Multiview: Finding Blind Spots in Access-Deny Issues Diagnosis

Bingyu Shen[*] Tianyi Shan[*] Yuanyuan Zhou
University of California, San Diego

Abstract

Access-deny issues are hard to fix because it implies both availability and security requirements. On one hand, system administrators (sysadmins) need to make a change quickly to enable legitimate access. On the other hand, sysadmins need to make sure the change does not allow excessive access. Fulfilling the second requirement on security is especially challenging because it highly requires the sysadmins' knowledge of the system environments and security context. Blind spots in knowledge and system settings may hinder sysadmins from finding solutions that align with the security context. Insecure fixes can over-grant permissions, which may only get noticed after the security vulnerability gets exploited.

This paper aims to help sysadmins reduce blind spots in diagnosis by providing multiple directions to resolve access-deny issues. We propose a system, called Multiview, that automatically mutates the configurations to explore possible directions to fix the access-deny issue and lets the configuration changes in each direction grant as few permissions as possible. Multiview provides a detailed diagnosis report, including access-control configurations that are related to the denial, possible configuration changes in different directions to allow the request, as well as the impact on the access-control state of the entire system.

We conducted a user study to evaluate Multiview with 20 participants on five real-world access-deny issues. Multiview can reduce the percentage of insecure fixes from 44.0% to 2.0% and reduce the diagnosis time by 62.0% on average. We also evaluated Multiview on 112 real-world failure cases from eight different systems and server applications, and it can successfully diagnose 89 of them. Multiview accurately identifies the failure-causing configurations and provides possible directions to each access-deny issue within one minute.

1 Introduction

1.1 Motivation

Access control is critical to protect the systems resources and users' data from unauthorized access [38, 70]. As the business grows and evolves, the organization's access-control configurations changes constantly, such as adding new members or performing system upgrades. Sysadmins need to adjust the access-control configurations to accommodate the dynamic needs of the systems, as well as fix issues in case of problems. *Access-deny issues*, where legitimate users' access to data is falsely denied, have become common issues faced by sysadmins [79].

Inaccurately fixing access-deny issues often results in permission over-grant mistakes. According to a recent study [79] on how sysadmins resolve access-deny issues in real-world situations, 38.1% of cases introduced security misconfigurations. Many of the misconfigurations are caused by trial and error when sysadmins misunderstand the root causes. Many suggestions on the forum are often ill-advised as they lead sysadmins to introduce permission over-grant mistakes. In an Apache server access-deny issue [59], the answer that received the most upvotes is to completely disable the access control of Apache. Sysadmins can easily over grant permissions through trial and error or randomly adopt online advice when they encounter an access-deny issue.

Permission over-granting mistake is one of the major triggers of security incidents [28, 39, 45, 56, 68, 69, 82]. In general, users only notify sysadmins when their requests are denied, and rarely notice or complain about excessive permissions. Over-granted permissions were only discovered after a security incident when the system has been exploited by malicious attackers [52]. Indeed, many enterprises have suffered from security incidents caused by permission misconfigurations, including data breaches, ransomware and system compromises [30, 35, 58, 70]. From 2018 to 2019, the number of records lost by misconfiguration rose by 80% as did the cost to the associated companies [30]. In 2021, the non-profit security community OWASP also chose broken access control as one of the top 10 vulnerabilities in web applications [47].

Real-world access-deny issues are difficult to get right because it implies both *availability* and *security* requirements. There are many ways to grant the required permissions. A good solution, however, should be correct and safe as it not only solves the access-deny issue but also does not grant excessive permissions. However, when sysadmins manually resolve access-deny issues, they may have blind spots because of misinterpreting the security context, neglecting the security consequence, or being under time pressure. Sysadmins can

[*] Co-first authors

Denied Request		
Subj.: apache	**Action**: read	**Object**:/home/alice/wsgi/index.py
Root-cause configuration		
Directory: /home/alice/wsgi/ Permission: `drwxr----- alice alice`		
Possible Directions		
1. Grant Apache's current role `other` with execute permission 2. Change the directory (/home/alice/wsgi/)'s group to be apache and grant execute permission. 3. Change the file and directory's owner to be apache.		

Figure 1: **A simplified access-deny issue with root cause in the file permission.**

Denied Request		
Subj.: Alice	**Action**: GET	**Object**:/Application/2021/
Root-cause configuration		
<Directory /var/www/html/Application/> Require group staff </Directory>		
Alice's group: gradstudent, volunteer		
Possible Directions		
1. Add Alice to the department staff group 2. Allow Alice current group `gradstudent` access "/Application/" 3. Allow Alice current group `volunteer` to access "/Application/" 4. Allow Alice to access only the directory "/Application/2021/"		

Figure 2: **A simplified access-deny issue with root cause in the Apache server configuration.**

misunderstand security context and adopt solutions that may unknowingly introduce security risks [21, 55]. Other times, sysadmins may neglect the security consequences and not thoroughly examine the modifications once the denied request is allowed. Moreover, it takes a large amount of time to understand and analyze the security context. Sysadmins may be under significant time pressure, and have to find a quick workaround to the access-deny issues [29]. These blind spots lead them to resort to risky solutions.

In a perfect world, sysadmins can always find the solution that fits the security context. However, this is hard in real-world scenarios because access-deny issues may have multiple ways to relax the security configuration to solve the issue. Following different ways to change the configuration may lead to different security consequences. Sysadmins with blind spots may miss the most suitable one. We can look at two common access-deny issues in the file system and the server application (Figure 1 and 2).

Figure 1 shows an example of a web request being denied due to file system permission [33]. The root-cause is that the process owner `apache` lacks execute permission on upper-level directories. Many real-life cases [33] with similar root causes often resolve the issues by disabling the protection (`chmod -r 777`) recursively from the home directory, as commonly suggested on online forums [62]. Part of the reason is that they do not know the directory that lacks permission and the required permissions. Another common risky suggestion on the forums [16, 60, 61, 66] is to disable all execute protection (`chown +x`) on the directory and now anyone can traverse this directory. One solution is relatively safe by granting `apache` execute permission while keeping the original owner `Alice`'s permission. While another way is to make `apache` the owner of the directory which hurts the availability of `Alice`. The changes in different directions can resolve the issue, but they have different implications which may not fit the security context.

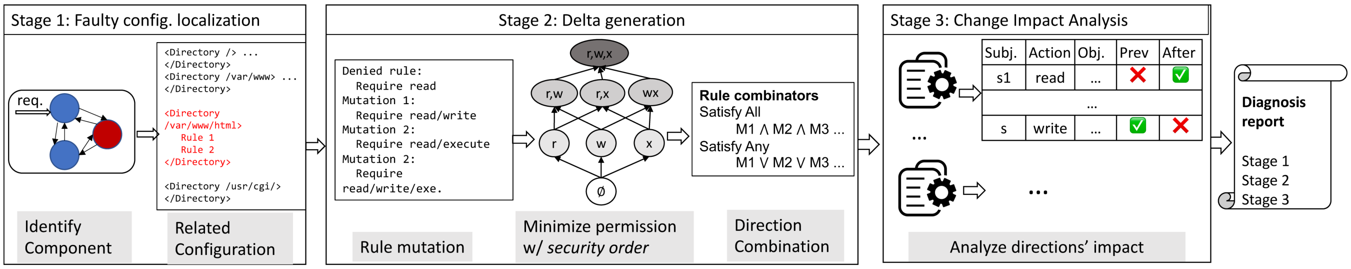
Figure 2 is adopted from a real-world access-deny issues [59] caused by Apache server configuration. The user `Alice`, who is in groups `volunteer` and `gradstudent`, cannot access the directory `/Application/2021` because the directory is only accessible to certain user group `staff`. She needs to serve as a volunteer to only access 2021 year's applications. One safer way is to build a new directory block `/var/www/html/Application/2021` in the configuration and allow Alice to access any data under the directory. However, it is easier for sysadmins to grant Alice to access all data under `/Application/` because they do not need to configure a more specific directory block. Moreover, sysadmins may misunderstand user's role and simply add Alice to the group `staff` which over-grants `Alice` permissions.

These examples show that access-deny issues can have multiple ways to change the access-control configurations, each with different security implications. As a result, it is difficult for sysadmins to manually gather all the possibilities as well as their impact. Their blind spots can prevent them from finding the solution that fits the security context. Without tooling support in the diagnosis process, sysadmins can only rely on their intuition and experience to compose a solution and determine whether it fits the security goal.

As such, it is important to relieve sysadmins' burden of exploring all the different possible directions and assessing the security implications to reduce permission misconfigurations for access-deny issues. Unfortunately, most previous works on misconfiguration diagnosis [22, 26, 36, 42, 43, 83] cannot help sysadmins find possible directions and explore the impact on access control state. Some previous works [64, 84] provide recommendations based on historical resolutions, but many historical resolutions are not safe. One of their tool-generated solutions for access-deny issues is to grant full privileges to every user in the system [64]. Sysadmins might blindly trust the solution provided by the tool and unknowingly introduce security risks. We will discuss more related works in §9.

1.2 Our Contributions

Our work focuses on helping sysadmins find possible directions to fix the access-deny issue and understand the security impact to reduce their chance of making risky resolutions. To achieve this goal, we present a new framework, called Multiview, that utilizes multiple analysis techniques to produce possible changes in different directions to allow the legitimate request and evaluate the impact of configuration changes on



```
<Directory /> ...
</Directory>
<Directory /var/www> ...
</Directory>

<Directory
/var/www/html>
    Rule 1
    Rule 2
</Directory>

<Directory /usr/cgi/>
</Directory>
```

```
Denied rule:
  Require read
Mutation 1:
  Require read/write
Mutation 2:
  Require read/execute
Mutation 2:
  Require
read/write/exe.
```

Subj.	Action	Obj.	Prev	After
s1	read	...	✗	✓
...				
s	write	...	✓	✗

Figure 3: **High level workflow of Multiview.**

each direction. Multiview's key insight is that, through the mutations of configuration entries and their combinations, we can help reveal the possible directions of configuration changes to address the issue which may fall in the blind spots of sysadmins when solving manually. Multiview further compares and minimizes the security impact with the pre-defined partial security order and only presents the directions of changes with the least permissions in the security order.

Multiview faces three major challenges in resolving real-world access-deny issues. (1) How to design a general approach to find the access-control configurations related to the denied request? (2) How to find different directions to solve the access-deny issue and find the minimal permissions to be granted with the changes in each direction? (3) How to systematically evaluate the impact on the global access-control state to help sysadmins find the direction of changes that best fits the security context?

To address the first challenge, we design a general technique named *toggle analysis* to turn on and off access-control checks at different levels and retry the original request. This can narrow down the access-control configurations related to the request. Multiview can help diagnose issues in multiple server applications through customized APIs, which are implemented by the software developers as a one-time effort.

To address the second challenge, Multiview performs mutations on the identified access-control configurations based on the category of each access-control rule in the configuration. This is based on the observation that each rule represents a restriction on the subject, object and action of the request. Multiview can systematically mutate and combine the rules to relax the access control to allow the access. Multiview then uses a predefined *security order* to measure and compare the permissions granted by different configuration changes. Then Multiview collects all the non-comparable directions of changes which each contains least permission changes (§4).

To address the third challenge, Multiview needs to compare the system-level impact of changes in each direction by comprehensively evaluating the access-control result before and after the change. We adopt the approach to replay requests to see the end-to-end access-control state changes on the system. To reduce the number of replayed requests, we develop a method to synthesize and prune the requests based on the configuration changes suggested in each direction.

We evaluated Multiview with 112 real-world cases from eight widely used systems and server applications including Apache HTTPD, Nginx, Vsftpd, Proftpd, PostgreSQL, MySQL, MongoDB, Squid. Multiview can successfully find the directions to change the configuration on 89 out of 112 cases. To evaluate the effectiveness of Multiview's diagnosis report, we conducted a user study with 20 participants (11 system professionals and 9 graduate students) with five real-world access-deny issues. The results of the study show that our tool can help sysadmins reduce the percentage of insecure fixes from 44.0% to 2.0%. Moreover, Multiview can help reduce the diagnosis time by 62.0%.

2 Overview

Multiview Workflow Multiview aims to help sysadmins diagnose access-deny issues by providing possible directions of changes and the security impact of changes in each direction. Multiview's workflow is shown in Figure 3. First, Multiview locates the faulty component and access-control rules through toggle analysis. This narrows down the range of access-control rules that Multiview needs to mutate to find possible directions (§3). Second, based on the located rules, Multiview generates mutations for each individual rule. Multiview then uses the security order to minimize the permissions each mutation grants. Then, Multiview combines the mutations of each rule based on the combinational logic, such as "Satisfy All" and "Satisfy Any", available in the access-control configuration to generate possible directions of changes (§4). Last, Multiview analyzes the impact on the global access-control state for the changes in different directions. Multiview synthesizes the requests based on the available roles and resources, then replays the requests to generate the impact. Multiview may reduce the number of replayed requests by excluding requests not related to the configuration change of the direction (§5).

The generated diagnosis report includes the access-deny related configuration rules, the possible directions of changes to allow the request along with the impact on the access-control state of the entire system after applying the changes in each direction. But still, sysadmins need to determine which change best fits the security context.

Usage Multiview is designed to be used by sysadmins when an access-deny issue arises. To start the diagnosis process, sysadmins need to provide the information of denied request from the log file, or manually specify such information. Multiview performs the analysis in a packed virtualized environment which contains the same settings as in a production environment (§6.1). The analysis process is automatic and generates a diagnosis report for the access-deny issue.

To make one piece of software fit into the Multiview framework, we may require a few inputs from the software's *developers*: (1) The configuration parser to parse the configuration and get the configuration entries; (2) The general APIs provided by Multiview to manipulate the component, configuration objects and access control rules. Multiview provides the common combinational logic to combine the mutations and generate the directions, but the software developers may provide additional logic to combine the rules.

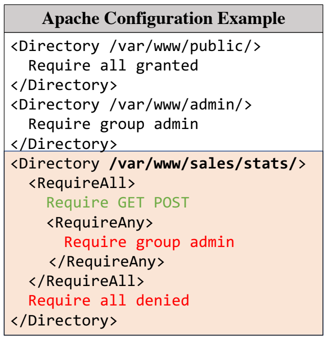
Multiview may require additional inputs from the *sysadmins* if the software's setup is different from common settings. This includes: (1) the software's configuration file for configuration mutation; (2) the information related to the denied access; (3) all subjects and objects in the application, for Multiview to perform the change impact analysis.

3 Faulty Configuration Localization

Multiview first attempts to identify the access-control configurations that introduce the access-deny issue. Multiview takes a black-box approach to mutate the access control configurations by turning on and off the protection at different levels, to identify the access-control rules associated with the denied request. We name this approach *toggle analysis*, as it mimics how sysadmins narrow down the causes of access-deny issue in real-world scenarios.

3.1 Identify Faulty Component

Real-world systems often have multiple components, and each component can customize its access-control rules. Multiview first tries to identify the faulty component that causes the access-deny issue, as the request may need to pass access-control checks for multiple components and may fail in any component. For example in a web application scenario, when a web request tries to access a file following the URL, the Apache web server first needs to check the user's permission according to the server's configuration. After passing the application-level check, the request needs to pass the file system checks on behalf of the end user to access the file. For each component, Multiview uses the predefined API to temporarily remove all the access-control checks thus allowing all users to access all data in the component. If the replayed original request is still denied after removing access-control checks, Multiview can exclude the possible error in this component. The API should be defined by the software developers of each component. We will discuss more about the implementation in §6.



Apache Configuration Example

```
<Directory /var/www/public/>
  Require all granted
</Directory>
<Directory /var/www/admin/>
  Require group admin
</Directory>
<Directory /var/www/sales/stats/>
  <RequireAll>
    Require GET POST
    <RequireAny>
      Require group admin
    </RequireAny>
  </RequireAll>
  Require all denied
</Directory>
```

Figure 4: **A simplified Apache HTTPD configuration example.** A user's request to URL matches with the configuration block for /var/www/sales/stats/ and gets denied by the rules in configuration block. The rule marked in green represents the access-control result for this rule is allowed, while others marked in red are denied.

3.2 Identify Request-related Configurations

3.2.1 Identify Related Objects

Multiview attempts to identify the access-control rules related to the denied request. Server applications and file system usually follow a similar mechanism to design their access-control configuration with the access control list (ACL), where the object has a list of access-control rules for every user and role in the system [20]. Therefore, Multiview first identifies the related objects for which the request was denied, and then finds the access-control rules associated with the objects.

Multiview first needs to identify which objects the user needs to access in the denied request, and then proceed to probe whether the user lacks permissions for each object. A single request may involve multiple objects in the system. For example in the file system, when a user wants to read a file, the user must also have the search permission on upper-level directories [33]. Thus the file permissions of the upper-level directories are also identified as relevant objects. Similarly, the web server configuration is designed to reflect access control to a directory tree structure, as shown in Figure 4. An Apache configuration block will be matched for objects in each directory. Each block represents either file system objects or URLs and contains a set of access-control rules. For a SQL request in PostgreSQL database, the request must pass permission checks at the database, schema, and table level before retrieving the related data [5].

3.2.2 Identity Related Rules

Multiview continues to examine the access-control rules associated with the objects that the user lacks proper permissions in order to find the rules related to the denial. The access-control rules are expressed differently for the object in different types of software. Unix file systems use 9 permission bits to represent access control for owner, group and others; Web applications use directives which can place a restriction on end user's attributes (e.g., IP, method, user/group). The rules can serve two purposes, allow or deny. The first type is the rules to allow, which can allow requests if the request's attributes match the requirement; The other type is to deny. As shown in Figure 4, multiple access-control rules exist in the configuration block, and each rule's result is either allowed (marked in green) or denied (marked in red) for each rule. (We will discuss more about the combination operators in §4.4.)

To find possible problem-solving mutations for the rules, Multiview should find which rules are causing the denials by performing toggle analysis at the individual rule level. First, Multiview examines each individual rule to determine whether the attribute restricted by the rule is allowed or denied. Then Multiview conducts toggle analysis at the single rule level to probe the rules that cause the access-deny issues.

4 Delta Generation

4.1 Delta Generation Overview

Through the toggle analysis at different levels, Multiview locates objects that the user cannot access and access-control rules related to the denial. But still, there are many possible ways for the sysadmins to modify the configuration and grant permissions. Multiview aims to help sysadmins find as many directions as possible by systematically relaxing access-control rules and combining them as possible directions. We classify the ways to relax the access-control rules into three categories based on the components of the request: subject, object and action. Mutations to the access-control rule either relax the system's restrictions on the subject, object or action of the request. Therefore, each possible direction of changes to resolve the access-deny issue is a combination of those variations that relax the restrictions of the system in three different aspects.

To ensure that issues can be carefully addressed with minimum privilege, we developed a new partial order metric called the *security order* to measure and compare the permissions granted by each mutation of access-control rule. Once Multiview detects a mutation of configuration that can successfully permit the request, Multiview collects the mutation for later combinations to generate final directions of changes. Multiview only provides more insights on possible directions to solve the access-deny issue, not final solutions. To find the best suitable solution, sysadmins should further confirm and determine the solution that fits the security context of the organization.

4.2 Access-Control Rule Mutation

With toggle analysis, Multiview finds objects for which the user does not have enough permission, and the access-control rules on each object that deny the request. Multiview further mutates each rule to lose the restriction on different attributes of the rejected request respectively to cover all the possible modifications, then trim the mutations with the help of security order (§4.3).

We classify methods of mutating access control into three categories: relaxing subject, action, and object. We denote the subject, action and object in the access-denied request as s_{deny}, a_{deny} and o_{deny}.

- Relax s_{deny}: Multiview can assign user to different roles in the access-control configuration or allow user-specific attributes. By changing s_{deny} to a role with more privileges, s_{deny} can gain access to the denied object.
- Relax a_{deny}: Multiview can grant s_{deny} with more privileges to perform actions on the same object.
- Relax o_{deny}: Multiview can allow s_{deny} to access new objects (including the denied object). This can be done by adding new access-control rules for the denied objects based on the resource type.

Multiview needs to find the possible ways to relax restrictions on s_{deny}, a_{deny} and o_{deny}. From the results of faulty configuration localization, Multiview can identify o_{deny} as the objects that users need more privileges to access. The access-control restrictions on s_{deny} and a_{deny} are expressed as the rules associated with the object. Multiview cannot know the exact mutations of the rules to be able to allow the original request, such as changing to certain roles or granting specific permissions. Therefore, Multiview conducts speculative *delta generation* of access-control rules.

Multiview aims to mutate the access-control rules based on the system information and the characteristics of the rules to generate possible combinations to permit the access-deny issues. First, Multiview collects all the roles in the system that are allowed to perform the action a_{deny} on the object as S_a, and all the roles s_{deny} currently in as S_d. We denote all the roles as $S = S_a \bigcup S_d$. Second, Multiview assigns collected possible roles $s_0 \in S$ to s_{deny}, and then further relax the restriction of s_0's permissions. The relaxation is conducted based on the specific types of the access-control rule of o_{deny}. If the role s_0 is not allowed to access the object, Multiview would try to add the role to the object's access-control list with the different types of privileges.

4.3 Minimize Permissions w. Security Order

Multiview may produce multiple rule mutations to change the result of a single access-control rule. For example, when we want to relax the user's privileges on the object, we can grant (1) execute, (2) execute and read, or (3) execute, read and write, which all can allow the request but the amount of granted permissions are different. Since granting excessive permissions may introduce security risks, Multiview should prune these rule mutations to avoid misleading sysadmins.

We define a partial security order to compare the changes on different directions. This can be served to early terminate mutation process — if one mutation of the rule can allow the access-deny request, there is no need to try mutations with more permissions. By pruning redundant rule mutations, we can reduce Multiview's diagnosis time and avoid including too permissive mutations.

To formally define the security order, we denote the set of users, objects and actions in the system as S, O, and A respectively. Then, one *access* can be represented as a tuple $(s,a,o) \in (S,A,O)$. Given a configuration c, we can then define the binary function $\alpha_c : (s,a,o) \mapsto \{0,1\}$. $\alpha_c(s,a,o) = 1$ if and only if the access (s,a,o) is allowed by the server with the configuration c.

Definition 1 (Permissible Access Set) *Given a configuration c, define its access status image as* $R(c) := \{(s,a,o) \in (S,A,O) | \alpha_c(s,a,o) = 1\}$

Definition 2 (Partial Security Order) *Given two configurations* c_1, c_2, $c_1 \preceq c_2$ *if and only if* $R(c_1) \subseteq R(c_2)$.

Two rule mutations, which lead to configurations $c_1$and c_2 after the changes have been applied, can both solve the access i.e., $\alpha_{c_1}(s_{deny}, a_{deny}, o_{deny}) = \alpha_{c_2}(s_{deny}, a_{deny}, o_{deny}) = 1$. If $c_1 \preceq c_2$, c_1 will be a more secure solution if we regard less access as more secure.

Handle incomparable possible rule mutations Note that the ordering for the access status result of configuration is a partial order. There are situations where two mutations are not comparable, i.e., when the symmetric difference of $R(c_1)$ and $R(c_2)$ is not empty, we can not determine which one grants less access based on the partial security order. For example, Multiview can generate mutations by either assigning the user to a high privileged role, or granting user's current role more permissions to access the object. Both can allow the request, but because the roles are different, the access result cannot be subset of each other. Then the two mutations bring different impact on the access control result. Multiview collects all the non-comparable rule mutations to generate the final directions of changes for sysadmins to determine which one to choose based on the security context.

4.4 Generate Final Directions

Multiple rules may exist in an object's ACL to represent the access control policy. When multiple rules associated with the object may deny the request, Multiview first generates possible rule mutations to resolve each rule and then combines the mutations based on the relationship between the rules. For example, in the web server configuration scenario (Figure 4), multiple rules are associated with the configuration block to provide access control on the subject or actions.

The mutations of access-control rules can be combined together to represent the final access-control result. We define the set of related access-control rules as P, and each access-control rule as $p_i \in P$. We denote the set of mutations for each rule p_i as C_{p_i}. We define the result of each rule p_i as r_i, where $r_i \in \{True, False\}$. $True$ represents the represents the access control is allowed while $False$ represents denied. Based on the common practice of access-control configurations, we find that here exists four combination operators for the directive's result, Satisfy All($\wedge$), Satisfy Any ($\vee$), Negate ($\neg$) and First Match ($\odot$). The rules' results can be combined with the above operators to represents the final access-control result.

1. For Satisfy All($\wedge$), the final result is $r_{final} = \wedge_{p_i \in P} p_i$. Then the final directions of changes to make the request to be allowed are $\mathbb{C} = \mathop{\Large\times}_{C_x \in \{C_{p_i} | r_i = True\}} C_x$, which represents all the rules needs to be allowed to make one final direction.
2. For Satisfy Any($\vee$), the final result is $r_{final} = \vee_{p_i \in P} p_i$. Then the final directions of changes are to make the request to be allowed are $\mathbb{C} = \bigcup_{C_x \in \{C_{p_i} | r_i = True\}} C_x$, which represents we can select any of the direction to allow one rule to make one final direction.
3. For Negate($\neg$), the final result is $r_{final} = \neg(\wedge_{p_i \in P} p_i)$. Then the final directions of changes are to make the request to be allowed are $\mathbb{C} = \mathop{\Large\times}_{C_x \in \{C_{p_i} | r_i = False\}} C_x$, which represents all the rules needs to be denied to make one final direction.
4. For First Match($\odot$), the final result is $r_{final} = p_\odot$ where $p_\odot \in P$ is the first matching rule. Then the final directions of changes are to make the request to be allowed are $\mathbb{C} = \{C_{p_\odot} | r_\odot = True\}$, which we can select any of the direction to allow $p_\odot$. If no matching rule exists, the final result is the default value.

The combination orders can be retrieved from the configurations based on keywords. For example in Apache configuration (Figure 4), the keyword `RequireAll` represents Satisfy All, and the keyword `RequireAny` represents Satisfy Any [41]. Multiview requires the annotation for such keywords for the software's configuration to reason about the access-control result. In some common system components like file system, the combination operator is implicitly implied as Satisfy All, as the request needs to pass the access-control checks on all the upper directories on the requested file.

5 Change Impact Analysis

Multiview further inspects the impact of each configuration change on the global access-control state. Some configuration changes that can resolve the access-deny issue may also change the access-control status of other resources in the system. Sysadmins need to be aware of the access-control result changes to judge whether these changes are intended based on the organization's security context. Therefore, Multiview applies *change impact analysis* to analyze the impact of configuration changes on the global access control state. Furthermore, sysadmins can compare the changes from different directions suggested by Multiview by examining the access-control state changes, which Multiview can not directly compare with the security order.

Previously we denote a configuration as c and the access control state as $R(c)$. After applying one possible change on the original configuration $c_{original}$, Multiview gets a new configuration c_{new}. Given two configuration settings $c_{original}$ and

Component	Subject	Object	Action
File System	OS users	Files	Read/Write/Execute
Apache	Webserver users	Web pages	GET/PUT/POST
DB	DB users	Table	SELECT/INSERT/DELETE

Table 1: **The example subject, object and action in different software systems.**

c_{new}, the impact of a configuration change can be represented as a set of tuples (s,a,o,r,r') where $r = \alpha_{c_{original}}(s,a,o)$ and $r' = \alpha_{c_{new}}(s,a,o)$ and $r \neq r'$. The subject, object, action could be different in various software to represent the end-to-end access control state as shown in Table 1.

Comprehensive requests generation To characterize the global access-control state comprehensively, the impact analyzer needs to collect all the subjects, objects, and actions. For example in the file system, the impact analyzer may collect all the users in the OS as subjects, and walk through the directories to collect files as objects. And the actions would include read, write and execute. Sysadmins may also customize the scope to limit generated requests only for the objects of interest. For example, sysadmins may limit the objects only in a certain file directory.

Requests reduction In a production system, the number of subjects and objects in the system could be large and a large number of requests are generated, which results in a long analysis time for one configuration change. To overcome this challenge, we leverage the characteristics of rule changes to reduce the number of related subjects and objects. We will discuss more about the implementation in §6.4.

6 Implementation

This section describes the general implementation methodology for all applications. First, we discuss the general considerations for implementing Multiview framework, including configuration parsing and safe replay. Second, we discuss the detailed implementation for all three stages, including toggle analysis, delta generation and change impact analysis.

6.1 General Considerations

Configuration Modeling The configurations format and semantics are vastly different among different applications. They can be generally classified into two types (1) configuration in text files and (2) configuration in software-specific settings. Multiview designs a unified nested JSON format to provide APIs mutation. This requires *developers* to parse the configuration into Multiview provided format, as well as the API to transform JSON to the software-specific configuration format. Note that the configuration parsing is a one-time effort as the configuration format changes much less frequently than software features or other functionalities.

Configuration in text files are represented as simple key-value pairs, while in complex configurations are represented in XML or similar format. This kind of configuration is usually loaded during starting up in most applications. To process this type of configuration into unified JSON format, developers either utilize existing configuration parsers [13, 14], or design a parser for the configuration format if no such parser exists.

The access-control configurations in some applications are stored in software-specific settings, such as file systems and databases. (1) For file systems, the permissions are represented as permission bits associated with the file. (2) For databases including PostgreSQL, MySQL and MongoDB, the access-control settings are stored as system privilege tables. To process this type of configuration, developers need to design functions to read the permission settings and store them as key-value pairs in a JSON format.

Safe Replay Multiview needs to apply configuration changes and replaying requests to analyze the access-control result in all three stages. This needs to be done in the same environment as the production data, to ensure the access-control results are the same as in production. However, randomly applying changes and replaying requests directly in the production environment may bring side effects on the data and affect production performance. Besides, it is hard to ensure the separate environment to be consistent with the production.

Multiview tackles this challenge by (1) replicating the production environment with virtualization techniques to ensure faithful replay results and (2) making safe mutations without affecting production data. **First**, Multiview uses virtualization techniques to prepare a safe environment separate from production. Nowadays most cloud systems adopt virtualized environment (e.g., containers) to deploy services. Multiview reuses the virtual environment to achieve isolation from the production. **Second**, Multiview ensures the access-control configuration changes and replay do not affect production data with several techniques. Multiview uses copy-on-write techniques to share the same copy of data and configurations as production [40, 50]. When a modification happens, Multiview will create a copy of the original data and the modification is only applied to the modified version without affecting other production systems. For replaying the SQL commands or queries in databases, Multiview utilizes the transactions to wrap mutation commands in a transaction to safely roll back.

6.2 Toggle Analysis Implementation

Multiview requires some APIs to be implemented by *developers* to perform the toggle analysis at different levels, including components, objects and specific rules. The toggle analysis builds on the assumption that each unit's access-control works independently from others. On the component level, server software, as we observed in current practice, treats other components as black boxes and relies on error codes to communicate. The access-control checks are performed layer by layer and are therefore independent of each component. On the object and rule level, the system checks each object's permission respectively to get the final access-control result.

Category	Definition & Example
Relax Subject	**Def.:** Relax access control by assigning users to new roles.
	Ex.: Add a user to an existing group or create new groups.
	Ex.: Change the file owner to be the user in file system.
Relax Action	**Def.:** Grant subjects more privileges on the denied object.
	Ex.: Allow more HTTP methods such as POST/OPTIONS.
	Ex.: Grant users execute permission to the scripts.
Relax Object	**Def.:** Allow subjects to access previously inaccessible objects.
	Ex.: Add new configuration blocks to allow users access the directory.
	Ex.: Create new resource groups for users to access.

Table 2: **Definitions and examples of different types of modification that can relax the access control and permit the access-deny issues.**

At the component level, Multiview treats each application as one component. To toggle the component, *developers* need to implement the API to disable access-control checks. This can be done by granting all permissions on the accessed objects to user (e.g., recursively grant 777 to user in file system; all privileges in databases); or remove all access-control configurations (e.g., web servers and FTP servers).

At the object level, Multiview performs toggle analysis in two steps. First, Multiview finds all the relevant objects based on the characteristics of the denied object. For example, the access to table in a database requires privileges at both table level and database level. The access to file also requires permission on upper level directories. Second, Multiview identifies the objects that lack permissions by testing each object' access control by granting all the other objects with all privileges.

At the rule level, Multiview performs toggle analysis on the rules associated with the objects to evaluate each individual rule is allowed or denied. Multiview evaluates each rule's result by only keeping the rule associated with the object and replaying to get its result.

6.3 Delta Generation Implementation

Delta generation help sysadmins find possible directions by (1) relaxing individual access-control rules to generate rule mutations and (2) combining them with combination operators to generate possible directions. We classify methods of mutating access control into three categories: relaxing subject, action, and object based on the rule type. Developers need to write the mutation for each rule type as a one-time effort.

For rule mutations, Multiview relaxes the rules based on the rule type. For subject, Multiview collects all the possible roles in the system and assign the role to the denied user. The security order to compare the roles is that, if the role's privileges are a subset of another role, Multiview prunes the role with more privileges. For action, Multiview grants the denied user more privileges. The security order is that if the granted permissions of one mutation are a subset of the other, Multiview removes the one mutation with excessive permission. For object, Multiview creates more specialized rules for the denied object in the configuration as an exception, by only allowing the denied user to access the block.

For rule combinations, Multiview identifies the combination operators in the configuration with keywords or specifies default combination logic in each application. Multiview stores the combination operators with the related rules in a nested JSON format, which can be further translated to the software-specific configuration format.

6.4 Change Impact Analysis Implementation

Multiview applies change impact analysis to analyze the impact of configuration changes on the global access control state. Multiview first synthesizes the requests based on the subjects, actions and objects in the system. Sysadmins need to provide APIs to collect all possible roles and objects in the system. The actions on the objects are the predefined privilege set by the developers.

To reduce the number of requests, Multiview leverages the characteristics of rule changes to reduce the number of subjects and objects in the synthesis.

- Reduce requests based on objects: If the rule change is only related to a specific object (e.g. a file directory), only the access control states of files under this directory are affected. Therefore the impact analyzer only needs to walk through the directory to collect the objects.
- Reduce requests based on subjects: If the rule change only involves a certain role (e.g., adding a user to a role), only the users with the specific role are affected. Therefore the impact analyzer only needs to collect all the users with the affected role as the subjects.

If the configuration change involves multiple rule changes, the subjects, actions and objects would be the union from each rule changes to keep all the possible access-control result changes [41]. We discuss more details about the implementation for each application in supplementary materials [15].

7 Evaluation

We evaluate Multiview's effectiveness and efficiency with three sets of experiments.

First, we evaluate whether Multiview can effectively diagnose real-world access-deny issues. We collected and reproduced 112 access-deny issues for eight large systems and server applications. Then we used Multiview to diagnose these issues to see whether Multiview can generate helpful directions to resolve the access-deny issues.

Second, we evaluate whether Multiview can help sysadmins find solutions that fit the security context for access-deny issues. We designed a user study based on real-world issues and conducted it with 20 system professionals with and without Multiview's diagnosis reports.

Third, we evaluate the effort in adopting Multiview and Multiview's diagnosis time. We report the effort of customization for each server application, and the diagnosis time for the access-deny issues in a real-world environment.

Appl.	No	Description of access-deny issues	Config	Directions
FS	1	Apache uses the `mod_wsgi` module to redirect requests to python scripts. However, Apache cannot access the location of the python scripts.	Yes	2
	2	In a shared server, a web service denies all requests because the Apache process user does not have access to multiple directories for this user.	Yes	10
Apache	3	Anonymous requests are blocked because the default configuration of Apache blocks the request method.	Yes	2
	4	Authenticated users cannot access internal web pages due to improper implementation of role access-control configuration in Apache.	Yes	4
Nginx	5	The Nginx configuration blocks a range of IP addresses with an IP masking that accidentally included the IP address of the load balancer.	Yes	1
Vsftpd	6	The configuration entry `deny_file` contains an incorrect settings that matches all filenames, causing all FTP commands to be access denied.	Yes	1
Proftpd	7	One user could not login because the configuration only limits login from a user group, and the user is not in this user group.	Yes	3
MySQL	8	One SQL query on Table A involves foreign key reference to Table B gets denied, because the user only has no references privilege to Table B.	Yes	2
PostgreSQL	9	One user's SQL query to run a function in one schema gets denied, because the user lacks permission on the schema.	Yes	2
MongoDB	10	User can not query the stats of a database with `dbStats` command, because the user lacks permission to run administrative commands.	Yes	2
Squid	11	One user's connections to websites are denied by the Squid configuration, because the user's IP address matches one subnet IP range block list.	Yes	1

Table 3: **Results for 11 representative real-world access-deny issues.**

We conducted all Multiview's analyses on a single machine with Intel Core i7-7700 CPU (3.6GHz, 8 cores), 16 GB memory, 1 TB HDD running a Ubuntu 18.04 distribution.

7.1 Real-world Access-Deny Issues

7.1.1 Methodology

We choose popular open-source applications as our targets including web servers (Apache, Nginx), FTP servers (Vsftpd, Proftpd), databases (MySQL, PostgreSQL, MongoDB) and network proxy server (Squid). We collect a set of access-deny issues from mailing lists [2, 8, 10, 11] and sysadmin online forums [1, 3, 4, 6, 7, 9]. We search for issues related to applications by tags. Then we filter issues that contain access-control related keywords such as "permission", "access control", "access denied" or "forbidden". We only select issues that have comments that are marked as answers. 453 cases are crawled, and we examine each case and exclude issues not related to access-deny. In the end, we have 186 access-deny issues.

Next we try to reproduce these cases based on the description and root causes by recreating their server configuration files and file system permissions. We have to exclude 74 cases with no concrete causes which we cannot reproduce. We reproduced 112 cases. Multiview can diagnose 89 cases of them. We select 11 common cases from all software to present in Table 3 along with Multiview's report summary including related configuration, possible directions of changes in Appendix.

7.1.2 Results

For each access-deny issue in Table 3, we present the results of (1) fault localization: whether we can find the specific component and configurations related to the access-deny issue and (2) delta generation: whether we can generate multiple directions of changes that may resolve the access-deny issue. More details about the results are in Appendix A. For the change impact analysis, because we do not have the data from the original issue, we do not have diagnosis results for the impact analysis. We will discuss more about the usefulness of impact analysis in the user study (§7.2) and real-world scenario evaluations(§7.3).

For all the cases that Multiview can successfully locate to the configuration, Multiview can find the possible directions based on the combination of mutations of subject, action and object. Multiview can successfully find 1-10 directions of changes that can resolve the access-deny issue. Sysadmins determine the solution based on the access-control context. Note that the directions found by Multiview also cover original post's solution that is regarded as secure based on the problem context. We use three case studies to show how Multiview's reports can help sysadmins diagnose.

Case 1: File System Figure 5 shows a common file system

<table>
<tr><th colspan="2">Counselor Diagnosis Report for Application – Apache FS</th></tr>
<tr><th>Denied request information</th><th>Faulty configuration</th></tr>
<tr><td>File: /var/www/scripts/index.py
User: apache
Method: Read</td><td><b>File:</b> /var/www/scripts/
<b>Permission:</b> drwxr----- alice apache
← group lacks execute perm.</td></tr>
<tr><th colspan="2">Directions to allow the request</th></tr>
<tr><td colspan="2">Direction 1: Grant group apache with execute permission on /var/www/scripts/
sudo chmod g+x /var/www/scripts/</td></tr>
<tr><td colspan="2">Direction 2: Change apache to be owner of the directory
sudo chown apache:apache /var/www/scripts/</td></tr>
</table>

Figure 5: **Multiview diagnosis report for case 1.**

<table>
<tr><th colspan="2">Counselor Diagnosis Report for Application – Apache Configuration</th></tr>
<tr><th>Denied Request Information</th><th>Faulty configuration</th></tr>
<tr><td>URL: /var/www/sales/stats/
User: Alice
Method: GET
Alice's groups: sales, sales_manager</td><td>&lt;Directory /var/www/sales/stats/>
&lt;RequireAll&gt;
Require method GET POST
&lt;RequireAny&gt;
Require group admin ← not passing
&lt;/RequireAny&gt;
&lt;/RequireAll&gt;
&lt;/Directory&gt;</td></tr>
<tr><th colspan="2">Directions to allow the request</th></tr>
<tr><td>Direction 1: Add Alice to admin group in the configuration
admin: sysadmin, Alice</td><td rowspan="2">Direction 3: Allow Alice's current role sales to access the block
&lt;RequireAny&gt;
Require group admin
Require group sales
&lt;/RequireAny&gt;</td></tr>
<tr><td rowspan="2">Direction 2: Allow Alice's current role sales_manager to access the block
&lt;RequireAny&gt;
Require group admin
Require group sales_manager
&lt;/RequireAny&gt;</td></tr>
<tr><td>Direction 4: Only allow Alice to access
&lt;RequireAny&gt;
Require group admin
Require user Alice
&lt;/RequireAny&gt;</td></tr>
</table>

Figure 6: **Multiview diagnosis report for case 4.**

<table>
<tr><th colspan="2">Multiview Diagnosis Report for Application – PostgreSQL</th></tr>
<tr><th>Denied request information</th><th>Faulty configuration</th></tr>
<tr><td><b>DB user:</b> joe
<b>SQL query</b>:
SELECT * marketing.runStats();</td><td><b>Database:</b> companydb
<b>Schema:</b> marketing ←lack USAGE
<b>Function:</b> dailyStats ←lack EXECUTE</td></tr>
<tr><th colspan="2">Directions to allow the request</th></tr>
<tr><td colspan="2">Direction 1: Grant USAGE on schema and grant EXECUTE on function to joe
GRANT USAGE ON SCHEMA marketing TO joe;
GRANT EXECUTE ON FUNCTION marketing.runStats TO joe;</td></tr>
<tr><td colspan="2">Direction 2: Grant the role of schema's owner to joe
GRANT marketing TO joe;</td></tr>
</table>

Figure 7: **Multiview diagnosis report for case 9.**

access-deny issue on forums. Many of the suggested answers are risky. The denied web request attempts to read the data under the directory /var/www/scripts/ but lacks execute permission on the directory. Many administrators did not know that Apache process user not only needs read permission of the requested file, but also needs to execute permission on all the upper-level directories. Users sometimes do not know how to quickly examine the permissions on all the upper related directories and may try to grant unnecessary privileges for each path. Even if the users know that the directory /var/www/scripts/ lacks the execute permission, they often over-grant execute permission to all users instead of to the group, which is the current role of Apache process user. Besides, Multiview can only grant the group execute permission to the related directory; Multiview also mutates the configuration to make Apache process user the owner in case this fits more in the context. Based on the security order, Multiview does not change the Apache process user to be the role "other" of the directory since it would be considered more permissive than its current role "group". Both directions of changes can resolve the access-deny issues and grant fewer privileges compared with forums' answers.

Case 2: Apache Configuration Figure 6 shows an issue caused by Apache server's configuration. User Alice is trying to access the internal web page and she has two roles, sales and sale manager. However, her access is blocked because current configuration only allows admin to access this page. To resolve this issue, Multiview first narrows down to the related configuration sections. Multiview further finds the possible directions of changes by changing the Alice's role to groups that have access to the object, or granting Alice's roles to access this directory, or simply making an exception to only allow Alice to access.

Case 3: PostgreSQL Permission Figure 7 shows an issue caused by PostgreSQL database permission. User Joe is trying to run a function in the marketing schema to get stats. However, his access is denied because he lacks permission on both the schema and function. To resolve this issue, Multiview first narrows down to the related objects including the schema and the function. Multiview further finds the possible directions of changes: (1) granting the necessary permissions on each object to allow the access, (2) granting Joe the role of the object owner which also contains the required privileges. Sysadmins need to determine which direction is most suitable based on the scenario with the help of change impact analysis.

Unsuccessful diagnosis cases In total, there are 23 cases that Multiview cannot provide helpful diagnosis results. We classify the cases that Multiview is unable to handle into three categories. **First**, 13 cases are in the unknown components which are not covered in Multiview's current implementation. 12 cases are related to SELinux permissions in Apache, Nginx and Squid. SELinux has diagnosis tools such as audit2allow. Multiview can integrate the tool for SELinux in the future to enable diagnosis for this component. The other case is related to the connection settings in MySQL configuration file. Currently Multiview only supports MySQL access-control rules within the database, not rules in the configuration file which are read in the start-up stage. **Second**, six cases are related to the rules involving third-party modules. Multiview can narrow down to the rules within the configuration file, but can not provide meaningful delta generations. Four cases are related to the mod_security and mod_ssl modules in Apache. Two cases are related to the third-party authentication module. All these modules provide access-control functionality with rules that are not included in software's configuration, which Multiview can not access or provide rule mutations. **Third**, the remaining four cases are related to syntax errors, which

Name	Description
FS-1	Users could not download the shared file because they are not the correct Unix user group.
FS-2	Users could not access web pages because the server lacks permission to traverse the file system directory.
Config-1	Users could not access web pages because they are not in correct user group in server configuration.
Config-2	Users cannot access web pages because the IP address is wrongfully on the blacklist.
Config-3	Users cannot access a restricted web page because the current configuration misuses logical operators.

Table 4: **Problem descriptions of user study.** All the problems were designed based on real cases. Each question represents a common category of access-deny issues.

are caused by the syntax change between the versions. With syntax errors, Multiview cannot correctly parse and locate the faulty configuration. We include additional discussion on Multiview's limitation in §8.

7.2 User Study

7.2.1 Methodology

We conduct a controlled user study to evaluate if Multiview can help sysadmins diagnose the access-deny issues and fix them securely based on the security context. We design based on real-world problems and add security context for each question which is based on the questions in online forums. We ensure the security contexts in the scenarios are easy to understand based on the problem description and environment settings. Each question is drawn from different types of access-deny issues, including file system user and group, file system access methods, application role and identity, IP, and complex combinational operators, as shown in Table 4. Our study was approved with an IRB exempt status.

Design Our experimental setup includes one warm-up question with five experiment questions. The warm-up question help them to understand the experiment process, and get familiar with the server's environment settings, e.g., configuration location and installation location. Then we randomize the order of five experiment questions. For each question, we provide the information related to the denied request. Users can fix the problem in a virtual machine with access to all related log messages, data, and configurations. To further simulate real-world situations, we also allow users to use search engines to search for log messages or related commands.

For our control and treatment condition, each participant will have 2-3 questions with the report and 2-3 questions without the report. We randomly assign the numbers for each participant. In total, for each problem, we ensure that the numbers of participants in the group with the diagnosis report and the group without report are the same. Our diagnosis report includes four parts: (1) access-control configurations related to the request §3, (2) the specific rules related to the denial §3, (3) the possible directions of changes to resolve the problem §4 and (4) the change impact analysis §5. The user study materials are available here [12].

During the study, we monitor and record the participants' attempts for each question. Based on their modification, we judge whether they resolve the access-deny issues. Then we recruit a security expert to evaluate whether the modification might pose a security risk based on the question context. We also record the time it took them to complete each question, up to a maximum of 30 minutes. This penalizes the group with diagnosis report as the total time could be much longer.

To judge whether the participants' solutions are secure, we ask a security expert to evaluate each participant's attempts. The security expert first receives the same system environment and problem description. They suggest possible measures to resolve the issue securely. After that, the expert evaluates the resolutions from participants and decide whether these solutions pose a security risk to the system based on the security context, and if so, what the risk is.

Recruitment We recruited our participants from the server's mailing list, reddit and CS graduate student slacks. In total, we recruited 11 work professionals and 9 graduate students in total. All the graduate students major in computer science and they have taken computer system courses and are familiar with Linux server administration.

Ethical Considerations Even though the study received an IRB exempt status, we still followed all the requirement and best practices. (1) All the researchers were trained with ethics for user research before the study. (2) Before the study, the participants are provided with consent form approved by the IRB office, which informs the participants about the purpose, procedure, risks, benefits and other information. The participants are informed that they can withdraw from the study at any time without penalty or loss of benefits. (3) None of the graduate students participants are supervised by any researchers involved in the study. (4) No personal identifiable information is collected. (5) The study is voluntary and no compensation is made.

Limitation The user study naturally has limitations that may make it different from real-world measurements. We took several measures to reduce the bias in the experiment. (1) We recruited 20 participants with relevant server management experiences; (2) We shuffled the problem order and randomly assigned Multiview diagnosis reports for each problem. (3) We designed warm-up questions to help the participants get more familiar with the software and environment.

7.2.2 Results

Table 5 shows the percentage of participants who introduced security issues during user study. Note that even though one participant may make multiple security mistakes when solving one problem we only count it as one insecure fix for one participant in one problem. In total, for each problem we have 10 participants with report and 10 without report. We found

	FS-1	FS-2	Config-1	Config-2	Config-3
w/ report	0	0	10%	0	0
w/o report	0	50%	70%	60%	40%

Table 5: **The percentage of participants who made security mistakes for each problem in user study.**

that the percentage of participants in the treatment group that introduced insecure fixes was lower for each question compared to the control group.

We find that 70% of the participants ($n = 10$) in the group without report made security mistakes in problem Config-2. This problem describes the scenario that in graduate student admission, one student volunteer could not access this year's application web pages. The root cause is in the server configurations, only admission officers are allowed to access this folder. We find that the participants made security mistakes in three categories. (1) Wrong component: Two participants could not figure out the cause of this issue and tried to relax the folder's file permissions. (2) Too much permission to the user: Four participants simply added the denied user to the admission officer group, while the admission officer can access sensitive files like financial records in other directories. (3) Grant access to too many users: Two participants granted all the students to view the application web page, even though they are not volunteers of students admission. Note that one participant in the group with diagnosis report also made a mistake that added user to the admission officer. This is because this participant did not pay enough attention to the diagnosis report and ignored the impact analysis results.

In problem FS-2, we find that 50% of participants ($n = 10$) in the control group made security mistakes. The root cause is that the user lacks execute permission on the upper-level directory in the file system. We observe two categories of insecure fixes. (1) Wrong components. Two participants thought this was due to configurations and they went through a trial-and-error approach trying to allow all users in the configuration. (2) Grant too much file permission. Three users granted too much permission to the files other than the directory that lacks permission. Note that they made insecure changes led by suggestions from sysadmin forums. All of the three participants googled the log message, *"access to '/' denied (filesystem path '/home/alice/pages/hello.py') because search permissions are missing on a component of the path"*. This has been a common issue on sysadmin forums with various posts [16, 60, 61, 66]. The participants simply copied the commands from the forum which recursively granted execute permissions to all the related directories and users (all other users on the Unix system). The security expert rates this to be unsafe as it allows other users to traverse the directories. This also shows that even with log messages, sysadmins may still make insecure fixes.

Notice in FS-1, no users made security mistakes. FS-1 is a simple issue that requires minimal diagnosis. Multiview did not improve the performance of participants because the problem was already easy to resolve.

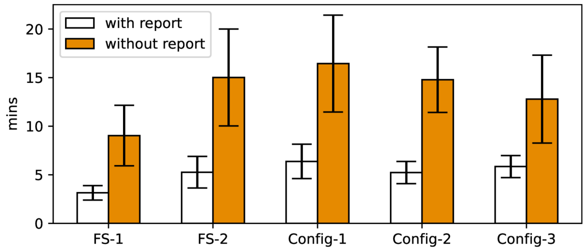


Figure 8: **The average completion time for each user study problem.** The error bar shows the 95% confidence interval.

Application	# LOC
Multiview Framework	2035
File system	256
Apache	482
Nginx	364
Vsftpd	280
Proftpd	320
MySQL	292
PostgreSQL	305
MongoDB	248
Squid	186

Table 6: **Number of LOC for Multiview implementation.**

Besides the percentage of users who made security mistakes during the user study, we find that the average completion time in the group with Multiview's diagnosis report are significantly lower than the group without diagnosis report for all problems (p<0.005, Mann-Whitney U test). Even though the participants in the group with Multiview's diagnosis report are required to read extra materials in the diagnosis report, they spent less time in resolving the issues and introduced fewer security mistakes.

7.3 Performance and Adoption Efforts

Adoption efforts Multiview is implemented in Python3. Multiview requires some efforts from the software developers to adopt it in diagnosing access-deny issues for each application. The main efforts are spent on parsing the configuration file and implementing APIs to manipulate configuration files for toggle analysis and delta generation. The adoption effort for one application is a one-time effort, but it would greatly benefit the sysadmins in diagnosing sensitive access-deny issues as shown in §7.2.

Diagnosis time We calculated the running time of Multiview diagnosis process the reproduced cases and all cases were under one minute, which is negligible compared to manually collecting the system's information or asking on online forums. Note that the time does not include change impact analysis, because we do not have the system's real resources for the reproduced cases. The time may vary depending on the size of the system. Based on the experiment on the data of user-study cases, Multiview can replay 520 requests per second on average.

Real-world systems	System Type	Resources	Requests
Department Courses	Nginx web server	296,036	5.05 B
Research Lab	Apache web server	58,436	8.06 M

Table 7: **Real-world systems for evaluation.** *Resources*: the total number of resources (files, directories) in the system; *Requests*: the total number of requests to run for every change impact analysis without reduction.

Real-world systems	Total	Dir. 1	Dir. 2	Dir. 3	Time
Department Courses	18,348	17,070	1,266	12	28.8 sec
Research Lab	38,460	7,554	30,900	6	56.9 sec

Table 8: **Change impact analysis performance in real-world systems with request reduction techniques.** *Total*: the total number of requests replayed to complete the change impact analysis; *Dir*: direction in short, the number of requests replayed for each direction; *Time*: the total time to finish the change impact analysis.

Change impact analysis time We measured the performance of Multiview's change impact analysis in real-world systems. We collected configurations and data from two real-world deployed systems, including all course websites for a university department, and all research resources for a research lab website, as shown in Table 7. We reproduced two real-life access-deny issues to show how Multiview's change impact analysis can scale in real-world systems.

Case 1: Department Courses We reproduced an access-deny issue taken from real events. A new user u_{deny} is denied to change the course material. The denied request attempts to modify the object o_{deny} under a course directory. Multiview detects that even though u_{deny} has read access to o_{deny}, it does not have write access. To resolve the issue, Multiview proposes a total of three directions to grant u_{deny} the proper permission. The basic way to synthesize all possible requests by emulating the number of resources, users and access methods in a large-scale system can result in 5.05 billion requests for each analysis. After reduction, Multiview generates 17,070, 1,266, and 12 requests receptively for each directions, as shown in Table 8. Multiview in total replayed 18,348 requests, finished in 28.8 seconds.

Case 2: Research Lab We reproduced an access-deny issue which denies an authenticated user u_{deny2} from accessing the machine reservation log o_{deny2}. Multiview first locates the faulty component to be the Apache server. The directory d_{deny2} which o_{deny2} belongs to is only accessible to the internal lab students group $g_{internal}$, which u_{deny2} does not belong to. Multiview suggests three directions, and performs change impact analysis for each direction respectively. If Multiview replays all possible access, it can reach to at least 8.06 million requests for each analysis. After reduction, Multiview generates 7,554, 30,900, and 6 requests receptively for each directions, as shown in Table 8. Multiview in total replayed 38,460 requests, finished in 56.9 seconds.

8 Limitations and Discussion

Can Multiview completely replace sysadmins in access-deny issue diagnosis? Multiview's goal is to help sysadmins collect the possible directions of changes and evaluate their impact to understand and fix the access-deny issue correctly. However, Multiview can not determine which should be taken as solution based on the security context—sysadmins are the ones who make decisions, which inevitably may suffer from human errors. As shown in our user study, while with Multiview's diagnosis report, the participants are less likely to give insecure solutions, there is still one participant who made insecure fixes, because he fixed the issue based on his intuition and ignored the results of the change impact analysis. Future work may explore automatic ways to determine the final solution with proper models of the security properties.

Can Multiview handle all configuration mechanisms? Multiview takes a black-box approach in the faulty configuration localization (§3), in which Multiview mutates the result of access control at different levels to identify faulty configurations. This requires Multiview to know all the related components and their configurations to start the mutations. However, some configuration options have default values that are not explicitly declared in the configuration file, or the access-control rules are encoded in the source code. Thus Multiview currently cannot find such root-cause configurations with a black-box approach. In the future, Multiview can be combined with the white-box approaches [17–19, 63] to analyze the implicit configuration options.

Are the Multiview's generated directions of changes complete? Multiview's mutation policies to generate possible directions of changes require the software developers to provide the APIs for mutating each individual access-control rule, as well as the partial security order. Therefore, the changes identified by Multiview may not be complete if not all the mutations are specified. However, the API implementations are mainly a one-time effort for one application. Developers can focus on the common rules to provide high coverage of real-world access-deny issues for the sysadmins.

Can Multiview's change-impact analysis always faithfully reproduce access-control results? Multiview synthesizes the requests based on the subjects, objects and actions in the system and replay the requests with the reproduced virtual environment. However, in some *non-deterministic* scenarios, access control results may not relate to the attributes of synthesis input, but relate to environment settings such as time [46] or rate limiting [31]. Multiview currently can not faithfully replay such cases as these require a complete reproduction of the environment. To achieve deterministic replay in such cases, more runtime information needs to be recorded [24].

How do Multiview display the change-impact analysis results? Multiview performs the change impact analysis to find the impact on the access-control state for each direction of changes. The number of subjects and objects could be large in real-world organizations. Even though Multiview tries to reduce the number of synthesized requests, the results of change impact analysis could be lengthy and sysadmins may need to spend additional efforts to read the report. Currently, Multiview displays the results in a line-by-line format which includes the subject, action, object and the access-control result changes. We leave the tasks including visualizing and identifying important access-control changes as future work.

How do Multiview ensure no private information is leaked in the report? Multiview assumes that sysadmins are fully trusted. When Multiview presents the diagnosis report, it may contain the information related to all the subjects available in the system to present a comprehensive view. If there is sensitive information related to data privacy, certain rules should be applied to filter or hide such information in the report. Existing works on automatic private information filtering techniques [23, 67, 72] can be combined with Multiview to identify potentially private user information.

9 Related Work

Zero Trust security approach Zero trust is a set of security principles that remove the assumption of trust from users and resources within certain security perimeters [37, 51, 74]. It requires rigorous authentication and authorization process for every access, and the accesses are continuously monitored and logged to detect and audit potential security events. In zero trust architecture, access control policies are still the core in the authorization process with dynamic contexts. Multiview can help sysadmins to mutate and test the access control policy changes in the diagnosis process. This is particularly important in a Zero Trust architecture.

Access-deny issues characteristic study Xu et al. [79] first quantitatively studied the real-world practice of resolving access-deny issues based on the cases from online forums. The study demonstrates that sysadmins lack adequate information which prevents them from developing a precise understanding of the system and diagnosing the issue. However, no solution was proposed and evaluated in their study. Shen et al. [54] explored improving access-deny logging to help developers write better log messages for sysadmins. Our work tackles the problem by finding configurations related to the issue. Besides, we further perform delta analysis to find more directions to resolve the access-deny issue and evaluate their impact on the global access-control state.

Access control misconfiguration detection Many works have been proposed to detect the inconsistencies in the access-control policies with data mining [22, 26, 77, 83], testing [42, 43], and verification approaches [32, 36]. These works rely on the security property in a formal model or test oracles specified by the sysadmins, or learned policies from the probably correct systems, and then compare with the configuration settings to detect inconsistencies. Some works [22] can suggest fixes that learned from the other functional systems, but it introduces false positives as the security goals in each system may be different. In contrast, our work mutates the configuration settings based on the security order to provide more directions, and evaluate their impact on the access-control state to determine the one that best fits the security context.

Misconfiguration detection and diagnosis Previous research [17–19, 34, 49, 53, 63, 71, 73, 75, 78, 81, 83–85] has applied various types of techniques to the configuration error detection and diagnosis, including black-box and white-box approaches. In general, these works try to provide better tooling support for *developers* to diagnose configurations by finding the related configuration entries. They rely on symptoms like function failures or performance degradation to detect configuration errors. However, the access control configuration errors are still intended behavior by the source code. Besides, some rely on a modified test environment and run the instrumented programs again to debug the error. However, sysadmins do not read or have access to the source code as software developers, which makes them less likely to apply these tools for diagnosis. Besides finding the related configurations, our work took a step further to explore the possible directions to solve the access deny issue and evaluate the changes' impact to help sysadmins find a solution based on their security context.

Access control code vulnerabilities The access-control code in the software, if not properly implemented, may expose vulnerabilities that allow attackers to bypass the authentication or authorization process. Many prior works have been proposed to detect the bugs in access control [27, 44, 57, 65, 76] or enforce access-control properties at runtime with techniques like dataflow tracking [25, 48, 80]. Our work is complementary to their works as they focus on bugs in the code, while we focus on helping sysadmins find configuration settings that would solve the access-control issues that best fits in the security context.

10 Conclusion

This paper proposes a new diagnosis framework Multiview that can provide multiple directions of changes for resolving access-deny issues to help sysadmins reduce their blind spots in diagnosis. Multiview achieves this by categorizing access-control changes and uses delta generation to systematically mutate configurations and compares the mutations with security order. Multiview also examines the impact of changes in each possible direction to help sysadmins determine which one is suitable according to the security context. Our evaluation on real-world access-deny issues shows that Multiview can successfully help diagnose 89 out of 112 reproduced issues. We further conducted a user study with 20 participants on five real-world access-deny issues. The user study shows that Multiview can reduce the percentage of insecure fixes from 44.0% to 2.0% and reduced their diagnosis time by 62.0% on average.

References

[1] Apache user mailing list. https://httpd.apache.org/lists.html#http-users.

[2] Dba stack exchange. https://dba.stackexchange.com/.

[3] Mongodb forum. https://www.mongodb.com/community/forums/.

[4] Nginx user mailing list. https://mailman.nginx.org/archives/list/nginx@nginx.org/.

[5] PostgreSQL Privileges. https://www.postgresql.org/docs/current/ddl-priv.html.

[6] Postgresql user mailing list. https://www.postgresql.org/list/pgsql-general/.

[7] Proftpd user mailing list. https://sourceforge.net/p/proftp/mailman/proftp-user/.

[8] Server fault. https://serverfault.com.

[9] Squid user mailing list. http://lists.squid-cache.org/pipermail/squid-users/.

[10] Stack exchange. https://stackexchange.com/.

[11] Stack overflow. https://stackoverflow.com/.

[12] User study materials. https://drive.google.com/file/d/1JMBt-MC4kpJ8ix1L4YPlkgqRIQBJcrv0/view?usp=sharing.

[13] Apache configuration parser. https://github.com/etingof/apacheconfig, 2022.

[14] Nginx configuration parser. https://github.com/nginxinc/crossplane, 2022.

[15] Supplementary materials. https://github.com/ucsdopera/Multivew/blob/main/supplementary.pdf, 2023.

[16] askubuntu. Apache: access denied because search permissions are missing. https://askubuntu.com/questions/451922/apache-access-denied-because-search-permissions-are-missing, 2015.

[17] Attariyan, M., Chow, M., and Flinn, J. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)* (2012), pp. 307–320.

[18] Attariyan, M., and Flinn, J. Using causality to diagnose configuration bugs. In *USENIX Annual Technical Conference* (2008), pp. 281–286.

[19] Attariyan, M., and Flinn, J. Automating configuration troubleshooting with dynamic information flow analysis. In *OSDI* (2010), vol. 10, pp. 1–14.

[20] Barkley, J. Comparing simple role based access control models and access control lists. In *Proceedings of the second ACM workshop on Role-based access control* (1997), pp. 127–132.

[21] Barrett, R., Kandogan, E., Maglio, P. P., Haber, E. M., Takayama, L. A., and Prabaker, M. Field studies of computer system administrators: analysis of system management tools and practices. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work* (2004), pp. 388–395.

[22] Bauer, L., Garriss, S., and Reiter, M. K. Detecting and resolving policy misconfigurations in access-control systems. *ACM Transactions on Information and System Security (TISSEC) 14*, 1 (2011), 1–28.

[23] Castro, M., Costa, M., and Martin, J.-P. Better bug reporting with better privacy. *ACM SIGOPS Operating Systems Review 42*, 2 (2008), 319–328.

[24] Chen, Y., Zhang, S., Guo, Q., Li, L., Wu, R., and Chen, T. Deterministic replay: A survey. *ACM Computing Surveys (CSUR) 48*, 2 (2015), 1–47.

[25] Dalton, M., Kozyrakis, C., and Zeldovich, N. Nemesis: Preventing authentication & [and] access control vulnerabilities in web applications.

[26] Das, T., Bhagwan, R., and Naldurg, P. Baaz: A system for detecting access control misconfigurations. In *USENIX Security Symposium* (2010), pp. 161–176.

[27] Deepa, G., Thilagam, P. S., Praseed, A., and Pais, A. R. Detlogic: A black-box approach for detecting logic vulnerabilities in web applications. *Journal of Network and Computer Applications 109* (2018), 89–109.

[28] Detectives, S. Australian sports fan portal leaks 132GB of private data. https://www.safetydetectives.com/blog/bigfooty-leak-report/, 2020.

[29] Dietrich, C., Krombholz, K., Borgolte, K., and Fiebig, T. Investigating system operators' perspective on security misconfigurations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), pp. 1272–1289.

[30] DivvyCloud. 2020 cloud misconfigurations report. https://divvycloud.com/wp-content/uploads/2020/02/Cloud-Misconfiguration-Report-FINAL.pdf, 2020.

[31] Docs, N. Nginx rate limiting. https://docs.nginx.com/nginx/admin-guide/security-controls/controlling-access-proxied-http/, 2022.

[32] Fisler, K., Krishnamurthi, S., Meyerovich, L. A., and Tschantz, M. C. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering* (2005), pp. 196–205.

[33] Hatch., B. Linux file permission confusion pt 2. https://www.hackinglinuxexposed.com/articles/20030424.html, 2003.

[34] Huang, H., Shen, B., Zhong, L., and Zhou, Y. Protecting data integrity of web applications with database constraints inferred from application code. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (2023), pp. 632–645.

[35] IBM Security. Cost of a data breach report 2020. https://www.capita.com/sites/g/files/nginej146/files/2020-08/Ponemon-Global-Cost-of-Data-Breach-Study-2020.pdf, 2020.

[36] Jayaraman, K., Ganesh, V., Tripunitara, M., Rinard, M., and Chapin, S. Automatic error finding in access-control policies. In *Proceedings of the 18th ACM conference on Computer and communications security* (2011), pp. 163–174.

[37] Kerman, A., Borchert, O., Rose, S., and Tan, A. Implementing a zero trust architecture. *National Institute of Standards and Technology 2020* (2020), 17–17.

[38] Lampson, B. W. Protection. *ACM SIGOPS Operating Systems Review 8*, 1 (1974), 18–24.

[39] Lawler, R. Capital One data breach affected 100 million in the US. https://www.engadget.com/2019/07/29/capital-one-data-breach/, Jul. 2019.

[40] Linux. Overlay filesystem. https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html, 2021.

[41] Manual, A. O. Apache authentication and authorization. https://httpd.apache.org/docs/2.4/howto/auth.html, 2022.

[42] Martin, E., and Xie, T. Automated test generation for access control policies via change-impact analysis. In *Third International Workshop on Software Engineering for Secure Systems (SESS'07: ICSE Workshops 2007)* (2007), IEEE, pp. 5–5.

[43] Martin, E., and Xie, T. A fault model and mutation testing of access control policies. In *Proceedings of the 16th international conference on World Wide Web* (2007), pp. 667–676.

[44] Near, J. P., and Jackson, D. Finding security bugs in web applications using a catalog of access control patterns. In *Proceedings of the 38th International Conference on Software Engineering* (2016), pp. 947–958.

[45] Night lion security. Astoria company data breach research and analysis. https://www.nightlion.com/blog/2021/astoria-company-breach/, 2021.

[46] Oasis. Xacml v3.0 time extensions. https://docs.oasis-open.org/xacml/xacml-3.0-time-extensions/v1.0/csprd01/xacml-3.0-time-extensions-v1.0-csprd01.html, 2022.

[47] OWASP. Owasp top 10 vulnerabilities - 2021. https://owasp.org/Top10/, 2021.

[48] Parno, B., McCune, J. M., Wendlandt, D., Andersen, D. G., and Perrig, A. Clamp: Practical prevention of large-scale data leaks. In *2009 30th IEEE Symposium on Security and Privacy* (2009), IEEE, pp. 154–169.

[49] Rabkin, A., and Katz, R. Precomputing possible configuration error diagnoses. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* (2011), IEEE, pp. 193–202.

[50] Rodeh, O., Bacik, J., and Mason, C. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS) 9*, 3 (2013), 1–32.

[51] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. Zero trust architecture. Tech. rep., National Institute of Standards and Technology, 2020.

[52] Security World. 9 years to discover a data breach. https://www.secureworldexpo.com/industry-news/9-years-incident-to-breach-discovery-time, 2019.

[53] Shen, B. *Automatic Methods to Enhance Server Systems in Access Control Diagnosis*. University of California, San Diego, 2022.

[54] Shen, B., Shan, T., and Zhou, Y. Improving logging to reduce permission over-granting mistakes. In *USENIX Security Symposium* (2023).

[55] Shen, B., Wei, L., Xiang, C., Wu, Y., Shen, M., Zhou, Y., and Jin, X. Can systems explain permissions better? understanding users' misperceptions under smartphone runtime permission model. In *USENIX Security Symposium* (2021), pp. 751–768.

[56] Silicon Angle. Pharma giant pfizer exposes patient data on unsecured cloud storage. https://siliconangle.com/2020/10/20/pharma-giant-pfizer-exposes-patient-data-unsecured-cloud-storage/, 2020.

[57] Son, S., McKinley, K. S., and Shmatikov, V. Fix me up: Repairing access-control bugs in web applications. In *NDSS* (2013).

[58] sophos. The state of cloud security 2020. https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-the-state-of-cloud-security-2020-wp.pdf, 2020.

[59] Stack overflow. Error message "forbidden you don't have permission to access / on this server". https://stackoverflow.com/questions/10873295/error-message-forbidden-you-dont-have-permission-to-access-on-this-server, 2013.

[60] stack overflow. Apache - Permissions are missing on a component of the path. https://stackoverflow.com/questions/25190043/apache-permissions-are-missing-on-a-component-of-the-path, 2015.

[61] stack overflow. Apache 2.4.7 / Search permissions. https://stackoverflow.com/questions/33477056/apache-2-4-7-search-permissions, 2016.

[62] StackOverflow. Apache2 mod_wsgi access denied issue. https://serverfault.com/questions/357804/apache2-mod-wsgi-django-named-virtual-servers, Last accessed 2022.

[63] Su, Y.-Y., Attariyan, M., and Flinn, J. Autobash: improving configuration management with operating system causality analysis. *ACM SIGOPS Operating Systems Review 41*, 6 (2007), 237–250.

[64] Su, Y.-Y., and Flinn, J. Automatically generating predicates and solutions for configuration troubleshooting. In *USENIX Annual Technical Conference* (2009).

[65] Sun, F., Xu, L., and Su, Z. Static detection of access control vulnerabilities in web applications. In *USENIX Security Symposium* (2011), vol. 64.

[66] superuser. Permission denied because search permissions are missing on a component of the path, after chmod and chgrp. https://superuser.com/questions/882594/permission-denied-because-search-permissions-are-missing-on-a-component-of-the-p, 2016.

[67] Taneja, K., Grechanik, M., Ghani, R., and Xie, T. Testing software in age of data privacy: A balancing act. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (2011), pp. 201–211.

[68] Twitch. Twitch update on the security incident. https://blog.twitch.tv/en/2021/10/15/updates-on-the-twitch-security-incident/, 2021.

[69] Update, B. Biometrics company allegedly leaves unhashed fingerprint data of thousands exposed to internet. https://www.biometricupdate.com/202003/biometrics-company-leaves-unhashed-fingerprint-data-of-thousands-exposed-to-internet, 2020.

[70] Verizon. 2020 Data Breach Investigations Report. https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf, 2020.

[71] Wang, H. J., Platt, J. C., Chen, Y., Zhang, R., and Wang, Y.-M. Automatic misconfiguration troubleshooting with peerpressure. In *OSDI* (2004), vol. 4, pp. 245–257.

[72] Wang, R., Wang, X., and Li, Z. Panalyst: Privacy-aware remote error analysis on commodity software. In *USENIX Security Symposium* (2008), pp. 291–306.

[73] Wang, Y.-M., Verbowski, C., Dunagan, J., Chen, Y., Wang, H. J., Yuan, C., and Zhang, Z. Strider: A blackbox, state-based approach to change and configuration management and support. *Science of Computer Programming 53*, 2 (2004), 143–164.

[74] Ward, R., and Beyer, B. Beyondcorp: a new approach to enterprise security.; login: 39 (6), 6-11, 2014.

[75] Whitaker, A., Cox, R. S., Gribble, S. D., et al. Configuration debugging as search: Finding the needle in the haystack. In *OSDI* (2004), vol. 4, pp. 6–6.

[76] Xiang, C. *Detecting Access Control Misconfigurations with Change Validation*. University of California, San Diego, 2021.

[77] Xiang, C., Wu, Y., Shen, B., Shen, M., Huang, H., Xu, T., Zhou, Y., Moore, C., Jin, X., and Sheng, T. Towards continuous access control validation and forensics. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019), pp. 113–129.

[78] Xu, T., Jin, X., Huang, P., Zhou, Y., Lu, S., Jin, L., and Pasupathy, S. Early detection of configuration errors to reduce failure damage. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (2016), pp. 619–634.

[79] Xu, T., Naing, H. M., Lu, L., and Zhou, Y. How do system administrators resolve access-denied issues in the real world? In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017), ACM, pp. 348–361.

[80] Yip, A., Wang, X., Zeldovich, N., and Kaashoek, M. F. Improving application security with data flow assertions. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), pp. 291–304.

[81] Yuan, D., Xie, Y., Panigrahy, R., Yang, J., Verbowski, C., and Kumar, A. Context-based online configuration-error detection. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference* (2011), USENIX Association, pp. 28–28.

[82] ZDNet. Database leaks data on most of Ecuador's citizens, including 6.7 million children. https://www.zdnet.com/article/database-leaks-data-on-most-of-ecuadors-citizens-including-6-7-million-children/, 2019.

[83] Zhang, J., Renganarayana, L., Zhang, X., Ge, N., Bala, V., Xu, T., and Zhou, Y. Encore: Exploiting system environment and correlation information for misconfiguration detection. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems* (2014), pp. 687–700.

[84] Zhang, S., and Ernst, M. D. Automated diagnosis of software configuration errors. In *2013 35th International Conference on Software Engineering (ICSE)* (2013), IEEE, pp. 312–321.

[85] Zhang, S., and Ernst, M. D. Which configuration option should i change? In *Proceedings of the 36th International Conference on Software Engineering* (2014), pp. 152–163.

A Diagnosis Results

Detailed descriptions for Multiview's diagnosis results for cases in Table 3.

Case 1 (FS)

Related configurations

Apache process user lacks execute permission on "/var/www/html/www1/pages"

Possible directions

1. Add execute permission on group so Apache process user has execute permission.
2. Make apache process user to be owner of "/var/www/html/www1/pages"

Case 2 (FS)

Related configurations

Apache process user lacks execute permission on "/var/www/html/greg/"

Apache process user lacks read permission on "/var/www/html/greg/index.html"

Possible directions

Multiview has 3 mutations to access "/var/www/html/greg/"

1. Grant Apache execute permission as other.
2. Change directory's group to be Apache.
3. Change directory's owner to be Apache.

Multiview has 3 mutations to access "/var/www/html/greg/index.html"

1. Grant Apache read permission as other.
2. Change directory's group to be Apache.
3. Change directory's owner to be Apache.

Combine both directories' mutations to 9 directions.

10. Add user to the group of both directories "/var/www/html/greg/index.html" and "/var/www/html/greg/".

Case 3 (Apache)

Related configurations

Request denied by the directive
“Require method GET POST”.

Possible directions

1. Add DELETE method in the allowed HTTP methods.
2. Restrict the DELETE method to only be allowed to the requested data.

Case 4 (Apache)

Related configurations

Request denied by the directives in the config directory block “/var/www/sales/stats/”.

Possible directions

1. Add denied user to group admin.
2. Allow the denied user’s role ‘sales_manager‘ in the configuration
3. Allow the denied user’s role ‘sales‘ in the configuration
4. Allow the denied user ‘Alice‘ in the configuration

Case 5 (Nginx)

Related configurations

Request denied by the directives the config directory block “/var/www/html”.

Possible directions

1. Allow the denied user’s IP in the configuration block.

Case 6 (Vsftpd)

Related configurations

Request denied by the configuration entry “deny_file”.

Possible directions

1. Remove the option that matches with the denied object in the “deny_file”.

Case 7 (Proftpd)

Related configurations

Request denied by rules `“<Limit LOGIN>”` in the configuration block `<VirtualHost>`.

Possible directions

1. Add `<AllowUser>` for the denied user in the Limit section.
2. Add `<AllowGroup>` for current group of the denied user in the Limit section.
3. Add the denied user to the current allowed group `ftpuser`.

Case 8 (MySQL)

Related configurations

The user requires references privilege on Table B.

Possible directions

1. Grant the user with references privilege on Table B.
2. Grant the user with the same role of the owner of Table B to grant a set of privileges on Table B including references.

Case 9 (PostgreSQL)

Related configurations

The user requires usage privilege on the schema,
and execute privilege on the function.

Possible directions

1. Grant the user with USAGE privilege on the schema and grant the user with EXECUTE privilege on the function
2. Grant the user with the same role of the schema’s owner so the user can have a set of privileges to access the schema and the function.

Case 10 (MongoDB)

Related configurations

The user requires administrative privilege to read the storage stats on the table.

Possible directions

1. Create a role with `dbStats` privilege and grant the user with the newly created role.
2. Grant the `dbAdmin` role to the denied user.

Case 11 (Squid)

Related configurations

The user’s IP is inside the IP subnet of the block list.

Possible directions

1. Allow the IP with `http_access allow denied_IP;` before the rule to block the whole subnet.