege differences between the domains or object types. To eliminate the false positives, we cluster domains and object types separately and recommend two lists of rules separately by the above method. Afterwards, the intersection of the two lists is the final recommendation rules. In other words, if the rule recommendations according to the clustering of domains and object types are consistent, we generate a list of rule recommendations. Additionally, we also minimize the possibility of false positives by adopting a more strict threshold setting. Through this approach, we provide a reasonable rule recommendation method based on clustering results.

C. Anomaly Detection and Policy Refinement

The recommended rules simply determine that domains or object types in the same category have identical privileges. Since SELinux policy are complex and sophisticated, it is also imperative to take into account their integrity and availability. It is essential to complete all the necessary rules to perform the behavior according to the constraint files in SELinux, instead of adding only one single rule to execute a certain operation. However, blindly adding missing rules would lead to overly-authorization. Therefore, it is necessary to assess if the inclusion of new recommended rules in the policy will result in any anomalies. There are different configuration files in SELinux, and each file specifies different perspective of completeness. According to the configuration files, we identify grammar and semantic anomalies, including constraint conflicts, policy inconsistencies and permission incompleteness, by employing regular matching. The goal of detecting anomalies is to refine policy properly, so as to ensure that the modified SELinux policy can work effectively. The following describes these anomalies and how to patch the policy without over-authorization when they occur.

1) Constraint Conflicts: The constraints can be parsed into the following form: $< Attr, c, \{p_i\} >$, which means that only types in attribute $Attr$ have permissions of p_i on objects of class c. The reason for these anomalies is that the attribute does not contain the type of added rule. In this case, it is necessary to add the type to the attribute, but other privileges of this attribute will also be presented. Generally, if this type and other types of this attribute are clustered into a category by CASPR, we refine the policy based on the recommendation of rule modification. If they are not in the same category, there are no other privileges assigned.

2) Policy Inconsistencies: The *file_patterns.spt* file defines the policy consistencies. Specifically, it can be parsed into the following form: $< beh : [\{beh\_f\}, \{beh\_s\}] >$, which means that if a subject of domain d performs beh behavior, it needs to have privileges not only to perform beh_s on the file but also to perform beh_f on its parent directory. For example, if a subject requires to read a file, it possesses not only read permission on the file but also the necessary permissions to search the directory containing the file. We must take into account the privileges of the parent directory.

3) Permission Incompleteness: In SELinux, the permissions completeness are defined in the *obj_perm_sets.spt* file. Specifically, it can be parsed into the following form: $< c : \{beh : \{p_i\}\} >$, which means if a subject performs beh behavior on an object of class *c*, it needs to gain permissions of p_i. For example, if a subject desires to read a file and lacks the *open* permission, it is unable to read this file, regardless of possessing the *read* permission. These anomalies are caused by authorizing only part of the permissions, which prevents the operations from executing correctly. It is different from the permissions of writing and reading in the DAC mechanism in Linux. It is also necessary to consider other relevant prerequisite permissions.

V. Evaluation

In this section, we evaluate the performance of CASPR in the following aspects: whether the clustering of domains and object types is effective, whether the recommended rules are accurate, without missing or excessive authorization, whether the anomaly is detected, and whether CASPR applies to different versions of SELinux policy recommendations. We use three cases to describe the usage scenarios.

A. Data Process

1) Description of Operating System Context-aware Information List: We implement CASPR in python 3.7 and deploy it in six different versions of CentOS and Ubuntu systems. They are all enforced by SELinux and the related information is shown in Table I. Different systems enforce distinct access control policies due to variations in system resources. We illustrate the data in Table I with the example of CentOS7, which adopts 3.13.268.el7 version of SELinux and enforces the v31 version of policy. Its standard SELinux policy contains 107,834 rules. After removing duplicates, there are 99,054 rules. In the policy, some rules define the association permissions on the file system. Its purpose is to associate a file to the file system. These rules typically have the same domain and object types. These permissions are basic requirements for file system associations. Therefore, the associations related to *filesystem* are authorized. Besides, excluding these rules from clustering calculations can greatly reduce the number of domains. Afterwards, according to the attribute and type mapping relationships, the attribute-based rules are parsed to convert them to type-based rules. The parsed rules contain a significant degree of redundancy. If the domains and object types in the rules are the same, and only the permissions are different, it is feasible to combine the rules. It contains 173,514,894 privileges that are not duplicated and can be represented as $r_a =< a, d, t, c, p >$. The policy rules are associated with 823 domains and 4,279 object types in 124 object classes, which are the samples for clustering in CASPR. In addition to the policy rules in SELinux, we also obtain context-aware information. It includes 5,372 mappings of file locations, 8,748 audit logs, and 14,950 mapping relationships of attributes and types. These information is also used for clustering calculation as a feature to determine the similarity of privilege sets of types.

Operating Systems	SELinux Version	Rules	Privileges	File Locations	Audit Logs	Attribute Mappings
CentOS6	3.7.19-312.el6	304,755	128,258,009	3,904	8,255	8,814
CentOS7	3.13.1-268.el7	99,054	173,514,894	5,372	8,748	14,950
CentOS8	3.14.3-139.el8	108,038	197,389,960	5,609	8,326	13,581
Ubuntu20	2:2.20190201-8	96,758	43,008,323	4,458	6,854	9,402
Ubuntu22	2:2.20210203-10	90,772	35,880,295	4,287	7,749	8,871
Ubuntu24	2:2.20240202-1	36,400	37,213,711	4,484	7,753	8,935

TABLE I: Context-aware information list for different operating systems.

2) Train Set and Test Set Setup Instructions: CASPR employs an unsupervised clustering model, but in order to verify its effectiveness, we choose the original SELinux policy as the ground truth and divide it into train set and test set. We randomly select 90 percent of the rules to train CASPR. With the remaining 10% of the rules, we also create predefined rules for new types as our test set. We compare whether the result

of the recommended rules is consistent with the test dataset to evaluate the performance of CASPR.

3) Negative Samples Instructions: Since SELinux implements a MAC mechanism, the privilege of an access request is authorized only based on the policy. If a relevant rule exists in the policy, the access request is allowed. Otherwise, it is denied. Therefore we generate a comprehensive set of rules by matching domains and object types individually. If the rule is included in the policy, we consider it as a positive sample. Conversely, if the rule is not included in the policy, it is considered as a negative sample. Additionally, we randomly generate rules to test the effect of CASPR detecting anomalies and refining the policy.

In the following sections, we focus on the performance of policy clustering and rule recommendations for CentOS7.

B. Clustering of Subjects and Objects

1) Clustering Result: To illustrate the feasibility of CASPR for policy recommendation, we analyze the clustering scatterplots for samples including domains and object types. We first analyze the privileges of domains in the policy rules and the newly added domains, which are a total of 885 domains. According to the performance at different thresholds analyzed in Section V-C3, we cluster domains into 88 categories.

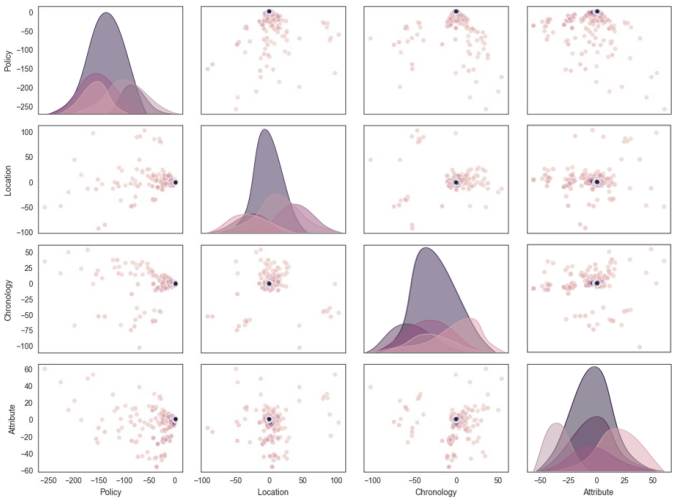

Fig. 6: Cluster Result of Domains.

Since we take a multi-dimensional feature matrix in clustering, it is blurry for the scatterplot generated from projecting all the sample points onto a plane in a two-dimensional space. Therefore, the direct analysis cannot produce satisfactory results. Therefore, we perform principal component analysis (PCA) dimensionality reduction in clustering, which involves calculating the covariance matrix and eigenvalues and eigenvectors. Then, we choose the principal components, which are the eigenvectors that correspond to the biggest eigenvalues. The scatterplot of the cluster of domains is shown in Figure 6.

The horizontal and vertical coordinates in the figure represent the principal components, which correspond to the features of policy, location, chronology, and attribute. On the diagonal is the variance of each principal component represented as a histogram. *PC1* is the principal component corresponding to the policy rule feature and has a wider distribution on the image, indicating that the variance of the data in the *PC1* direction is larger. The scatterplot represents the projection of data points onto two principal components. Through the scatterplot, it is clear that there are multiple domains clustered into one category based on selecting identical privileges as features. It proves the feasibility of our recommendation rules based on identical privileges.

Object Class	Number	Object Class	Number	Object Class	Number
socket	5,437	filesystem	32	service	48
file	3,555	context	13	passwd	107
dir	2,494	msg	54	peer	820
process	829	security	2	association	818
netif	2	process2	39	capability	809
key	530	system	9	database	55
packet	224	msgq	62	memprotect	87
dbus	277	nscd	86	bpf	5
fd	412	node	1	lnk_file	3,166
sem	817	shm	818	unknown	1,029

TABLE II: Statistics of object types in each object class.

We use the same method to analyze the object types in each object class such as file or process. The clustering results obtained for some classes are similar to the clustering results for the domains. Table II shows the statistics on the number of object types in CentOS7 that are related to the object class. For example, there are 3,555 object types in the $file$ class. Based on context-aware information, object types with more identical privileges are clustered into one category. Figure 7 shows a clustering scatterplot of randomly selected fifteen categories of object types in $file$ class clustered. The clustering effect is apparent, indicating the presence of identical privileges among different object types in the same category.

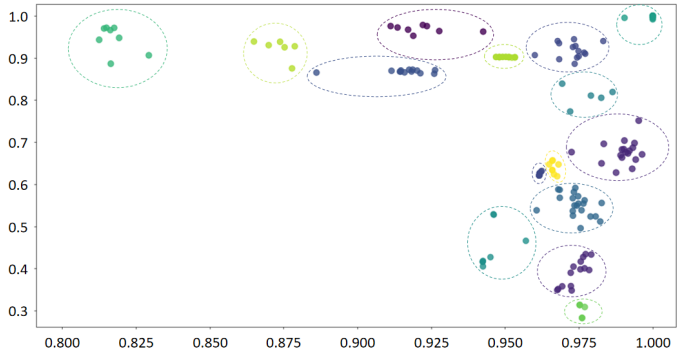

Fig. 7: Cluster Result of Object Types.

2) Contribution of Context-aware Feature based on SHAP Interpretability Analysis: To demonstrate the positive impact of the selected features on the clustering of types, we calculate the contribution value of each feature to the clustering result using the SHAP interpretable program. SHAP is built upon the concept of Shapley values and is a widely used classification model in coalitional games. Based on the samples represented by feature vectors, SHAP can assign a value to each feature input to the classifier which explains the relevance of that

feature to the classification result. Specifically, we use a K-means model to generate labels and employ one-hot to encode the features. To calculate the value of SHAP, we artificially determine the samples with identical privileges. According to the encoded features and the comparison between predetermined labels and labels generated by clustering. The SHAP value effectively quantifies the contribution value of each feature.

In order to express the contribution of each feature intuitively, we randomly selected ten categories to calculate the SHAP values of each feature when clustering existing and newly defined types and represent feature importance as summary plots, which are shown in Figure 8 and Figure 9, respectively. The vertical coordinates represent the selected features. Four features are selected for the clustering of existing types, while only three features are selected for clustering of newly defined types due to the absence of rules in the policy. The horizontal coordinates represent the magnitude of the absolute value of the average SHAP. The colors show the value of the contribution of the features to various categories.

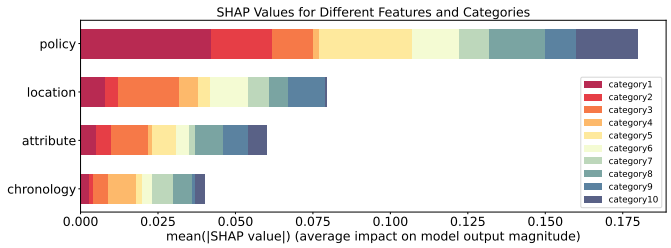

Fig. 8: Mean Absolute Value of the SHAP Values for Each Feature for clustering of existing types.

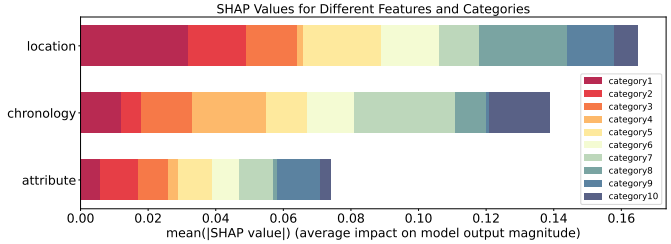

Fig. 9: Mean Absolute Value of the SHAP Values for Each Feature for clustering of newly defined types.

Figure 8 indicates that the security policy feature plays a crucial role in the clustering of existing types, which means that the policy rules are more decisive in determining identical privileges. In contrast, some samples lack certain features, and therefore these features contribute less to clustering. For instance, the majority of types in category 4 lack contained attributes, so the attribute feature has less contribution to this category. In addition, category 9 is prone to limited monitoring data in the audit logs, thus the chronological features are insignificant for this category and the remaining three features contribute to clustering. When clustering newly defined types, the attribute feature has less contribution. This is because newly defined types are often not designated to be associated with an attribute. Although the SHAP values are different for each category, it proves the importance of each feature for CASPR clustering.

C. Rule Recommendation

First, we introduce the evaluation metrics and detail the false samples and their treatment. Afterwards, we control the granularity of clustering by adjusting the thresholds of the model and calculating the accuracy of the recommendation rules at different thresholds in CentOS7. We also compare the accuracy of CASPR with other policy recommendation models to demonstrate the superiority of CASPR in the automated recommendation of rules.

1) Performance of Rule Recommendation: We use the statistics of the positive and negative samples to calculate four metrics in the confusion matrix, including TP, FP, TN, and FN, to measure the performance of CASPR. Table IV shows that CASPR achieves 92.439% accuracy, 93.472% precision, 94.627% recall, and 94.046% F1-score. The high accuracy indicate the effectiveness of automatically recommending rules based on the identical privileges between domains or object types. The high F1-score indicates that CASPR has few false positives and false negatives, which still cannot be ignored. We analyze the reasons and illustrate the treatment.

Analysis of the Reasons for the False Samples: False Positives (FP) is the number of rules leading to over-authorization that should be denied but still recommended. False Negative (FN) is the number of missing rules that should have been allowed but not added to the recommended list. We conduct a manual analysis to identify the underlying reasons causing them. Firstly, it is ideal for the samples that are clustered into one category to have exactly identical privileges. However, it is unlikely for the types in SELinux. Therefore, the privileges differences circumvented during the computation of the learning model in the SELinux access control mechanism are normal. In the process of circumventing these differences, we may inadvertently lead to over-authorization of rule recommendations. Secondly, CASPR is not suitable for analyzing the categories containing fewer samples. In addition, using the object classes with fewer object types as input affects the accuracy of clustering calculations, which results in underreporting. However, the omissions are not significant as these missing rules can be largely addressed with anomaly detection.

Disposal of False Samples: After recommending rules by CASPR, rules that are not recommended and cause under-privileging still require manual assessment. Unpermitted requests that are not covered by rules are recorded in the audit log. The system administrator only needs to determine legitimate privileges according to the audit logs. Filtering the recommended rules from those that would cause excessive privileges is also more feasible for administrators than making rules from scratch. CASPR significantly reduces the time required for manual policy configuration by system administrators and concurrently lowers the probability of configuration errors. In this way, CASPR makes a positive effort to narrow the attack surface.

2) Recommendation Rules for New Types: To illustrate the effectiveness of CASPR for new types in recommending rules, we conduct a separate analysis. Its accuracy is 82.366%, which is slightly lower than the accuracy rate when recommending rules for existing types for the following two reasons. First, the crucial feature of security policy is absent. By analyzing

Threshold Setting	True Malicious (TP)	False Malicious (FP)	True Benign (TN)	False Benign (FN)	True Positive Rate (TPR)	False Positive Rate (FPR)	Accuracy	F1-score
n=2	69.631%	30.369%	41.903%	58.097%	47.895%	35.726%	53.938%	61.153%
n=4	73.985%	26.015%	49.729%	50.271%	63.444%	38.154%	62.854%	65.982%
n=6	78.697%	21.303%	56.694%	43.306%	69.764%	32.298%	69.003%	70.897%
n=8	82.722%	17.278%	64.263%	35.737%	76.353%	27.275%	75.014%	75.732%
n=10	93.472%	6.528%	90.612%	9.388%	94.627%	11.302%	92.439%	94.046%
n=12	89.503%	10.497%	80.034%	19.966%	87.988%	17.649%	89.503%	85.457%

TABLE III: Performance of CASPR recommendation rules under different thresholds.

the feature contribution values in Section V-B2, it is evident that the security policy is the most important feature, while the newly defined types lack the relevant rules, and many of them even lack attributes, which affects the accuracy. Secondly, we establish a stricter threshold for the clustering of new types. The FP of recommending rules for new types is 8.469% and FN is 26.941%. In the scenario of recommending rules, FP occurs when rules that cause excessive privileges are recommended for types, while FN occurs when the privileges they should have are not added to the policy. For system administrators, if there is underreporting, the related behaviors will be denied and administrators need to add them manually; while if there are a lot of misreporting scenarios, security vulnerabilities may occur due to over-authorization. To minimize the false positive rate, we set a stricter threshold, which also affects the accuracy of the rule recommendation.

3) Accuracy of CASPR at Different Thresholds: The thresholds, which control the granularity of clustering are determined by the number of categories to be clustered, denoted as k. The effect of the recommendation of rules based on the clustering results at different thresholds is shown in Table III.

The granularity of the clustering is determined by adjusting the threshold k. Because the domains and object types can be customized in SELinux, the number of samples in the clustering process is not fixed. For the scalability of CASPR, as the number of samples changes, the number of clusters k can be adjusted accordingly to maintain the size of each cluster unchanged. It is assumed that each category consists of n samples on average, which means that $n = all\_samples/k$. We use n as the criterion for clustering granularity. Generally speaking, the more categories are clustered, the fewer samples will be included. But there will always be a situation where the number of clustered samples is large in some categories, and the rest of the categories contain very few samples. Therefore, we choose categories which contain more than 3 samples to analyze to recommend the rules.

We compare the effect of the rule recommendation of CASPR at different values of n. When the value of n is too small, the granularity of clustering is excessively fine, causing types with identical privileges to be clustered into different categories and resulting in errors in rule recommendation. Additionally, the number of samples in each category is too small in fine-grained clustering, if the number is less than 3 we will ignore this category without analyzing it, thus leading to a large number of underreporting. For example, clustering 823 domains when n=2 means that they are required to be clustered into 411 categories, and as a result, there are 298 categories that contain fewer than 3 samples, which is unfavorable for the analysis of privileges.

As the value of n increases, the clustering granularity becomes coarser, and the rule recommendation of CASPR becomes better. Until n=10, the accuracy of the recommendation reaches 92.439%. However, when n continues to increase, the accuracy of CASPR decreases. Since the clustering granularity is too coarse, resulting in clustering samples whose privileges are not so identical into one category. For example, with n=12, it means that we cluster all domains into 69 categories, where the largest category contains 42 samples, thus leading to generating over-granting rules. Thus, the experimental data suggests that clustering is best at n=10.

4) Comparison CASPR with Policy Recommendation Models: In addition, we compare CASPR with a baseline model based on EASEAndroid and SEPAL. Although they are designed for SEAndroid policy recommendations, the policy analysis can also be generalized. SEAndroid inherits the core architecture of SELinux and is customized for mobile devices. It leads to a high degree of similarity in the expression of their policies. Figure 10 compares SEAndroid and SELinux policies, which show that they contain the same fields, including types and permissions. The similarity ensures consistency in data input, enabling the methods to be applied in both systems. We reproduce the baseline models in SELinux and calculate the performance of the recommendation of SELinux policy rules in CentOS7.

```
# SELinux policy rules for process execution and termination.
allow init_t domain : process { sigchld sigkill sigstop signull signal } ;

# SEAndroid policy rules for process execution and termination.
allow init domain:process { sigkill signal };
-----------------------------------------------------------------------------------------------------------
#SELinux policy rules for process access to directories and symbolically linked files.
allow init_t tmp_t : dir { open create read getattr setattr search } ;
allow init_t tmp_t : lnk_file { read getattr } ;

#SEAndroid policy rules for process access to directories and symbolically linked files.
allow init vold_data_file:dir { open create read getattr setattr search };
allow init vold_data_file:lnk_file { getattr };
```

Fig. 10: Comparision of SELinux Policy and SEAndroid Policy.

EASEAndroid uses classifiers to refine the policy collaboratively. In particular, both the Nearest-neighbors-based classifier and the Pattern-to-rule distance measure judge whether the rule should be allowed or not by measuring the distance between the unknown and known rules. Specifically, two rules can be neighbors if they are similar. An unknown rule can be classified if the number of neighbors exceeds m and the majority of neighbors account for more than σ. And if the

distance between two rules is less than $Dist$ as measured by the pattern-to-rule distance, they are judged to be similar.

Compared to EASEAndroid, SEPAL not only considers rule similarity but also integrates additional information to represent the policy semantics better. The implementation of SEPAL consists of the collection of atomic rules, feature extraction, and classification using a wide & deep learning model. SEPAL extracts attribute features, user ID features, and NLP-based features. SEPAL integrates a wide linear model and a deep neural network to capture more complex correlations between rules and thus determine whether one rule is allowed or not by predicting the preference of subjects for objects based on their features.

Metrics	Threshold	Evaluation Indicators				
		Accuracy	Precision	Recall	F1-score	FPR
CASPR	n=10, $0.2<\theta<0.3$	92.439%	93.472%	94.627%	94.046	11.302%
EASEAndroid	m=10, σ=55%, $Dist$=1	78.193%	84.166%	80.067%	82.348%	25.935%
SEPAL	-	88.436%	84.794%	89.286%	86.983%	12.078%

TABLE IV: Comparison with other policy recommendation models.

CASPR considers the policy, file locations, audit logs, and attribute information of the domains and object types. The addition of context-aware information enhances the ability of CASPR to recommend rules. We reproduce EASEAndroid and SEPAL and deploy them in CentOS7 and train the model using policy rules in SELinux. The comparison results are shown in Table IV. CASPR achieves 92.439% accuracy, while EASEAndroid obtains 78.193% accuracy and SEPAL obtains 88.436% accuracy. The results indicate that CASPR outperforms the baseline models.

In addition to significantly higher accuracy, another major advantage of CASPR is a model for generating recommendation rules using context-aware information. EASEAndroid and SEPAL can only recommend policy rules for existing access requests and determine whether a certain access request is allowed or not. While CASPR generates rules based on privilege calculation, it is not limited to simply determining existing rules. Therefore, CASPR is more responsive to the requirements of the system, especially for assigning privileges to new types.

D. Anomaly Detection

We add the recommended rules to the policy and analyze if any anomaly arises subsequent to the policy modification. Then, we patch the policy after detecting the anomalies. Out of the recommended 97,583 rules in CentOS7, we identify a total of 168 anomalies. Among these anomalies, the number of constraint conflicts is 46, the number of policy inconsistencies is 54, and the number of permission incompleteness is 58. Some anomalies are caused by incomplete rules after recommendation, while some anomalies are caused by false positives in CASPR. Eliminating these anomalies requires recalculating all the rules involved in the anomalies, including those generated to fix the anomalies, and inputting them to CASPR. If the majority of the rules are not recommended, they are deleted. Conversely, if they are recommended, we classify them in the recommended list. It is intended to address inconsistencies and incompleteness and to ensure the effective enforcement of other rules. Additionally, we artificially generate some sets of rules and use them to assess the effectiveness of anomaly detection. The algorithms for anomaly detection and policy refinement rely solely on basic regular matching and apply to different versions of SELinux policy anomaly detection as they have the same grammar and semantics. The algorithm recognizes three kinds of anomalies, and we verify that the method is accurate and practicable.

E. Performance Evaluation

1) Universality between Different Versions: To demonstrate the adaptability of CASPR to various versions of SELinux policy rule recommendations, we deploy it in different operating systems. We extract the standard SELinux policies in each operating system and use 10% of those rules, as a test set, in addition to some new types customized for testing the effect of CASPR's rule recommendations.

Table V shows the results of CASPR in recommendations for different versions of the SELinux policy. CASPR achieves 92.582% accuracy, 92.397% precision, 93.982% recall, and 93.761% F1-score on average. The performance of policy recommendations is affected by the number of rules in different systems, and the adequacy of the selected context-aware information. In conclusion, the experimental results indicate that whether applied to CentOS or Ubuntu systems, CASPR consistently performs well, indicating its robustness in recommending SELinux policies. It demonstrates that the privilege calculation based on context-aware information is universal to adapt to diverse system environments.

	Accuracy	Precision	Recall	F1-score	FPR
CentOS6	91.163%	92.859%	94.987%	93.905%	15.214%
CentOS7	92.439%	93.472%	94.627%	94.046%	11.302%
CentOS8	92.687%	93.085%	93.508%	93.323%	14.365%
Ubuntu20	90.584%	90.925%	92.945%	93.457%	12.283%
Ubuntu22	91.422%	92.649%	94.384%	93.986%	14.744%
Ubuntu24	91.196%	91.394%	93.481%	93.847%	14.497%
Avg.	91.582%	92.397%	93.982%	93.761%	13.734%

TABLE V: Performance of CASPR for different versions of SELinux policy.

2) Base Configuration and Computational Efficiency: For each version of the SELinux policy, CASPR needs to be retrained. The initialization, including dataset loading, feature extraction, and feature matrix construction, takes around one hour on average on a server equipped with a 6-core CPU and 30GB of RAM. This is primarily due to the computational complexity and the large volume of data involved. After initialization, the clustering time for different domains and types of entities varies significantly. These differences arise from factors such as the sample size, convergence speed, and the selected hyperparameters. Specially, clustering subject domains takes approximately 1.5 minutes, while clustering

object types requires around 8 minutes. Policy recommendation, based on straightforward pattern matching after the clustering phase, takes an additional approximately 3 minutes to complete.

Despite variations in data sizes and rule complexities across SELinux versions, CASPR consistently maintains a reasonable runtime. The overall computational efficiency of CASPR demonstrates its scalability and adaptability across different systems and policy versions. Its ability to handle large datasets and deliver timely policy recommendations makes it a versatile solution for dynamic access control scenarios.

F. Case Analysis

We demonstrate CASPR on three cases to address the following questions, whether rules are permitted and what privileges are granted to a new type, and how to refine policy after detecting anomalies.

1) Rule Recommendation for Existing Types: With the development of technology and the continuous updating of system resources, SELinux encounters various access requests. To ensure the system can make prompt and accurate decisions on these access requests, it is crucial to regularly update the SELinux policy. However, if the SELinux policy lacks rules, the system will deny access requests that should be allowed, causing system irregularities or software functionality that cannot be performed properly. Consequently, the system administrator must manually authorize these access requests, which is error-prone and time-consuming.

After ascertaining the categories of the domain and the object type within the rule, the rules are determined by CASPR. Subsequently, we judge them based on the privileged information of other samples within the category. If the majority of the domains in the category to which the subject belongs have specific permission on this object, and the majority of the types in the category to which the object belongs also have the same permission to be implemented by this subject, CASPR recommends this rule. For example, for the rule $allow\ systemd\_machined\_t\ system\_dbusd\_var\_run\_t : dir\ \{\ read\ \}$, the domain is $systemd\_machined\_t$, which is used to control the systemd-machined process. The object is a directory, and its type is $system\_dbusd\_var\_run\_t$, which is used to identify system-level D-Bus services running in the /var/run/dbus/system_bus_socket directory. For the subjects, the domains $systemd\_located\_t$, $systemd\ \_hostnamed\_t$, $systemd\_timedated\_t$, and so on, of its category have permissions of reading to the directory of type $system\_dbusd\_var\_run\ \_t$. If the object class is *dir*, the types $system\_dbusd\_t$, $dbusd\ \_etc\_t$, $system\_dbusd\_var\_lib\_t$, and so on, are authorized to be implemented with reading permission by a subject of domain $systemd\ \_machined\_t$. Therefore, we artificial that this rule is allowed and add it to the policy.

2) Rule Recommendation for New Types: Typically, when new software is installed on a system, it inherits the security label of its parent directory. However, SELinux allows system administrators to customize new security labels to achieve fine-grained access control. In addition to customizing a new security label, the administrator also grants the label with corresponding privileges. However, given the extensive array of rules in SELinux policy, it is challenging to avoid being negligent. At the initial state, there are no rules related to new types and we recommend policy rules through context-aware information except for policy rules. If not granted authorization promptly, behaviors related to subjects with a new domain and objects with a new object type that are not assigned rules will be intercepted.

For example, we install new software and assign it with a new security label. When configuring the SELinux policy to grant some privileges to this new software, to keep the minimal authorization principle in SELinux and not to cause over-authorization, we audit the privileges of the software to know what privileges it needs to work properly. Then it is essential to consider mainly its file and directory security, process security, port security, and other resources to configure the policy. This process is cumbersome and error-prone. We use security context-aware information to infer what privileges this software has. For instance, after installing the Nginx server, the processes are assigned a new domain, such as *nginx_t*. The request of the client for files includes receiving network connections, reading files, and sending file contents. These behaviors are recorded in audit logs with short time intervals. If clients are allowed to request files, these behaviors should be authorized. Therefore, in addition to being authorized to read files of *https_sys_content_t* type, processes of domain *nginx_t* also have send_msg permission on files of type *https_client_packet_t*. We feed both the known privileges and other context-aware information into CASPR. We recommend rules that are required for the software to function properly and add them to the policy.

3) Anomaly Detection and Policy Refinement: By CASPR calculation, it is determined that a certain domain of subjects should have privileges but the policy lacks the relevant rules to authorize them, it is recommended to add new rules to the policy. However, in many cases, just adding these rules does not allow the privileges to be executed properly. For example, when assigning the permission of writing to the *nginx_t* domain, we not only recommend the rule *allow nginx_t httpd_sys_content_t : file write;*, but also grant it the permissions, including *getattr, append, lock, ioctl and open* due to the restriction *write_file_perms:open getattr write append lock ioctl*. Since the anomaly detection in CASPR is a rule-matching process, CASPR effectively identifies anomalies and refines the policy as long as the SELinux restriction file is complete.

VI. Related Work

Due to the complex semantics of security policies, it is error-prone and time-consuming. There is a huge number of prior studies related to security policies, which focus on policy analysis, anomaly detection, and automatic generation of security policy rules to refine and optimize the policies.

A. SELinux Policy Analysis

Multiple studies analyze security policy. Eaman et al. [13] summarize 18 policy analysis tools and compare them from aspects such as safety analysis, completeness analysis, integrity analysis, and information flow analysis. Various approaches [7], [10], [19], [22], [39] have been proposed to

analyze SELinux and Security Enhancements for Android (SE-Android) [44] policy. Some studies analyze the completeness, and consistency of policy through semantic analysis [4], [22], adjacency matrix [51], [52], information flow analysis [34], and colored petri net [56] methods. Additionally, other work focuses on the visualization of policy. These studies, such as SPtrack [10], SEGrapher [27], BIGMAC [18], PoliGraph [11] perform data visualization and simplifies policy analysis by knowledge graphs, clustering methods, and natural language processing.

B. Policy Anomaly Detection

Policy anomaly detection mainly detects inconsistency [8], [42], incompleteness [49] and conflict anomalies. Various methods have been proposed for detecting policy inconsistencies, such as data mining [6], [12], [50], [58], testing [14], and verification [16], [23]. There are some methods to effectively detect misconfiguration when changing access control policies, such as mining association rules from access history [6], monitoring access control metadata updates [12], monitoring access control behaviours based on a novel incident-based decision tree approach [47], [50], analyzing the interaction between configuration entities [58].

In order to test whether the policy is correct, it is necessary to determine whether the output is as expected. Unfortunately, manual test generation is tedious and insufficient to exercise various policy behaviors. NLP technology [55] and fault detection model [14], [28] were proposed to achieve high structural coverage and fault-detection capability. Furthermore, to verify the policy, Margrave [16], which converts role-based access control policies into the form of decision diagrams for analyzing query and semantic difference information. Additionally, a new abstraction-refinement technique [23] was proposed in ARBAC security policies for automatic finding errors. Automatic detecting of incompleteness and inconsistent anomalies of policies in a large number of complex policies is a difficult and challenging problem. There are some methods, such as data classification tools [40], partial order relationships [57], propagation of conflicts [56], trusted computing libraries [22], conflict auditing methods [17], which efficiently detect and resolve conflicting rules of security policies.

C. Automated Policy Generation

Typically, policy automation recommendation mines information from logs or policy rules. The classical methods, such as large-scale semi-supervised learning methods [45], SPDL [32], DOMinator [46], approximate mining and probabilistic algorithms [31], quantitative scoring [36], rule mining algorithm [29]. They are used to mine information from rules to automatically analyze and refine policies [21]. The following is the study of log information extraction and automatic generation policy. They are based on time-changing decision tree methods [50], static analysis [43], supervised learning-based approach [24], a graph-based policy management mechanism [25] to extract policies from logs to address legitimate privileges while avoiding over-granting [9].

The previous researches on security policies lack the comprehensive analysis of security context-aware information. CASPR, as a policy recommendation and anomaly detection tool, greatly improves accuracy and adapts to complex environments.

VII. Discussion

This paper proposes a security policy rule recommendation based on context-aware information and anomaly detection methods, which are different from traditional methods. CASPR utilizes multi-factors for comprehensive recommendations, which can recommend effective and feasible security policy rules. The importance of CASPR, feasibility of deployment, and limitations are discussed below.

A. Significance.

Most of the previous researches focus on extracting information from logs, policy rules, and anomalies, and then mining the information to optimize security policy rules. CASPR uses context-aware information for privilege calculation and policy recommendation. Experiments in Section V-C demonstrate that this approach receives high accuracy. Reproducing the policy recommendation models of EASEAndroid and SEPAL on SELinux and comparing them with CASPR illustrates the superiority. The performance of CASPR demonstrates the importance of context-aware information for policy recommendation, which is more efficient and flexible.

B. Deployment.

CASPR as a policy rule recommendation tool can be easily deployed in the operating system. Firstly, it collects the context-aware information in the system and preprocesses data for normalization. Then CASPR is used to automatically generate a list of recommendation rules and continuously optimize and refine the policies. The experimental results show that CASPR applies to different versions of SELinux policies, as shown in Section V-E. Although different versions of the policy require retraining of the model, it is only trained once during deployment. The experiments have shown that the recommendations are efficient in accommodating the need for privilege optimization in dynamic access control systems.

C. Limitation.

CASPR significantly improves the accuracy of policy rule recommendations. Nonetheless, the following limitations should be borne in mind. The primary limitation is the difference of policies for different versions, which causes the results of policy analysis to be only suitable to the current version of the policy. The policies for other versions need to be re-learned and re-analyzed, by CASPR to recommend policy rules and detect anomalies based on the analysis results. The second limitation is that the context-aware information of some samples is incomplete. For example, there is no specific location information for port type. The context-aware information lacks location information. Therefore, the location information is only effective for types where location information is available. Additionally, the algorithm of K-means model is sensitive to the initial clustering centers and needs to pre-determine the k-value. Therefore, multiple attempts and adjustments need to be made based on the data to select the optimal clustering result. Notwithstanding these limitations, the study suggests that context-aware information contributes

to policy recommendations, improving accuracy by calculating these information.

VIII. Conclusion

This paper proposes a novel approach called CASPR to address the challenges of recommending policy automatically. CASPR takes a comprehensive view of context-aware information, clusters domains, and object types by privilege calculation automatically recommends security policy rules, and detects three kinds of anomalies, such as constraint conflicts, policy inconsistencies, and permission incompleteness. In this paper, we implement CASPR and evaluate its performance. The experimental results show that CASPR effectively recommends appropriate security policy rules, which satisfy the functional requirements without leading to over-granting. Compared with other security policy rule recommendation methods, CASPR significantly improves the accuracy rate and reduces human effort. This study has significant research potential and value for rule recommendation.

Acknowledgement

We sincerely thank all anonymous reviewers for their valuable comments and these insightful comments have greatly contributed to the improvement and refinement of this paper. This research is supported by the Project on Dynamic Regulation of Complex Environmental Resources, Institute of Information Engineering, Chinese Academy of Sciences, Grant No. E3Z0101205.

References

[1] CVE-2019-13272. 2019. [Online]. Available: https://nvd.nist.gov/vuln/detail/ CVE-2019-13272

[2] Security-Enhanced Linux in Android. 2024. [Online]. Available: https://source.android.com/devices/tech/security/selinux

[3] SELinux Project. 2020. [Online]. Available: http://selinuxproject.org

[4] M. Archer, E. I. Leonard, and M. Pradella, "Modeling security-enhanced linux policy specifications for analysis," in *3rd DARPA Information Survivability Conference and Exposition (DISCEX-III 2003), 22-24 April 2003, Washington, DC, USA*. IEEE Computer Society, 2003, pp. 164–169. [Online]. Available: https://doi.org/10.1109/DISCEX.2003.1194959

[5] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat, "Practical domain and type enforcement for UNIX," in *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*. IEEE Computer Society, 1995, pp. 66–77. [Online]. Available: https://doi.org/10.1109/SECPRI.1995.398923

[6] L. Bauer, S. Garriss, and M. K. Reiter, "Detecting and resolving policy misconfigurations in access-control systems," *ACM Trans. Inf. Syst. Secur.*, vol. 14, no. 1, pp. 2:1–2:28, 2011. [Online]. Available: https://doi.org/10.1145/1952982.1952984

[7] T. Bui and S. D. Stoller, "A decision tree learning approach for mining relationship-based access control policies," in *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies, SACMAT 2020, Barcelona, Spain, June 10-12, 2020*, J. Lobo, S. D. Stoller, and P. Liu, Eds. ACM, 2020, pp. 167–178. [Online]. Available: https://doi.org/10.1145/3381991.3395619

[8] S. Calzavara, T. Urban, D. Tatang, M. Steffens, and B. Stock, "Reining in the web's inconsistencies with site policy," in *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/reining-in-the-webs-inconsistencies-with-site-policy/

[9] S. Chari, I. M. Molloy, Y. Park, and W. Teiken, "Ensuring continuous compliance through reconciling policy with usage," in *18th ACM Symposium on Access Control Models and Technologies, SACMAT '13, Amsterdam, The Netherlands, June 12-14, 2013*, M. Conti, J. Vaidya, and A. Schaad, Eds. ACM, 2013, pp. 49–60. [Online]. Available: https://doi.org/10.1145/2462410.2462417

[10] P. Clemente, B. Kaba, J. Rouzaud-Cornabas, M. Alexandre, and G. Aujay, "Sptrack: Visual analysis of information flows within selinux policies and attack logs," in *Active Media Technology: 8th International Conference, AMT 2012, Macau, China, December 4-7, 2012. Proceedings 8*. Springer, 2012, pp. 596–605.

[11] H. Cui, R. Trimananda, A. Markopoulou, and S. Jordan, "Poligraph: Automated privacy policy analysis using knowledge graphs," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, 2023, pp. 1037–1054. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/cui

[12] T. Das, R. Bhagwan, and P. Naldurg, "Baaz: A system for detecting access control misconfigurations," in *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*. USENIX Association, 2010, pp. 161–176. [Online]. Available: http://www.usenix.org/events/sec10/tech/full_papers/Das.pdf

[13] A. Eaman, B. Sistany, and A. P. Felty, "Review of existing analysis tools for selinux security policies: Challenges and a proposed solution," in *E-Technologies: Embracing the Internet of Things - 7th International Conference, MCETECH 2017, Ottawa, ON, Canada, May 17-19, 2017, Proceedings*, ser. Lecture Notes in Business Information Processing, E. Aïmeur, U. Ruhi, and M. Weiss, Eds., vol. 289, 2017, pp. 116–135. [Online]. Available: https://doi.org/10.1007/978-3-319-59041-7_7

[14] Evan Martin and Tao Xie, "A fault model and mutation testing of access control policies," in *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, Eds. ACM, 2007, pp. 667–676. [Online]. Available: https://doi.org/10.1145/1242572.1242663

[15] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, 2001. [Online]. Available: https://doi.org/10.1145/501978.501980

[16] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, G. Roman, W. G. Griswold, and B. Nuseibeh, Eds. ACM, 2005, pp. 196–205. [Online]. Available: https://doi.org/10.1145/1062455.1062502

[17] M. A. E. Hadj, A. Khoumsi, Y. Benkaouz, and M. Erradi, "A log-based method to detect and resolve efficiently conflicts in access control policies," in *Proceedings of the 12th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2020), 15-18 December 2020*, ser. Advances in Intelligent Systems and Computing, A. Abraham, Y. Ohsawa, N. Gandhi, M. A. Jabbar, A. Haqiq, S. F. McLoone, and B. Issac, Eds., vol. 1383. Springer, 2020, pp. 836–846. [Online]. Available: https://doi.org/10.1007/978-3-030-73689-7_79

[18] G. Hernandez, D. J. Tian, A. S. Yadav, B. J. Williams, and K. R. B. Butler, "Bigmac: Fine-grained policy analysis of android firmware," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. USENIX Association, 2020, pp. 271–287. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/hernandez

[19] B. Hicks, S. J. Rueda, L. S. Clair, T. Jaeger, and P. D. McDaniel, "A logical specification and analysis for selinux MLS policy," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 3, pp. 26:1–26:31, 2010. [Online]. Available: https://doi.org/10.1145/1805974.1805982

[20] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST special publication*, vol. 800, no. 162, pp. 1–54, 2013.

[21] P. Iyer and A. Masoumzadeh, "Mining positive and negative attribute-based access control policy rules," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018, Indianapolis, IN, USA, June 13-15, 2018*, E. Bertino, D. Lin,

and J. Lobo, Eds. ACM, 2018, pp. 161–172. [Online]. Available: https://doi.org/10.1145/3205977.3205988

[22] T. Jaeger, R. Sailer, and X. Zhang, "Analyzing integrity protection in the selinux example policy," in *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003. [Online]. Available: https://www.usenix.org/conference/12th-usenix-security-symposium/analyzing-integrity-protection-selinux-example-policy

[23] K. Jayaraman, V. Ganesh, M. V. Tripunitara, M. C. Rinard, and S. J. Chapin, "Automatic error finding in access-control policies," in *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011, pp. 163–174. [Online]. Available: https://doi.org/10.1145/2046707.2046727

[24] L. Karimi, M. Aldairi, J. Joshi, and M. Abdelhakim, "An automatic attribute-based access control policy extraction from access logs," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 4, pp. 2304–2317, 2022. [Online]. Available: https://doi.org/10.1109/TDSC.2021.3054331

[25] X. Li, Y. Chen, Z. Lin, X. Wang, and J. H. Chen, "Automatic policy generation for inter-service access control of microservices," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. D. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 3971–3988. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/li-xing

[26] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[27] S. Marouf and M. Shehab, "Segrapher: Visualization-based selinux policy analysis," in *4th Symposium on Configuration Analytics and Automation, SafeConfig 2011, Arlington, VA, USA, October 31 - November 1, 2011*, J. Banghart, E. Al-Shaer, T. Sager, and H. V. Ramasamy, Eds. IEEE, 2011. [Online]. Available: https://doi.org/10.1109/SafeConfig.2011.6111675

[28] E. Martin and T. Xie, "Automated test generation for access control policies via change-impact analysis," in *Third International Workshop on Software Engineering for Secure Systems, SESS 2007, Minneapolis, MN, USA, May 20-26, 2007*. IEEE Computer Society, 2007, p. 5. [Online]. Available: https://doi.org/10.1109/SESS.2007.5

[29] Matthew W. Sanders and Chuan Yue, "Mining least privilege attribute based access control policies," in *Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC 2019, San Juan, PR, USA, December 09-13, 2019*, D. Balenson, Ed. ACM, 2019, pp. 404–416. [Online]. Available: https://doi.org/10.1145/3359789.3359805

[30] F. Mayer, D. Caplan, and K. MacMillan, *SELinux by example: using security enhanced Linux*. Pearson Education, 2006.

[31] I. M. Molloy, Y. Park, and S. Chari, "Generative models for access control policies: applications to role mining over logs with attribution," in *17th ACM Symposium on Access Control Models and Technologies, SACMAT '12, Newark, NJ, USA - June 20 - 22, 2012*, V. Atluri, J. Vaidya, A. Kern, and M. Kantarcioglu, Eds. ACM, 2012, pp. 45–56. [Online]. Available: https://doi.org/10.1145/2295136.2295145

[32] Y. Nakamura, Y. Sameshima, and T. Tabata, "Seedit: Selinux security policy configuration system with higher level language," in *Proceedings of the 23rd Large Installation System Administration Conference, November 1-6, 2009, Baltimore, MD, USA*, A. Moskowitz, Ed. USENIX Association, 2009, pp. 107–117. [Online]. Available: http://www.usenix.org/events/lisa09/tech/full_papers/nakamura.pdf

[33] L. Qiu, Y. Zhang, F. Wang, M. Kyung, and H. R. Mahajan, "Trusted computer system evaluation criteria," *National Computer Security Center*, vol. 12, p. 71, 1985.

[34] B. S. Radhika, N. V. N. Kumar, and R. K. Shyamasundar, "Flowconseal: Automatic flow consistency analysis of seandroid and selinux policies," in *Data and Applications Security and Privacy XXXII - 32nd Annual IFIP WG 11.3 Conference, DBSec 2018, Bergamo, Italy, July 16-18, 2018, Proceedings*, ser. Lecture Notes in Computer Science, F. Kerschbaum and S. Paraboschi, Eds., vol. 10980. Springer, 2018, pp. 219–231. [Online]. Available: https://doi.org/10.1007/978-3-319-95729-6_14

[35] P. V. Rajkumar and R. S. Sandhu, "POSTER: security enhanced administrative role based access control models," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1802–1804. [Online]. Available: https://doi.org/10.1145/2976749.2989068

[36] M. W. Sanders and C. Yue, "Minimizing privilege assignment errors in cloud services," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY 2018, Tempe, AZ, USA, March 19-21, 2018*, Z. Zhao, G. Ahn, R. Krishnan, and G. Ghinita, Eds. ACM, 2018, pp. 2–12. [Online]. Available: https://doi.org/10.1145/3176258.3176307

[37] R. S. Sandhu, "The typed access matrix model," in *1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 4-6, 1992*. IEEE Computer Society, 1992, pp. 122–136. [Online]. Available: https://doi.org/10.1109/RISP.1992.213266

[38] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[39] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan, "Policy analysis for administrative role based access control," in *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*. IEEE Computer Society, 2006, pp. 124–138. [Online]. Available: https://doi.org/10.1109/CSFW.2006.22

[40] R. A. Shaikh, K. Adi, and L. Logrippo, "A data classification method for inconsistency and incompleteness detection in access control policy sets," *Int. J. Inf. Sec.*, vol. 16, no. 1, pp. 91–113, 2017. [Online]. Available: https://doi.org/10.1007/s10207-016-0317-1

[41] Z. Shan, X. Wang, and T.-c. Chiueh, "Enforcing mandatory access control in commodity os to disable malware," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 4, pp. 541–555, 2012.

[42] Y. Shao, Q. A. Chen, Z. M. Mao, J. Ott, and Z. Qian, "Kratos: Discovering inconsistent security policy enforcement in the android framework," in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/kratos-discovering-inconsistent-security-policy-enforcement-android-framework_0.pdf

[43] B. Shen, T. Shan, and Y. Zhou, "Improving logging to reduce permission over-granting mistakes," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, 2023, pp. 409–426. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/shen-bingyu-logging

[44] S. Smalley and R. Craig, "Security enhanced (SE) android: Bringing flexible MAC to android," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013. [Online]. Available: https://www.ndss-symposium.org/ndss2013/security-enhanced-se-android-bringing-flexible-mac-android

[45] R. Wang, W. Enck, D. S. Reeves, X. Zhang, P. Ning, D. Xu, W. Zhou, and A. M. Azab, "Easeandroid: Automatic policy analysis and refinement for security enhanced android via large-scale semi-supervised learning," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 351–366. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wang-ruowen

[46] Z. Wang, W. Meng, and M. R. Lyu, "Fine-grained data-centric content protection policy for web applications," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 2845–2859. [Online]. Available: https://doi.org/10.1145/3576915.3623217

[47] S. Wi, T. T. Nguyen, J. Kim, B. Stock, and S. Son, "Diffcsp:

Finding browser bugs in content security policy enforcement through differential testing," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/diffcsp-finding-browser-bugs-in-content-security-policy-enforcement-through-differential-testing/

[48] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, D. Boneh, Ed. USENIX, 2002, pp. 17–31. [Online]. Available: http://www.usenix.org/publications/library/proceedings/sec02/wright.html

[49] A. Xiang, W. Pei, and C. Yue, "Policychecker: Analyzing the GDPR completeness of mobile apps' privacy policies," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 3373–3387. [Online]. Available: https://doi.org/10.1145/3576915.3623067

[50] C. Xiang, Y. Wu, B. Shen, M. Shen, H. Huang, T. Xu, Y. Zhou, C. Moore, X. Jin, and T. Sheng, "Towards continuous access control validation and forensics," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 113–129.

[51] W. Xu, M. Shehab, and G. Ahn, "Visualization-based policy analysis for selinux: framework and user study," *Int. J. Inf. Sec.*, vol. 12, no. 3, pp. 155–171, 2013. [Online]. Available: https://doi.org/10.1007/s10207-012-0180-7

[52] W. Xu, X. Zhang, and G. Ahn, "Towards system integrity protection with graph-based policy analysis," in *Data and Applications Security XXIII, 23rd Annual IFIP WG 11.3 Working Conference, Montreal, Canada, July 12-15, 2009. Proceedings*, ser. Lecture Notes in Computer Science, E. Gudes and J. Vaidya, Eds., vol. 5645. Springer, 2009, pp. 65–80. [Online]. Available: https://doi.org/10.1007/978-3-642-03007-9_5

[53] D. Yan, J. Huang, Y. Tian, Y. Zhao, and F. Yang, "Policy conflict detection in composite web services with rbac," in *2014 IEEE International Conference on Web Services*. IEEE, 2014, pp. 534–541.

[54] B. Yang, "Enforcement of separation of duty constraints in attribute-based access control," *Comput. Secur.*, vol. 131, p. 103294, 2023. [Online]. Available: https://doi.org/10.1016/j.cose.2023.103294

[55] D. Yu, G. Yang, G. Meng, X. Gong, X. Zhang, X. Xiang, X. Wang, Y. Jiang, K. Chen, W. Zou *et al.*, "Sepal: Towards a large-scale analysis of seandroid policy customization," in *Proceedings of the Web Conference 2021*, 2021, pp. 2733–2744.

[56] G. Zhai, T. Guo, and J. Huang, "Sciatool: A tool for analyzing selinux policies based on access control spaces, information flows and cpns," in *Trusted Systems - 6th International Conference, INTRUST 2014, Beijing, China, December 16-17, 2014, Revised Selected Papers*, ser. Lecture Notes in Computer Science, M. Yung, L. Zhu, and Y. Yang, Eds., vol. 9473. Springer, 2014, pp. 294–309. [Online]. Available: https://doi.org/10.1007/978-3-319-27998-5_19

[57] H. Zhang, P. Ma, and M. Wang, "Detecting inconsistency and incompleteness in access control policies," in *Cloud Computing and Security - 4th International Conference, ICCCS 2018, Haikou, China, June 8-10, 2018, Revised Selected Papers, Part II*, ser. Lecture Notes in Computer Science, X. Sun, Z. Pan, and E. Bertino, Eds., vol. 11064. Springer, 2018, pp. 731–739. [Online]. Available: https://doi.org/10.1007/978-3-030-00009-7_65

[58] J. Zhang, L. Renganarayana, X. Zhang, N. Ge, V. Bala, T. Xu, and Y. Zhou, "Encore: exploiting system environment and correlation information for misconfiguration detection," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, Salt Lake City, UT, USA, March 1-5, 2014*, R. Balasubramonian, A. Davis, and S. V. Adve, Eds. ACM, 2014, pp. 687–700. [Online]. Available: https://doi.org/10.1145/2541940.2541983