UNIVERSITY OF CALIFORNIA SAN DIEGO

Automatic Methods to Enhance Server Systems in Access Control Diagnosis

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Bingyu Shen

Committee in charge:

Professor Yuanyuan Zhou, Chair
Professor Patrick Pannuto
Professor George Porter
Professor Geoffrey M. Voelker
Professor Xinyu Zhang

2022

The dissertation of Bingyu Shen is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2022

DEDICATION

To my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Professor Yuanyuan (YY) Zhou for her constant support, guidance and inspiration throughout my PhD journey. Her unique observations and insights have always inspired me on my research projects. I have learned so many invaluable lessons from her about research, life, and character, all through examples. She taught me to think big and target on real-world problems with large impact; She taught me how to think critically and articulate research problem to different audience; She taught me to pursue perfection and even tiny details will make a difference; She taught me to be tough and face the real-world challenges. Her encouragement and support helped me out from the hard times. I am fortunate to work with and learn from her over the past five years.

I would like thank Professors Patrick Pannuto, Geoffrey M. Voelker, George Porter, and Xinyu Zhang for being on my doctoral committee. Their discussions and constructive feedback greatly improve this dissertation.

I would like to thank Professor Geoffrey M. Voelker. Geoff is always encouraging and gave me a lot of insightful feedback on my research. I still remember that Geoff knew my name on the first day when I got to the department, which is extremely heart-warming for an international student. The SysNet hiking and end-of-year celebration organized by Geoff are among the best memories during my PhD journey.

I would like to thank other professors and staff in the SysNet group. Professor Stefan Savage provided very helpful feedback during the experiment design of Multiview project. Professor George Porter, Yiying Zhang and Alex Snoeren gave me insightful feedbacks during the Syslunch presentation and 294 seminars. I am grateful to Cindy More, the system administrator of SysNet group. Cincy helped maintain the SysNet servers and enlightened me on administrators' perspectives on access control.

I am also thankful to my collaborators from the Whova company: Tianwei Sheng, Xinxin Jin. They provided valuable anonymized data and insightful comments in two projects.

I would like to thank my labmates in the Opera group. They are both my collaborators and friends. Chengcheng Xiang and Yudong Wu helped me a lot on research and life when I was a junior student. Tianyi Shan collaborated two projects with me and burned the midnight oils to fight the paper deadlines. I am deeply in debt to her. Haochen Huang and Li Zhong discussed the research projects together with me and helped me in several experiment evaluations. I also enjoyed the casual research discussions with Eric Mugnier. I also want to thank my friends, Vector, Mingyao, Lixiang, Shelby, Nishant, Zesen, Enze, Yizhou, and many others in the SysNet group. Thank you for all the constructive feedback and interesting discussions.

Last, I am immensely grateful to my family. My parents, Qingbin Shen and Ping Liu, and my brother, Bingshen Shen, encouraged to pursue my dream and supported me with their unconditional love. My wife, Ye Xiao, went through every bit of the PhD journey with me and made my life meaningful and colorful. Words cannot express how grateful I am for their love and support. I dedicate this dissertation to them.

Chapter 2 and 4, in full, is a reprint of the material as it appears in the 32nd USENIX Security Symposium, 2023. Bingyu Shen, Tianyi Shan, Yuanyuan Zhou. “Improving Logging to Reduce Permission Over-Granting Mistakes”. The dissertation author was the primary investigator and author of this paper.

Chapter 3 and 4, in part, is a reprint of the material under submission. Bingyu Shen, Tianyi Shan (co-first authors), Yuanyuan Zhou. “Multiview: Finding Blind Spots in Access-Deny Issues Diagnosis”. The dissertation author was the primary investigator and author of this paper.

VITA

2017	B. E. in Computer Science and Technology, Shanghai Jiao Tong University
2021	M. S. in Computer Science, University of California San Diego
2022	Ph. D. in Computer Science, University of California San Diego

PUBLICATIONS

Bingyu Shen, Tianyi Shan (co-first authors), Yuanyuan Zhou. "Multiview: Finding Blind Spots in Access-Deny Issues Diagnosis". Under submission.

Bingyu Shen, Tianyi Shan, Yuanyuan Zhou. "Improving Logging to Reduce Permission Over-Granting Mistakes". To appear in the 32nd USENIX Security Symposium (Security'23), August 2023.

Bingyu Shen, Lili Wei, Chengcheng Xiang, Yudong Wu, Mingyao Shen, Yuanyuan Zhou, Xinxin Jin. "Can Systems Explain Permissions Better? Understanding Users' Misperceptions under Smartphone Runtime Permission Model". The 30th USENIX Security Symposium (Security'21), August 2021.

Chengcheng Xiang, Yudong Wu, Bingyu Shen, Mingyao Shen, Haochen Huang, Tianyin Xu, Yuanyuan Zhou, Cindy Moore, Xinxin Jin, Tianwei Sheng. "Towards Continuous Access Control Validation and Forensics", The 26th ACM Conference on Computer and Communications Security (CCS'19), November 2019.

ABSTRACT OF THE DISSERTATION

Automatic Methods to Enhance Server Systems in Access Control Diagnosis

by

Bingyu Shen

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California San Diego, 2022

Professor Yuanyuan Zhou, Chair

Access control configurations are gatekeepers to block unwelcome access to sensitive data. Unfortunately, system administrators (sysadmins) sometimes over-grant permissions when they resolve unintended access-deny issues reported by legitimate users. The mistakes in the access control configurations can result in severe consequences, such as data breaches and system compromises. To make things worse, the access control misconfigurations may stay silent until the security incident happens.

This dissertation explores two approaches to help sysadmins diagnose the access-deny issues and reduce the permission over-granting mistakes. The first approach takes the problem of insufficient access-control logging in server applications. We designed an automated tool,

SecLog, to automatically add missing access-deny log messages, and also enhance existing ones with relevant information to guide sysadmins to diagnose the access-deny issues. The second approach tackles the problem of blind spots in knowledge and system settings for sysadmins in solving access-deny issues. We propose a system, Multiview, to automatically mutate the system configurations to explore possible directions and let 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 directions to allow the request, as well as the impact of each direction on the access-control state of the entire system to assist sysadmins during diagnosis.

Chapter 1

Introduction

1.1 Motivation

Access control mechanisms have been adopted in various computer systems to prevent unauthorized principals from performing certain actions on protected data [55]. Access control policies in computer systems have been customized to achieve the security goals of an organization. As modern software systems and access control policies are becoming more complex to accommodate the dynamic needs, companies rely on sysadmins to manage access control policies, as well as fix issues in case of problems.

Unfortunately, access control has been reported as "profoundly broken" due to the prevalence of misconfigurations [56, 79, 108]. Many recent security incidents have shown that even subtle errors in the access control configurations can result in severe data breaches and system compromises. For example, a misconfigured server of a billion-dollar consulting services company exposed customers' private information, certificates, 40,000 passwords, and other sensitive data to the public [22]. Similarly, 154 million U.S. voters' information was leaked simply because of one access control mistake - the database was not password protected at all [83]. Table 1.1 shows more recent real-world newsworthy security incidents caused by on access control

Table 1.1: Recent publicly-reported security incidents caused by access control misconfigurations.

Time	Incident	Organization
2018.3	42,000 patients information leaked [36]	Huntington hospital
2018.4	63,551 patients records breached [35]	Middletown medical
2019.1	24 million financial records leaked [40]	Ascension
2019.7	140 thousand SSNs and DoB leaked [57]	Capital One
2019.9	20 million citizen records exposed [115]	Elasticsearch
2020.03	76,000 fingerprints leaked [95]	Antheus Tecnologia
2020.05	132GB of sensitive data published [39]	Big Footy

misconfigurations.

While access control misconfigurations can be introduced during initial settings, many are caused when sysadmins change configurations to resolve issues [97]. One of the most prominent examples is *access-deny issues*, i.e., legitimate users are denied access to data that they are supposed to have. As these users need to gain access to the data to perform their jobs, the time pressure can easily lead sysadmins to make mistakes when changing access control settings to grant these users access. As these users need to gain access to the data to perform their jobs, sysadmins need to diagnose the issue and change access control settings to grant these users access. However, server software often does not provide informative logging to guide sysadmins to understand the reported problems, which may result in incorrect fixes. Mistakes in granting permissions often lead to permission over-granting security vulnerabilities, which can go unnoticed for months until catastrophic data breaches happen.

One of the key problem is that the current server systems are not well designed for the servers users, i.e., system administrators, in access control management, especially in server issues diagnosis. The problem is manifested in two ways.

First, server software often does not provide informative logging to guide sysadmins to understand the reported problems, which may result in incorrect fixes. Mistakes in granting permissions often lead to security vulnerabilities, which can go unnoticed for months until catastrophic data breaches happen. Figure 1.1 shows a real-world access-deny issue in Vsftpd.

Problem: client could not download a file
Log message: `get xxx.mp4` `550 Permission Denied.`
Attempts
1. Change file permission to 777 [harmful][useless] 2. Change file owner to the logined user. [userless] 3. Remove the `userlist_file` config entry to allow all users login. [harmful][useless]
Correct and safe fix
Access denied because the server's configuration will deny access to file matched with the pattern specified in `deny_file`. Change the pattern in this entry will allow download this file.

Figure 1.1: Real world example of an access-deny issue for Vsftpd.

The user was denied from accessing files on the FTP server and the server log only shows "Permission denied". The sysadmin attempted several fixes during the problem-solving process, some of which opened up access to more users and introduced security holes that can be easily exploited by malicious users. According to a recent study on sysadmin practices based on sysadmin online forums and mailing lists, 38.1% of the access-deny issues were resolved by insecure modifications that over-granted accesses, introducing security vulnerabilities [108].

Second, sysadmins lack accessible tools in the troubleshooting and fixing process. 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 misunderstand security context and adopt solutions that may unknowingly introduce security risks. Other times, sysadmins may neglect the security consequences and not thoroughly examine the modifications once the denied request is allowed. Moreover, the process of resolving access-deny issues correctly 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 [41]. These blind spots lead them to resort to risky solutions.

1.2 Contribution

This dissertation presents two approaches to help sysadmins diagnose the access-deny issues and reduce the permission over-granting mistakes. The first approach takes the problem of insufficient access-control logging in server applications. We designed an automated tool, SecLog, to automatically add missing access-deny log messages, and also enhance existing ones with relevant information to guide sysadmins to diagnose the access-deny issues. The second approach tackles the problem of blind spots in knowledge and system settings for sysadmins in solving access-deny issues. We propose a system, Multiview, to automatically mutate the system configurations to explore possible directions and let 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 directions to allow the request, as well as the impact of each direction on the access-control state of the entire system to assist sysadmins during diagnosis.

Thesis Statement: *By automatically enhancing the server systems' diagnosability with logging improvements and diagnosis tooling support, we can reduce the number of sysadmins' mistakes in solving access-deny issues.*

1.2.1 Improving Logging of Server Systems with SECLOG

Chapter 2 presents SECLOG. SECLOG aims to help developers improve log messages in order to help sysadmins correctly understand and fix access-deny issues without over-granting permissions.

In Chapter 2, I first conducted an observation study to understand the current practices of

access-deny logging in the server software. The study shows that many access-control program locations do not have any log messages; and a large percentage of existing log messages lack useful information to guide sysadmins to correctly understand and fix the issues. On top of the observations, SECLOG uses static analysis to automatically help developersfind missing access-deny log locations and identify relevant information at the log location.

Chapter 2 evaluates SECLOG with ten widely deployed server applications. Overall, SECLOG identified 336 new log statements for access-deny cases, and also enhanced 551 existing access-deny log messages with diagnostic information. We have reported 114 log statements to the developers of these applications, and so far 70 have been accepted into their main branches. We also conducted a user study with sysadmins (n=32) on six real-world access-deny issues. SECLOG can reduce the number of insecure fixes from 27 to 1, and also improve the diagnosis time by 64.2% on average.

1.2.2 Finding Blind Spots in Access-Deny Issues Diagnosis with Multiview

Chapter 3 presents Multiview. Multiview aims to provide multiple directions to resolve access-deny issues in order to help sysadmins reduce blind spots in diagnosis. Multiview automatically mutates the system configurations to explore possible directions and lets 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 directions to allow the request, as well as the impact of each direction on the access-control state of the entire system.

Chapter 3 evaluates SECLOG with a user study of 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 64 real-world failure cases from four different systems and server applications, and it can successfully diagnose 45 of them. Multiview accurately identifies the failure-causing configurations and provides possible directions to each access-deny issue within one minute.

Chapter 2

Improving Logging of Server Systems with SECLOG

2.1 Introduction

2.1.1 Motivation

Our main goal is not just about fixing the unintended access-deny issue for that particular user: we are more concerned about insecure "fixes" that over-grant the permissions. After all, there can be many ways to "fix" such issues for that particular user. For example, many incorrect fixes simply disable the entire protection domain and make the object accessible to everyone [108]. While such a "fix" resolves the issue for that particular user, it opens a big door for others, including malicious users, to get into the system and access sensitive data. Even worse, it can remain undetected until exploited by malicious users, causing a major real-world security incident.

Therefore, to reduce the number of such security incidents, it is important to help sysadmins to resolve access-deny issues safely. While it is impossible to eliminate human mistakes (after all, to err is human), one natural question is whether system builders (i.e., server software

developers) can improve our software to help sysadmins diagnose and fix such issues.

Fortunately, the answer is 'Yes'. When a sysadmin tries to fix an access-deny issue, the first step is to understand (1)*Who gets denied? (i.e., "subject")*; (2)*What operations get denied? (i.e., "operation)*; and (3) *What data is denied access? (i.e., "object")*. These questions may sound trivial but on investigation are complicated. The "subject" may not be the user who reported this issue, because today's systems involve many components such as application servers, databases, etc. Similarly, the "object" and "operation" may not be that straightforward either. For example, for users who cannot access a web page, their problem may be related to that page's file permission, or the clients' IPs.

A common practice to understand access-deny issues is to check log messages generated by the related software systems. Unfortunately, logging practice in today's systems is ad-hoc and lacks critical details [108]. The incompleteness of access-deny related log messages is reflected in two ways. **First**, some access control checks deny permissions silently without any log at all to guide sysadmins. Figure 2.1a shows such an example in the Cherokee web server [2]. If there is no log message at all, sysadmins have to go on a wild goose chase. Sometimes sysadmins resort to the online internet forums, but the inadequate log messages often bring incorrect suggested fixes which can result in major security vulnerabilities [109]. **Second**, even when log messages exist, the messages are often too generic, and lack critical information related to subject, object and operation, all of which are important for sysadmins to understand the root cause and fix the settings safely. Figure 2.1b shows a real-world example of denied access with only generic information in log messages. Although there are three different possible reasons, i.e., 1) HTTP request method, 2) IP, 3) environment variables, that can result in access denial, it has only one log message: `AUTHZ_DENIED`. While this may make the source code easy to read, such generic error code provides very little value for sysadmins to troubleshoot and fix the access-deny issue.

As such, it is critical to improve logging quality to provide accurate information for sysadmins to fix access deny issues correctly. Unfortunately, to the best of our knowledge very

```
/* Cherokee-1.2.103: cherokee/config_reader.c*/
dir = cherokee_opendir (path->buf);
if (dir == NULL)
+    LOG_CRITICAL ("Could not open directory '%s', check the
server user and file permissions.", path->buf);
    return ret_error;
```

Silently ignoring the denied access in reading config file

(a) Cherokee

```
/* httpd-2.4.46: modules/aaa/mod_authz_core.c */
status ip_check_authorization(request *r, ...) {
  …; return AUTHZ_DENIED;/*check request's IP w/ config.*/
}
status method_check_authorization(request *r, ...) {
  …; return AUTHZ_DENIED;/*check request method w/ config.*/
}
status env_check_authorization(request *r, ...) {
  …; return AUTHZ_DENIED;/*check environment vars. w/ config.*/
}
int authorize_user(request* r, ...) {
 ...
 if (auth_result == AUTHZ_DENIED) {
  ap_log_rerror(APLOG_ERR, r, APLOGNO(01631)
   "%s: authorization failure for \"%s\": ",r->user, r->uri);
}
```

Invoke above functions based on the request's config. All w/ the same error msg.

(b) Apache httpd

Figure 2.1: Two log message examples in two widely-used web servers, Cherokee and Apache httpd.

little work has been done on this direction. The closest work was efforts by Yuan et al. on improving log messages for the purpose of software bug diagnosis [114, 111, 120]. While this work is inspiring, logging for developers is different from logging for sysadmins. To fix access-deny issues correctly, log messages for sysadmins need to have different critical information and thus different techniques need to be applied to identify such information.

2.1.2 Our contributions

This work makes one of the first attempts (to the best of our knowledge) to help developers improve log messages for sysadmins to diagnose and fix access-deny issues correctly and safely. First, we studied five large open-source server programs including Apache httpd, PostgreSQL, Vsftpd, NFS-Ganesha, and Proftpd to understand the problems in access-deny logging practice. Our study confirms our motivation that: (i) many access control check locations do not have any log statements for access-deny, leaving no information for sysadmins; (ii) Even for cases with access-deny log statements, a large percentage of them lack relevant details such as subject, object, and action of the denied access to guide sysadmins understand and fix such issues (§2.2).

Second, we built a tool called SECLOG which employs static analysis to automatically detect access control check locations in various server applications. SECLOG identifies missing access-deny log locations and relevant information at log location. We evaluated SECLOG with ten widely deployed server programs. Overall, SECLOG inserts 336 access-deny log statements in these software, and also automatically enhances 551 existing log messages with relevant information. We have reported 114 inserted or enhanced log messages to the developers of these popular programs, so far **70 have been confirmed and accepted into their main branches**.

Furthermore, to evaluate the effectiveness of our inserted or enhanced log messages, we conducted a **user study** with sysadmins (n=32). Our user study results show that the log messages enhanced by SECLOG can cut access-deny issue diagnosis time by 64.2%. More importantly, with the original log messages, the sysadmins introduced 27 over-granting security issues in their

fixes, whereas only one such issue was introduced with the enhanced log messages.

2.2 Understanding Access-deny logging in Real-world Applications

Before we dive into a solution, we first aim to understand the current status in access-deny logging practices and possible opportunities for inserting such log messages with useful information for sysadmins. We first conduct an empirical study with five widely-deployed server software systems – Apache httpd (webserver), PostgreSQL (database), Vsftpd and Proftpd (FTP server), and NFS-Ganesha (NFS), written in C/C++ languages. We focus on server software systems because their access control settings are usually critical, and over-granting permissions due to mistakes can lead to major security incidents, as we have depicted in Section 3.1.

2.2.1 Methodology

To collect the access-deny log messages in software systems, we first identify the access-deny program locations through two kinds of program signatures, error code and access control check functions. First, we collect error codes related to access denial in each application from their official documentation and manuals. For example, the error codes that would lead to a 403 HTTP status include `HTTP_FORBIDDEN`, `AUTHZ_DENID` in Apache httpd. From the location where the error code is assigned and the propagation of the error code, we record the log message that is specific to the denied access.

Second, we search by the code pattern that performs permission check to find the access-deny program locations. All the studied applications use access control check (ACC) functions to perform access check to system resources (e.g., file and port), or perform application-specific access check (authorization/authentication modules, DB privileges, ACL lists). Figure 2.1b shows

the example authorization functions in Apache httpd. Then we identify access-deny log messages by finding the call sites of ACC functions and propagation of check results. We remove the duplicates if one point can both be identified by error code and ACC function.

Our study first focuses on access-deny log points that are enabled by default during production such as `Fatal`, `Error`, `Warn` verbosity levels. To be more comprehensive, we also look into log points at more verbose levels that are usually not enabled during production, such as `Debug` or `Trace` levels, to further understand and expose issues in current practices of access-deny logging. (Finding 1)

Second, we examine whether the log message at the access-deny log point contains relevant information needed to understand the reason for denial. More specifically, we classify the information into the subject, access action and denied object at each point. (Finding 2)

Third, to understand more about how those access log messages were added, and whether they were added as afterthoughts, we also looked at the change history of the logging at each access-deny program point in the public version control repository. To identify patches relevant to the access-deny logging, we conduct the analysis for each software using its public version control repository. For each log statement, we collect the corresponding file name and line numbers. Then, we retrieve all the history commits related to a specific log statement using the `git log` utility. We regard the first commit to add each log statement as the one that adds the log statements; the following commits are the ones that revise the log statements. Last, we remove duplicate commits in case multiple log statements were added in one patch. Only commits make changes to access-deny log points are selected. (Finding 3)

We have two inspectors meet and discuss the standard for the collected information before studying each software. Each inspector independently investigates and records the information of interest, including the root cause configuration, the relevant variables in source code, and the request's characteristics such as subject, object, and action at the access-deny program location. Then the two inspectors compared the results and discussed them with each other in iterations.

All results reached a consensus in the end.

Threats to Validity. Like all characterization studies, there is an inherent risk that our study may be specific to the applications studied and thereby may not be generalized to other software. While we cannot establish representativeness categorically, we have taken care to select diverse server software systems that are widely used in different areas. Note that our study only focuses on server software where access control is critically important; the findings may not apply to client or mobile applications. These studied programs also share commonalities that all are open-source and written in C/C++, which is common in popular server systems.

Another potential source of bias is that we may miss access-deny points that use ad-hoc access control checks that can not be identified by the error codes or ACC functions. In practice, we found that the well-maintained server programs check for access via ACC functions or differentiate the logging via error codes. Ad-hoc checks which were treated as general errors by the application developers, were also excluded by our study, such as format checks of authorization headers.

2.2.2 Findings

Finding 1: *Under the **default** verbosity mode, 14.1% to 64.7% of cases have no log messages at all when an access is denied (Table 2.1).* Such an overlook in software practices makes it quite challenging for sysadmins to troubleshoot an unintended access-deny issue and fix the access control setting correctly without over-granting permissions and introducing security vulnerabilities.

Compared to other logging, access-deny logging is more important as it guides sysadmins in the right direction to fix access control settings correctly. Unlike other misconfigurations that usually have visible erroneous symptoms during software execution, *access-control misconfigurations that over-granting permissions can go silent for months without being noticed until being exploited by malicious users, causing catastrophic security incidents*.

Table 2.1: The number of access-deny program points that have log messages at default verbosity level and only at debug verbosity level.

Application	At default level		Only at debug level	
Apache httpd	115	64.6%	10	5.6%
PostgreSQL	374	77.8%	0	0.0%
Vsftpd	61	85.9%	0	0.0%
NFS-ganesha	24	35.3%	38	55.9%
Proftpd	145	56.8%	25	9.8%

Table 2.2: The subject, action and object in each application.

App.	Subject	Action	Object
Apache	Server user, process ID	HTTP methods; file perm. (`rwx`).	Webpages,files
Postgre.	DB user	DB privileges	Tables, schemas, etc.
Vsftpd	FTP user; process ID	FTP commands; file perm. (`rwx`).	Files, directories.
NFS.	Client IP, user name	NFS commands (e.g., `mnt`)	Files, dirs, NFS exports
Proftpd	FTP user; process ID	FTP commands; file perm. (`rwx`).	Files, dirs.

Apache httpd, NFS-ganesha and Proftpd contain some log information only at debug-level logging. While this is better than no log messages at all, the information may not be very helpful because (1) sysadmins may not know that there exist some log messages at debugging or tracing verbosity level (since most sysadmins do not read source code [108]); (2) it would require sysadmins to restart the software with debugging level enabled and reproduce the denied request to get the relevant information from log messages. As such, for the following characteristics studies, we only focus on the default level log messages.

Finding 2: *Most existing access-deny log messages lack relevant information to guide sysadmins, with 37.5-100% of the log messages missing subject information, 0.0-64.7% missing action or access type information and 0.0-75.4% missing information about the accessed objects. However, the majority (70.8% - 100%) of relevant information related to the denied access is available within the same function of the corresponding access check operation (Table 2.3).*

We classify useful information for sysadmins into three categories, subject, action and object, based on the specific scenario of the access [55]. There are mainly two scenarios of access-

Table 2.3: The number and percentage of log messages that do *not* contain subject, action or object information; "Same func" is the number of access-deny program points that have relevant information within the same function of the check operation.

App.	Total	Subject		Action		Object		Same func.	
Apache	115	102	88.7%	25	21.7%	18	15.7%	112	97.4%
Postgre.	374	275	73.5%	242	64.7%	107	28.6%	357	95.5%
Vsftpd	61	59	96.7%	20	32.7%	46	75.4%	61	100%
NFS.	24	9	37.5%	0	0.0%	6	25.0%	17	70.8%
Proftpd	145	145	100%	0	0.0%	0	0.0%	145	100%

control checks which involve different subjects, actions or objects. The first scenario checks access to *system resources* including files, network sockets, etc. In this scenario, the subject is the role of server process in the OS; the action is the operation to be processed on the resource (e.g., read or write the file, bind network port); the object is the system resource. The other scenario checks the access to application-specific resources (e.g., authorization/authentication modules, DB privileges, ACL lists). The subject is the role in the application to perform the operation (e.g., the DB user or the authenticated user in the web server); the action is the required access to the resource (e.g., `SELECT/REFERENCES` privileges in PostgreSQL); the object is the application-specific resource. More details about the subject, object and action in each application are shown in Table 2.2.

The information related to subject, action and object is fundamental for sysadmins to understand the access-deny issue and come up with correct solutions. Missing any of the information may cause additional difficulties for sysadmins. **First**, subject information is necessary to know who is being denied. However, among all software, only a few percentages of log messages include subject information. One might assume the subject information is the legitimate user who reported the access-deny issue. But for many software programs, it is more complicated. For example, in most database servers, DB user is usually the database account used by application server to perform the operation requested by the end user, which is different from the end user. For file accesses, subject information is usually the server process, but it would be more complicated

Table 2.4: The number of patches that added or revised access-deny logging statements in source code. Vsftpd does not have a public version control repository and was excluded from this study.

Application	Add logs	Revise logs
Apache httpd	40	511
PostgreSQL	165	1728
NFS-ganesha	7	344
Proftpd	33	1209

if `setgid` or `setuid` are used. **Second**, the action (access type) information is critical for sysadmin to grant only necessary privileges. For example, PostgreSQL utilizes logging templates in may places, but the template only includes the denied object, such as "`permission denied for table %s`". They do not have information related to the user role or required privileges while PostgreSQL has 12 distinct levels of privileges [8]. **Third**, the object information is useful to know what data access is denied. Sysadmins usually need to inspect the ACLs associated with the object to decide how to change the privileges.

Fortunately, the majority of the relevant information is available within the same function of the access check operation as shown in Table 2.3. This means that when developers write the access-deny log statements in the source code, they could have included the relevant information without much complexity.

Finding 3: *Access-deny logging practice is ad-hoc and many existing log messages are added as afterthoughts (Table 2.4).* We find that many efforts from the developers are needed to add and revise the access-deny logging statements. The patches are made to (1) add new log message in previously silently denied program location or (2) revise existing log message to include additional information (e.g., file name). Figure 2.2 shows two patch cases in Apache httpd. In Figure 2.2(a), a new message was added to log the reason why the request is denied. In Figure 2.2(b), an existing access-deny log message was revised to log the operation and the file path. The added information was crucial to help sysadmins resolve the denied access, but would require huge efforts from experts to find and understand the denied locations, which calls for the

```
if (apr_filepath_merge(&r->filename, conf, ...)!= 0) {
+ ap_log_rerror(APLOG_MARK, APLOG_ERR, rv, r,
+   "URI in %s maps to invalid filename", r->the_request);
  return HTTP_FORBIDDEN;}
```

(a) A commit to add new log message

```
rv = apr_file_open(ds->pathname, MODE_WRITE_SEEKABLE);
if (rv != APR_SUCCESS) {
  return dav_new_error(rv,
-   "An error occurred while opening a resource.");
+   "An error occurred while opening a resource for writing:
+             %s.", ds->pathname);
```

(b) A commit to add info. in the log message

Figure 2.2: Patches from Apache that add or revise log messages.

need for an automated tool to assist the developers.

Table 2.5: The number of access control check (ACC) function and the number of access-check program points that are identified by the ACC functions.

Application	# ACC function	# Call sites
Apache HTTPD	18	141 (79.2%)
PostgreSQL	34	319 (66.3%)
Vsftpd	17	52 (73.2%)
NFS-ganesha	15	40 (65.6%)
Proftpd	32	256 (100%)

Finding 4: *A small (15-34) number of ACC functions are used to perform access checks in many (40-319) program locations for different kinds of purposes (Table 2.5).*

The studied applications commonly use ACC functions to perform checking in modules related to network, files, or authentication and authorization. These ACC functions are called to perform checking in the majority (65.6%-100%) of access check program points. Only in a small number of cases, the application may perform ad-hoc checks for certain operations without directly calling an ACC function. For example, Vsftpd directly checks whether the anonymous user is allowed against server configurations in the initialization phase. ACC functions are called in many different locations of the source code. On average, one ACC function has 6.1 call sites. For example, `pg_class_aclcheck` in PostgreSQL which is used to check whether the user has

certain privileges to access a table, has 42 different call sites in its source code. Without tooling support, large number of access check locations make it challenging for developers to ensure good logging at every single point.

Table 2.6: The number of access-check program points identified by the ACC function where there are log statements at the function's call site, inside the function, or no log statements.

Application	Has log at caller	Has log in ACC func.	w/o logs
Apache HTTPD	100 (58.1%)	15 (8.7%)	72 (41.8%)
PostgreSQL	211 (66.1%)	100 (31.3%)	108 (33.9%)
Vsftpd	35 (67.3%)	8 (15.4%)	10 (19.2%)
NFS-ganesha	14 (35.0%)	1 (2.5%)	26 (65.0%)
Proftpd	146 (57.0%)	8 (3.1%)	110 (43.0%)

Finding 5: *Comparing all logging locations in the source code, logging at access control check function's call site is more common than other locations in source code (Table 2.6).*

We zoom in to understand where the log messages are placed in today's practice. Developers usually place log messages at the ACC function's call sites instead of inside the ACC function itself. The main advantage of placing messages at a call site is that it can log more relevant information. Only in a few cases, the developers choose to add log messages only inside the access control check function. Note that in PostgreSQL, all the ACC functions have log messages inside the function, but there are also logged messages at the function's call site. If SECLOG follows the common practice to place the logging statements at the call site of ACC functions, SECLOG can reuse most of the existing access-deny logging statements and avoid some performance overhead.

2.2.3 Implication

In summary, even in mature software systems, many access-deny program points do not have log messages at the default level. A large percentage of the existing log messages lack relevant information regarding the denied access, even though the information is available in the

same function. Despite developers have spent huge efforts to add and revise the log messages, the current logging practice is far from satisfactory due to either negligence or lack of tooling support. Therefore, a natural question is: can we automatically improve access-deny logging from the existing source code and assist developers in creating better log messages?

2.3 Challenges and Design Choices

Motivated by our real-world applications observation, we aim to design and build a tool, called SECLOG, to help software developers to enhance access-deny log messages and also insert missing log messages to provide guidance for sysadmins to fix access-deny issues without over-granting permissions. To achieve this goal and improve the quality of access-control logging in general, there are three fundamental challenges.

1. **How to log:** How to maintain high-quality log statements in a continuous software development process.
2. **Where to log:** How to identify the access-control check (ACC) program locations (including missing ones) in large server software and where to place the log statements.
3. **What to log:** How to find the critical information to add into log statements that can guide sysadmins in solving the access-deny issues safely and correctly.

We discuss the challenges and design choices in detail, as well as **compare them with the alternative approaches.**

2.3.1 How to Log

Software code is constantly evolving to meet the dynamic needs, which also requires developers to spend efforts to maintain high-quality log statements. However, current access-deny logging practice is ad-hoc as it relies on developers to manually identify log locations and add

useful logging information at each location (c.f. §2.2). One simpler approach is to provide the developers with a logging library to guide/enforce developers writing better log messages. While it is possible to manually make an improvement at each access-check program location, this process is tedious and error-prone because (1) there are hundreds of ACC program locations in server software (e.g., more than 300 access-check points in large software like PostgreSQL) and (2) for each location, developers need to examine the entire call chain to search for log statements or the lack of such statements, as well as to identify critical information related to the access denial.

We design SECLOG to automate the process with the help of static analysis for the following considerations. **First**, the source code contains rich information related to the denied access, which could be extracted with static analysis (Finding 2 in §2.2). **Second**, static analysis can go through all the access-control check locations to identify inadequate or missing log messages, which would be challenging for developers to manually go over. **Third**, the automated process could be integrated to the CI pipelines to help with code review, which enforces consistent logging practice for even new contributors. Though SECLOG still requires some annotations from developers, this as a one-time small effort would benefit the code development process in the long run.

2.3.2 Where to Log

Where to Find ACC Locations. It is challenging to identify ACC locations across different server applications because each application performs various access control checks on the requests. Manually identifying each location in a large server application requires expertise and huge efforts.

SECLOG addresses this problem by leveraging the common adoption of ACC functions to reduce the input from developers while achieving a high coverage of access check locations. To understand the percentage of access-check locations covered by ACC functions, we performed a

measurement on the five applications in §2.2. We find that the majority (65.6%-100%) of access check program points can be identified by ACC functions. Using ACC functions allows SECLOG to identify up to more than 300 access control check locations in large software like PostgreSQL with no more than 34 ACC functions (Table 2.5).

Although ACC functions still have to be annotated by developers, the effort is much lower than manually inspecting hundreds of access-check points. The developers can easily identify these ACC functions in modules related to system resources access (e.g., file or network), or application-specific checks in access control related source files. As shown in Evaluation §2.5.1, even novice developers could annotate the ACC functions with high coverage in a short amount of time.

Where to Place Log Statements. After SECLOG finds all ACC program locations, the second challenge is where the existing or new (to be added by SECLOG) access-control log statements should be placed. One approach would be placing the log statements *inside* the ACC function instead of after ACC function's call sites. However, the call site contains more relevant information for sysadmins to resolve the access-deny issues. According to our study, 68-100% relevant information related to the denied access is available within the same function of the call site of ACC functions (Table 2.3). Besides, the majority of access-deny logging statements are at ACC functions' call sites instead of inside a check function (More details in Table 2.6). Placing the logging statements at call sites, SECLOG can improve existing logging statements without intrusive logic modifications to source code.

Another alternative approach is to *always* add after the check at the call site, regardless of whether the access result is denied or not. This approach would have two disadvantages. **First**, too much logging will introduce higher performance overhead and downgrade the server throughput [96]. **Second**, too much logging can overwhelm sysadmins to find the related log message when they need to fix issues like an unintended access-deny case reported by a legitimate user.

In sum, SECLOG decides to place the log statements after the call sites of ACC functions only when the access is denied. To achieve this, we design a semantic pattern matching algorithm to identify the access-deny paths in Section 2.4.1.

2.3.3 What to Log

The third challenge would be what should be included in the access-deny log messages to help sysadmins fix the access-deny issues correctly. A naive solution would be to include all the parameters of the ACC function as relevant information since they are used to perform access control checks. However, it is imprecise and misses the opportunity to collect specific denied reasons. **First**, not all function parameters are involved in the decision of whether the access would be denied. Logging unrelated information may confuse the sysadmins. **Second**, the parameters may be a large object represented as a struct with many fields (e.g., the request object in Apache httpd contains more than 100 fields). Only the fields that cause the access denied are relevant and should be logged. **Third**, some accesses may be denied in an ACC function for different reasons. Figure 2.11 in Appendix 2.5.3 shows that the `cherokee_mkdir_p_perm` function could be denied for two reasons: (1) denied to *create* a new directory with the specified mode, or (2) denied to *open* an existing directory with certain permissions. To guide sysadmins for an unintended access-deny case, it is more useful to give the specific information in the log message.

To find precise information related to access control from source code, SECLOG identifies the information relevant to the access from two sources: (1) inside the ACC function and (2) at the ACC function's call site. To ensure the collected information is only related to the denied access, SECLOG performs backward slicing from the access-deny return value inside the ACC function and data dependency analysis to find the related variables. We discuss more about the analysis process in Section 2.4.2.

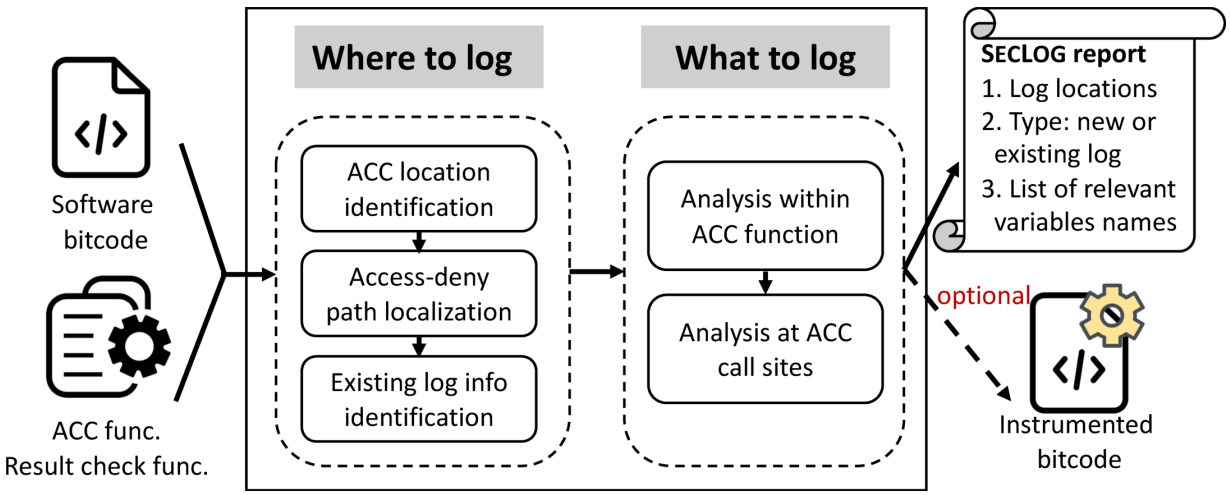

Figure 2.3: Workflow of SECLOG

2.4 Design and Implementation

SECLOG targets large server software where over-granting permissions and security incidents can cause major incidents. Since most server software was written in C/C++, SECLOG is built on top of LLVM compiler frameworks that can handle C/C++ programs. SECLOG's static analysis algorithms can be easily extended to handle Java applications.

The overview workflow of SECLOG is shown in Figure 2.3. SECLOG operates on the LLVM bitcode of the application and processes the IR representation of the source code. Besides, SECLOG takes two types of annotations from the developers: (1) A list of developer-selected ACC functions in the target software, and (2) the corresponding result check function paired with the ACC function that is used to determine whether the access is denied based on the return value of ACC functions. For example, if the return value from an ACC function is not `NO_ERROR`, then it is access-denied. One example ACC function and corresponding result check function is shown in Figure 2.4.

SECLOG further performs analysis to identify (1) where to log (§2.4.1) and (2) what to log (§2.4.2). SECLOG produces a report including the list of access-control log locations, the type of log at each location (existing or new), and the list of relevant variable names at each location.

1. Access control check function

```
// check permission on opening a file; return the check result as status struct
status_t check_open_permission(struct obj_handle* obj,
                struct flag_t* flags, bool create);
```

2. Access control check function call sites

```
status_t fsal_open2(struct obj_handle** obj, struct flag_t* flags,
          enum mode cmode, ...){
...  // init
status = check_open_permission(*obj, flags, cmode>=FSAL_EXCLUSIVE);
if (! (status.major == NO_ERROR)) goto out;
```

3. Result check function

```
bool result_check_int_1(status_t status) {
  return status.major != NO_ERROR;
}
```

Return true when the access control check function's return value represents *access denied*

Figure 2.4: An access control check function example from NFS-Ganesha.

Developers can utilize SECLOG's report to generate the final human-readable log messages. SECLOG can also be configured to instrument the bitcode by inserting the pairs of variable names and values at each logging location with provided logging functions.

2.4.1 Identify Where to Log

SECLOG identifies the logging locations with the help of ACC functions as discussed in §2.3.2. To achieve this goal, SECLOG will first perform static analysis to identify the access-control check locations. Then SECLOG places the log statements only when the access is denied at the check location, by identifying the access-deny paths at the ACC function call site. We discuss the detailed analysis algorithms as follows.

Identify Access-control Check Locations. SECLOG identifies the log location by first extracting the call graphs from the bitcode. With the help of call graph, SECLOG can comprehensively find all the call sites of ACC functions where the access is checked. Most ACC functions in the same software follow a similar convention to indicate access denial in the same software. This makes it easier for developers to provide result check functions.

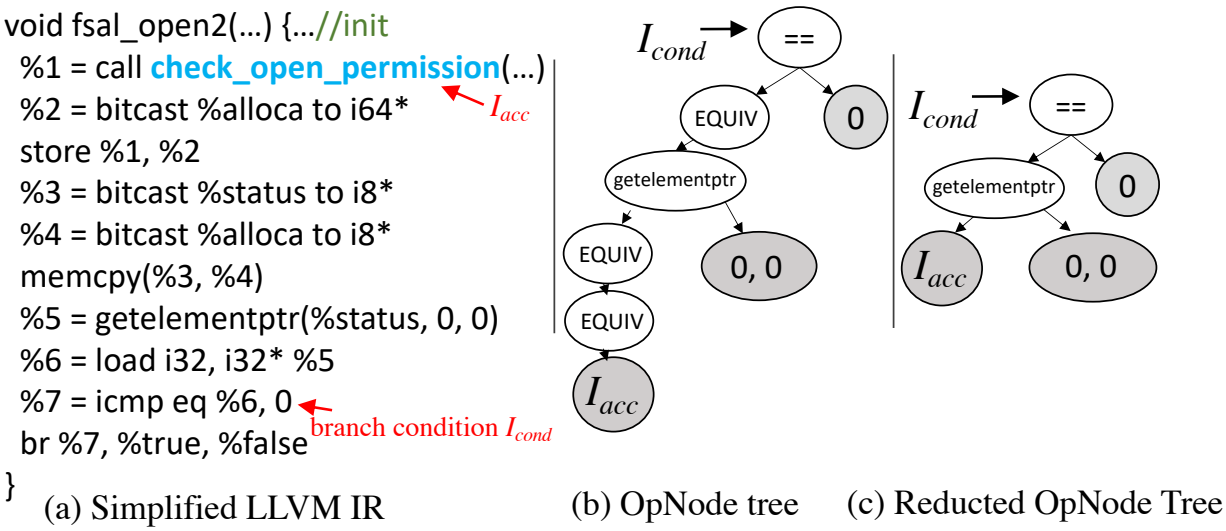

Figure 2.5: Result checking operations in LLVM IR for call site's code snippet.

We find that function pointers are used as special ACC functions in certain C-based applications, such as Apache httpd. The application uses function pointers in a struct as a function template to call different ACC functions at any call site. However, the use of function pointer makes the log messages at the function call sites less specific, since the different access control check functions may be called at the same site, but fails for different reasons, as shown in Figure 2.1b. To find the call sites via the function pointers, we require the developers to annotate the specific field in the struct that represents the function pointers to ACC functions. Then SECLOG identifies all possible functions that can be assigned to this type of pointer. This helps us extend the call graph between the call site of function pointer and the real ACC functions to conduct the remaining analysis.

Identify where to Place Access-Deny Logs. As discussed in §2.3.2, SECLOG decides to place the log statements only when the access is denied. This requires SECLOG to identify the existing access-deny path in the source code. If there is an access-deny path (i.e., the code snippets at the ACC function's call site to check whether the ACC function's return value represents access-deny), the snippets would be similar to the operations included in the result check function. Figure 2.5(a) shows the corresponding LLVM IR of the example function's call

site in Figure 2.4. The ACC function `check_open_permission` returns a struct of error status. To perform the check in LLVM IR code, the call site performs several operations to retrieve one field in the struct and check if that field represents the access is denied.

To capture the result checking operations and identify access-deny paths, as detailed in Algorithm 1, SECLOG performs a semantic pattern matching process by using the program slicing technique with data flow analysis starting from each access-deny program location. The algorithm will match the result check function with the denied branch.

Algorithm Details. The semantic pattern matching algorithm matches between the access-deny path at the call site and the result check function. By definition, the result check function takes the return value of the ACC function as an input, and returns true when the access is denied as shown in Figure 2.4. The algorithm perform the analysis to match the result check function with the denied branch in two steps.

First, for each call site, SECLOG identifies the call instruction I_{acc} (i.e., the instruction that invokes the ACC function) and the branch conditions impacted by I_{acc}. It performs intra-procedural static forward slicing that has data dependency on I_{acc}. S_{acc} is the set of all instructions in I_{acc}'s forward slice. SECLOG then computes the set of all related conditional branch instructions in S_{acc}: $\{I_{cond} | I_{cond} \in S_{acc} \cap \texttt{Type}(I_{cond})\texttt{=branch}\}$.

Second, SECLOG performs a semantic pattern matching between the result check function and the paths between I_{acc} and all branch instructions I_{cond} in S_{acc}. This step prunes out infeasible paths that do not match with the operations. Algorithm 1 `OpNodeTreeMatch()` lists the core steps.

- Construct the operation `OpNode` from instructions in S_{acc}. Each instruction in S_{acc} can be mapped to a `OpNode` that includes the instruction (i.e., operation), the operands, and the defined type of the LLVM IR in the SSA style.
- Construct `OpNode` tree backwards from I_{cond}. The root of the tree is the `OpNode` of I_{cond}, while the leaves of the tree can only be I_{acc} or constants types. The `OpNode` tree of the result check function R_{check} can be constructed similarly between the input parameter and R_{check}'s return

value.

- Reduce `OpNode` tree. The operations in defined `EQUIV` type like `load, store, cast` can be removed, so that the two `OpNode` trees can be matched in the simplest form.
- Match `OpNode` tree. The final operation matching between the two trees uses the `MatchTree()` procedure in Algorithm 1. The matching algorithm returns None if no match, otherwise it outputs the true or false branch for the branch instruction as the access-deny branch. Note that SECLOG also performs simple logic transformations, to ensure that the branch condition like `rv == -1` can match with specified result check functions like `bool check(rv){return rv < 0;}`. This can be further improved by a formal SAT solvers [37].

Identify Existing Logs or Add New Ones. To avoid redundant logging, SECLOG finds whether the access-deny log statements already exist inside the ACC function. SECLOG performs static backward slicing from the access-deny return values to search for a logging statement.

SECLOG first searches for logging statements specific to the denied access at the ACC function call site. We count them as specific for the denied access only when the basic block where the log statement lies in the denied branch (identified from our previous step), but does not post-dominate the allowed branch.

An access-deny error may be logged in the upper caller function by propagating the error code. SECLOG records the return value and performs data flow analysis in the call chain to check if the return value is propagated back to its caller in the call chain. SECLOG recursively looks at upper caller function at up to three levels for efficiency considerations.

If no access-deny branches were found, SECLOG can optionally instrument the bitcode by inserting a *checker* by calling the result check function. To avoid affecting the original semantics, the checker was inserted immediately after the basic block where the ACC function was called. Inside the branch, only a log statement is added.

Algorithm 1: OpNodeTreeMatch

```
struct {
    instr: the original instruction I;
    operands: operands of I;
    type: Defined type of I, (e.g. EQUIV type includes store,load,cast instructions)
} OpNode;
Function OpNodeTreeMatch(I_cond, I_acc, R_check, S_acc):
    Input: I_cond: branch condition instruction;
    I_acc: call instruction of access check function;
    R_check: OpNode tree root of the result check function;
    S_acc: the instructions tained by I_acc;
    ins_map ← {}                                   ▷ mapping from instruction to OpNode;
    for I_i ∈ S_acc do
        ▷ convert I_i to OpNode
        ins_map[I_i] ← OpNode(I_i, Operands(I_i), Type(I_i))
    root ← ins_map[I_cond]                         ▷ root is the tree from I_cond
    root ← ReduceTree(root, ins_map)
    return MatchTree(root, R_check)
Function ReduceTree(root, ins_map):
    if root.type = EQUIV then
        root.operands[0] =  ▷ EQUIV only has 1 operand
          ReduceTree(ins_map[root.operands[0]])
    else
        for operand ∈ root.operands do
            ReduceTree(operand, ins_map)
    return root
Function MatchTree(root, R_check):
    r_match ← []; retval ← 0
    if root.type = R_check.type then
        ▷ Recursively match operands of root and R_check
        r_match ← matchAllOperands(root, R_check)
        if None ∉ r_match then
            if Value(root) = ¬Value(R_check) then
                retval = 1 − retval
            return retval
    return None
```

2.4.2 Identify What to Log

Based on the characteristics of ACC function, SECLOG identifies the information relevant to the access from two sources: inside the ACC function and at the ACC function's call site.

Inside the ACC function. For each access-deny return value in the ACC function, SECLOG performs backward slicing to find a slice from the access-deny return value backward to the beginning of the function. Then SECLOG extracts all live-in variables [16] (i.e., function parameters, global variables, constants), on which the access-deny return value has a data or control dependency. The variables collected along each deny slice are added separately to represent various reasons for denials (e.g., Figure 2.11 in §2.5.3). Those variables are also used in analysis at the call site. If the access check function is a library call (i.e., the source code can not be analyzed), the above analysis is skipped and all function parameters are treated as relevant.

At the ACC function's call sites. ACC function's call sites have specific context information that is useful to guide sysadmins. From the function parameters, SECLOG traces back to the global variables of the configuration settings, and adds them into the corresponding access-deny log statements (e.g., Figure 2.12 in §2.5.3). More specifically, SECLOG collects live-in variables starting from the relevant variables identified inside the ACC function in the previous step. These variables are data-dependent on the relevant variables in ACC function. To collect such information, SECLOG performs a revised backward slicing which only performs the static backward slicing on the data flow graph.

2.5 Evaluation

We evaluate SECLOG to answer the following questions: First (§2.5.1), how much effort is it for the developers to use SECLOG? Second (§2.5.2), how effective is SECLOG in improving existing access-control log statements and identifying missing log locations? Third (§3.6.2), how effective are SECLOG-enhanced logging statements in helping the administrators in solving

Table 2.7: Evaluated applications. Applications in gray are covered in our study. We added five other software to evaluate the generality. *ACC func*: the number of ACC function. *Res. func*: the number of types of the result check functions. *Anno. Time*: average time in minutes that were spent on annotating the ACC and result check functions. *Analysis time* is total running time to analyze the application.

Applications	Category	LOC	# ACC func.	# Res. func.	Anno. Time	Analysis Time (mins)
Apache httpd[1]	Web server	199K	18	4	-	45
PostgreSQL[7]	Database	886K	34	2	-	354
vsftpd[11]	FTP	16K	17	3	-	0.2
NFS-ganshea[5]	NFS	165K	9	3	-	14.0
Proftpd[9]	FTP	228K	32	2	-	10.1
Postfix[6]	Mailserver	124K	13	3	72	32.1
HAProxy[3]	Proxy server	167K	5	3	55	4.7
Cherokee[2]	Web server	62K	12	3	32	0.2
Redis[10]	Key-value	134K	5	3	35	0.6
mSQL[4]	Database	35K	3	1	20	0.3

real-world access-deny issues? Lastly (§2.5.5), how efficient is SECLOG's analysis and what is the runtime overhead to the server software? The experiments were conducted on a machine with Intel Core i7-7700 CPU (3.6GHz, 8 cores), 16 GB RAM, 1 TB HDD with Ubuntu 16.04.

2.5.1 Efforts in Adopting SECLOG

Number of annotations. We annotated the ACC functions and result check functions in ten open-source server software including web servers, databases, FTP, proxy servers, etc. Table 2.7 lists the evaluated software. The number of ACC functions in each software is small (3-34), which varies with the application type and code base size. Each ACC function pairs with one result check function, but many ACC functions that have the same format to represent the return value can share the same result check function. In fact, each evaluated software only has 1-4 types of result check functions.

Manual efforts in annotation. Two authors measure the time that they each spent on annotating the ACC functions and the result check functions for five new applications that are not

Table 2.8: Evaluation results for number of annotated ACC functions and result check functions. The percentage is calculated based on the ACC functions used in the evaluation. Time includes the time to annotate the ACC functions and write result check functions.

Applications	Participant 1		Participant 2	
	# ACC func.(%)	Time (mins)	# ACC func.(%)	Time (mins)
PostgreSQL	34 (100.0%)	43	34 (100.0%)	32
vsftpd	16 (94.1%)	28	15 (88.24%)	14
NFS-ganshea	8 (88.9%)	42	6 (66.7%)	12
Postfix	12 (92.3%)	22	12 (92.3%)	23
HAProxy	4 (80.0%)	28	5 (100.0%)	16
Cherokee	11 (91.7%)	35	10 (83.33%)	28
redis	3 (60.0%)	32	5 (100.0%)	19
mSQL	3 (100.0%)	11	3 (100.0%)	12

studied in Section 2.2. As shown in Table 2.7, the annotation time is usually less than 1 hour since the number of ACC functions is small. We did not measure the time spent on the five applications covered in Section 2.2 because we studied them extensively to understand their practices.

To further measure the manual efforts in annotation for SECLOG, we recruited two graduate students in computer science to annotate the ACC functions and result check functions. Before the start of the study, we used two applications (Apache and Proftpd) to explain the definitions of ACC functions and result check functions. We also presented the examples in these two applications to further explain how we annotate the ACC functions and result check functions.

The participants are provided with the specific version of the software's source code and the access control related documentation. Different from the software maintainers, the participants have limited knowledge about the applications. We allowed them to read the documentations of software to understand the specific details of access control before the study. The time is not recorded when they read the documentations. The participants were instructed to find all the ACC functions and annotate the result check function for each ACC function. We recorded the total time they spent on each application.

Results: The annotation results are shown in Table 2.8.From the study, we find that even for developers have less experience and knowledge of the software, they are able to find the ACC functions and annotate the result check functions with high coverage (>90%) in a short amount of time (avg. 25 minutes). The manual efforts required for software maintainers could be even smaller.

Efforts in using SECLOG's outputs to write log messages. When the authors submit patches based on SECLOG's outputs, the main efforts are spent on understanding the meanings of the variables in the context of each application and correlating them to write log messages; the other efforts are mainly development efforts including finding appropriate logging functions and testing.

We conduct a user study to measure the developers' efforts in using SECLOG's outputs to write the log statements. This user study is approved with IRB exempt status via the IRB amendment. We recruited participants from the open-source software's slack, discussion forum, and mailing lists. Before the study, the participants will be provided with the four applications' source code with specific versions that SECLOG evaluated on, including Apache, PostgreSQL, vsftpd and mSQL. We randomly select 2-3 log locations identified by SECLOG in each application, which makes a total of 10 log locations. For each log location, we provide three kinds of information based onSECLOG's outputs: (1) Log location: the specific line number of the source code file which needs to write the log messages; (2) ACC function: the function name through which SECLOG identifies the logging opportunity; and (3) Variables: the list of variable names identified by SECLOG. For the variables that are fields in complex struct, we will present them in a `struct_name.field_name` format.

The participants are asked to write the log statements with the following requirements: (1) The log statements should utilize the information in the variables. (2) The log statements should be human-readable and provide useful information for the system administrators. We do not require the participants to write log statements with the specific log functions in each

application; instead, the participants only need to focus on the content of log messages and write the log message with `printf()` function. We record the time that each participant spent on each task. After the participant finish all the problems, we conduct a semi-structured interview to ask how they utilize the provided information to write the log statements and whether the information is easy to use.

Table 2.9: The time (in minutes) spent to write log statement based on the outputs.

Log ID	Participant 1	Participant 2	Participant 3
Apache-1	3.1	7.9	5.2
Apache-2	2.4	3.5	4.5
PostgreSQL-1	4.1	4.4	3.4
PostgreSQL-2	1.0	2.8	2.8
PostgreSQL-3	1.8	3.2	2.5
vsftpd-1	4.7	6.5	5.2
vsftpd-2	3.8	7.9	4.6
mSQL-1	6.9	9.7	5.1
mSQL-2	2.1	4.2	3.5
mSQL-3	2.3	5.1	6.2
Average	3.2	5.5	4.3

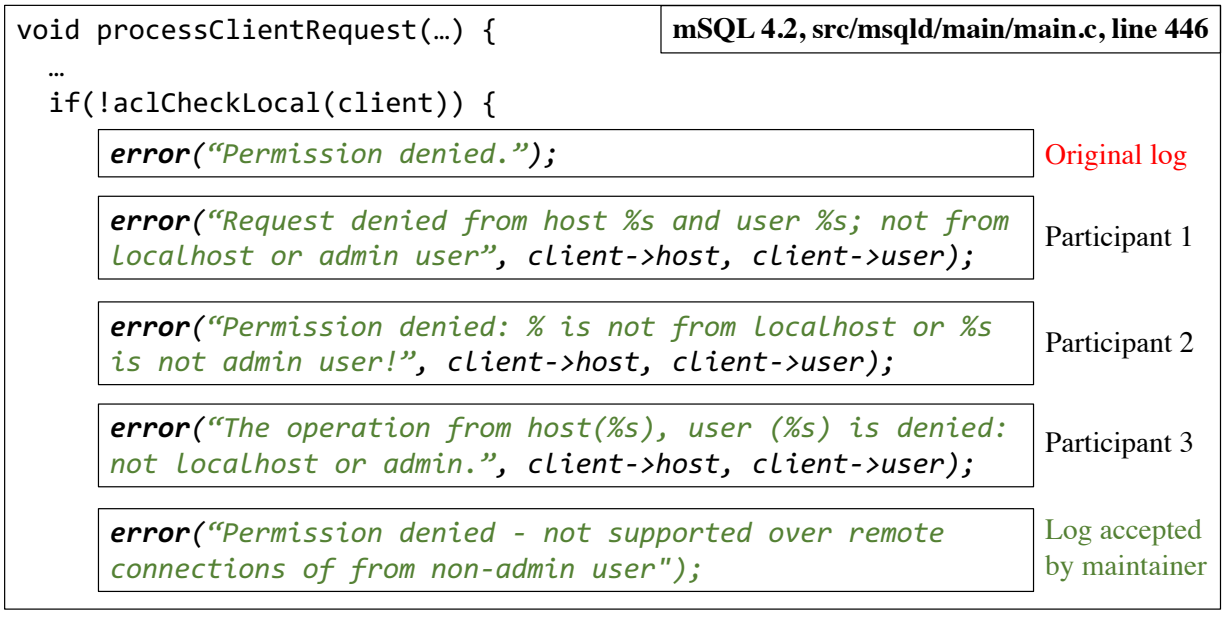

Figure 2.6: The log statements written by the participants in mSQL-1 with SECLOG's outputs.

Results. In total, we recruited three developers to participate in our study. We recorded

the time and log statements written by the participants at each program location. On average, the time required to write one log statement is 3.34 minutes, with minimum of 1.0 minute and maximum of 9.7 minute. The detailed results are shown in Table 2.9. In general, the time varies based on the log location in each application, which may relate to the number of variables and function size at the log location. The main efforts spent by the developers are to understand the meaning of the variables at each log location based on the source code and relevant document pages. Then the developers need to correlate the information to write the final log message.

We find that all the log statements written by the participants can cover the same set of information as the final patches accepted by the developers, even though the wording is not exactly the same. This may because they are all guided by the SECLOG-identified variables. For example, Figure 2.6 shows one complex log statement from the designed task mSQL-1, with the log statements written by the three participants and the one accepted by the software maintainers. The efforts spent by maintainers who are familiar with the code logic could be even less.

2.5.2 Improve Access-Deny Logging

To evaluate the effectiveness of SECLOG in improving existing access-control log statements and identifying missing logging locations, we applied SECLOG on ten server applications and verified the enhanced log messages and new ones.

Number of improved log messages. Table 2.10 shows the number of existing log messages enhanced by SECLOG and *new* log statements added by SECLOG, for the ten evaluated software applications, and the diagnostic information (represented as numbers of variables) added into the logging statement. SECLOG automatically inserted 6 to 121 log statements into these applications at those access-deny program locations that have no log in the call path. For the existing log messages, SECLOG has added about 1.67-8.75 additional variables per log message. For newly added log messages, there are about 3.0 to 14.28 variables per log message on average.

Table 2.10: Evaluation results for improvement of existing and new log statements. This table shows (1) The number of existing log messages enhanced and the average number of additional variables added per existing log statement. (2) The number of newly added log messages and the average number of relevant variables per new log statement.

Appl.	Existing logs		New logs	
	#	**# New vars.**	**#**	**# Vars.**
Apache.	93	5.62	44	7.05
Postgre.	203	2.61	121	4.16
vsftpd	39	1.67	9	2.44
NFS.	8	3.00	20	5.90
Proftpd	145	4.03	111	4.13
Postfix	8	8.75	36	14.28
HAProxy	3	3.33	10	6.40
Cherokee	11	2.18	23	3.35
redis	11	1.82	6	3.00
mSQL	19	2.00	-	-

The number of SECLOG-enhanced log statements varies with the number of access-control checks in the software due to (1) software size and (2) the design of ACC code. The larger software like Apache, PostgreSQL and Proftpd contains a higher number of checks. However, in software like redis/NFS-ganshea, the number of ACCs is small despite the software being large in terms of LOC. This may be because the design of ACC code varies based on the nature of each software. The web servers and database servers have many access control privileges and authentication/authorization modes, therefore the access control checks are conducted in many program locations. In contrast, the file system server or the memory cache server (redis/NFS-ganshea) mainly rely on file system protection and have fewer access control checks.

Cases confirmed and accepted by developer. We took a step further to report the SECLOG-enhanced log messages to the open-source projects. Note that some applications (e.g., vsftpd) do not have a formal forum for us to report issues, so we did not report them. We have submitted 114 enhanced log messages as patches to the developers of those applications that have formal forums. When we contact the developers, we prioritized submitting patches to those who responded to our initial patches. So far **70** of 114 have been accepted and merged in their *main*

Table 2.11: The coverage of the variables in existing log statements. The variables in the existing logs are manually picked by developers.

Appl.	# exi. logs	#(%) covering all vars. in the log	#(%) missing any vars.
Apache	93	73 (79.5%)	20 (21.5%)
Postgre.	203	203 (100%)	0 (0.0%)
vsftpd	39	39 (100%)	0 (0.0%)
NFS.	8	5 (62.5%)	3 (37.5%)
Proftpd	145	144 (99.9%)	1 (0.1%)
Postfix	8	7 (87.5%)	1(12.5%)
HAProxy	3	2 (66.7%)	1(33.3%)
Cherokee	11	11 (100%)	0 (0.0%)
redis	11	11 (100%)	0 (0.0%)
mSQL	19	19 (100%)	0 (0.0%)

branches.

Coverage of variables in existing log messages. To understand the accuracy of SECLOG-identified variables, we use the variables in existing log messages as the ground truth to compare with SECLOG's outputs. These variables are manually picked by developers. We consider one log message covered by SECLOG only when **all** the existing variables in the log message are in the SECLOG-identified variables; otherwise it is regarded as missed. As shown in Table 2.11, SECLOG achieves a high coverage in all applications.

We further looked into the log messages with variables *missed* by SECLOG. There are mainly two reasons. **First**, the variables logged by developers are not used in this access-control check. For example, in the 3 cases missed in NFS, developers choose to log the client's IP address when they perform the authentication (i.e., checking credentials). Similarly, for 7 cases in Apache, developers log the requested URL when the code checks access for requested files. **Second**, the variables logged by developers *mismatched* SECLOG's outputs. For example, for 9 cases in Apache, developers logged the filenames which are missed by SECLOG. This is because the file operation only uses the file descriptor to perform the check. SECLOG could not find the filename previously associated with the descriptor in the upper call chain.

Table 2.12: Evaluation results of improvement on the existing log statements.

Appl.	# exi. logs	Subject		Action		Object	
		before	after	before	after	before	after
Apache.	93	3 (3.2%)	88 (94.6%)	88 (94.6%)	93 (100%)	79 (84.9%)	93 (100%)
Postgre.	203	0 (0.0%)	203 (100%)	0 (0.0%)	152 (74.9%)	203 (100%)	203 (100%)
vsftpd	39	3 (7.7%)	26 (66.7%)	5 (18.2%)	39 (100%)	9 (23.1%)	33 (84.6%)
NFS.	8	0 (0.0%)	8 (100%)	1 (12.5%)	8 (100.0%)	0 (0.0%)	2 (25.0%)
Proftpd	145	0 (0.0%)	90 (62.1%)	145 (100%)	145 (100%)	145 (100%)	145 (100%)
Postfix	8	0 (0.0%)	7 (87.5%)	6 (75.0%)	8 (100%)	1 (12.5%)	4 (50.0%)
HAProxy	3	2 (66.7%)	3 (100%)	3 (100%)	3 (100%)	2 (66.7%)	2 (66.7%)
Cherokee	11	1 (9.1%)	9 (81.8%)	10 (90.9%)	10 (90.9%)	10 (90.9%)	10 (90.9%)
redis	11	0 (0.0%)	11 (100%)	9 (81.8%)	11 (100%)	5 (45.5%)	6 (54.5%)
mSQL	19	0 (0.0%)	19 (100%)	0 (0.0%)	11 (57.9%)	0 (0.0%)	11 (57.9%)

Improvements on subject/action/object information. To understand the improvement of log messages, we further classify the existing variables and newly added variables into subject, action and object based on the characteristics of each application. The results are shown in Table 2.12 and Table 2.13 for existing and newly added log statements respectively. For the existing log messages in Table 2.12, SECLOG can systematically improve the log message with variables related to subject, action and object. For subject, SECLOG can find process id for file accesses, and user info for the SQL queries in different applications; For action, SECLOG can find the access mode or privileges related to the access; For object, SECLOG can find the variables related to files, DB or table names.

However, SECLOG still can not identify all the information related to subject, action or object from the source code because the check may not involve all the attributes. For example, in 25.1% of the cases where SECLOG can not identify the action in PostgreSQL, the code simply checks whether the user is the owner of the object (e.g., table). Similarly, SECLOG can not identify the object in the authentication checks where the check only involves attributes related to the subject.

Helpfulness of SECLOG-identified variables. We conducted a survey to quantitatively evaluate the helpfulness of the information in SECLOG-identified variables in diagnosing access-

Table 2.13: The number and percentage of newly added log messages containing subject, action and object.

Appl.	# new logs	Subject	Action	Object
Apache	44	41 (93.2%)	44 (100%)	44 (100%)
Postgre.	121	121 (100%)	111 (91.7%)	121 (100%)
vsftpd	9	9 (100%)	9 (100%)	7 (77.8%)
NSF.	20	20 (100%)	20 (100%)	20 (100%)
Proftpd	111	102 (91.9%)	111 (100.0%)	111 (100.0%)
Postfix	36	36 (100%)	36 (100%)	31 (86.1%)
HAProxy	10	10 (100%)	10 (100%)	8 (80.0%)
Cherokee	23	18 (78.3%)	23 (100%)	20 (87.0%)
redis	6	6 (100%)	6 (100%)	6 (100%)
mSQL	-	-	-	-

Table 2.14: Average helpfulness rating of variables in a 5-point Likert scale, with 1 as not helpful and 5 as extremely helpful.

Appl.	SECLOG	Random
Apache	3.63	1.43
Postgre.	4.25	1.39
vsftpd	3.88	1.22

deny issues. We choose three commonly used applications, Apache, PostgreSQL and vsftpd, and in each application, we randomly selected 10 access-deny program points improved by SECLOG. For each program point, we carefully designed an access-denied scenario that would be denied access at the program point. Then we show the participants with the meaning of the SECLOG-identified variable and the actual value at the access-deny program point. To compare with the helpfulness of SECLOG-identified variables, we randomly selected two additional variables at the access-deny program point (i.e., within the call site of ACC function based on the LLVM IR code). For each scenario, participants are asked to rate the helpfulness of each variable on a 5-point Likert scale, with 1 as "not helpful" and 5 as "extremely helpful". To avoid overwhelming the respondents with too many questions, only four scenarios are randomly drawn for each respondent. Participants were not compensated for our survey.

Here we show one example survey question in PostgreSQL. The first three variables are

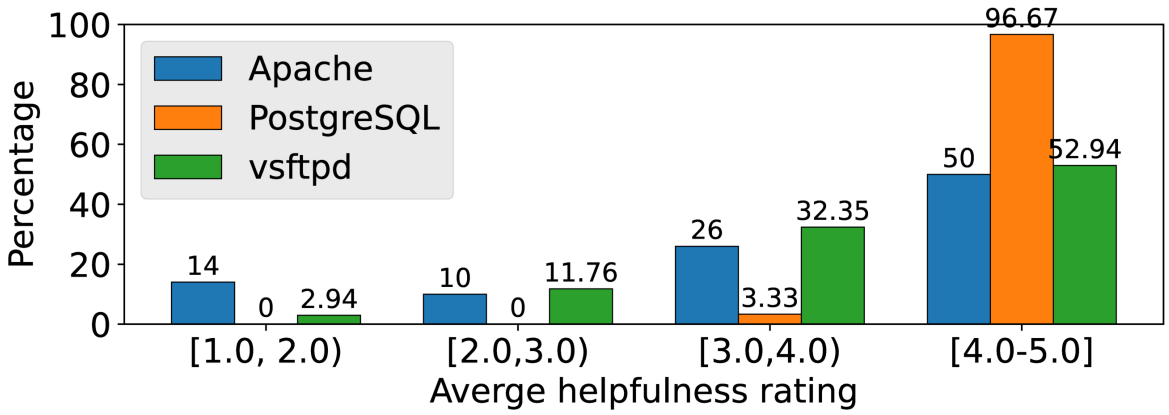

Figure 2.7: The distribution of SECLOG-identified information helpfulness ratings.

identified by SECLOG, and the last two variables are randomly selected from the source code. Note that the order of choices will be shuffled with Qualtrics survey utilities.

Example (PostgreSQL-4). *The following query tries to create a table referencing another table, but gets permission denied.*

```
CREATE TABLE tableA(
contact_id INT GENERATED ALWAYS AS IDENTITY,
customer_id INT,
PRIMARY KEY(contact_id),
CONSTRAINT fk_customer
FOREIGN KEY(customer_id)
REFERENCES tableB(customer_id)
);
```

Please rate the helpfulness of the following information in your diagnosis process. Each choice includes the description of the information and the value of the information.

We distributed our survey through the software's mailing list and slack channel, subreddits of sysadmin forums. In total, we received 108 valid responses (36 for Apache, 42 for PostgreSQL and 30 for vsftpd). No personal identifiable information is collected in the survey.

Results. The average helpfulness ratings of SECLOG-identified variables and randomly

Information	Value	Not helpful	Slightly helpful	Moderately helpful	Very helpful	Extremely helpful
The table name	TableB	○	○	○	○	○
Required privilege for the referred table	REFERENCES	○	○	○	○	○
User who is executing the query	staff	○	○	○	○	○
Number of columns	5	○	○	○	○	○
Default privilege for this user	No permission	○	○	○	○	○

selected variables are shown in Table 2.14. In all three applications, the helpfulness ratings of SECLOG-identified are between 3 (moderately helpful) and 5 (extremely helpful), which are significantly higher than randomly selected variables (p<0.005, Mann-Whitney U test).

The distribution of helpfulness ratings in SECLOG-identified variables is shown in Figure 2.7. For all applications, more than 50% of variables have ratings higher than 4 (very helpful) and more than 75% are higher than 3 (moderately helpful). In PostgreSQL, SECLOG can help identify the accessed table name (object), required privilege (object) and the user who is executing the query (subject) in all the locations, which are all regarded as very helpful.

We further look at the variables with ratings lower than 3 (moderately helpful) in Apache and vsftpd. This is because for library calls that SECLOG cannot analyze, SECLOG treats all parameters as related to avoid losing useful information, which may include additional less useful information. In Apache, 7 out of 12 variables are the `apr_file_t` object which stores the file handle, which are the parameters of libcalls. Similarly in vsftpd, 3 out 5 variables are introduced as libcall parameters. Future improvements may further prune such variables by adding filters for specific variable types.

2.5.3 Real-world Log Improvement Examples

We provided three examples to exemplify that SECLOG can insert variables including objects and actions (case 1), subjects (case 2) and configuration values (case 3). In addition, we provide the one log patch of the illustrating example in Figure 2.11.

```
…} else if (mode == WRITE_SEEKABLE) {
  rv = apr_file_open(fd,  pathname, WRITE|CREATE, …);
  if (rv != APR_SUCCESS) {
+     return dav_new_error( …, sprintf(“Could not open
+       an existing resource for writing: %s.”, pathname) );
  } else {
      rv = apr_file_open(fd, pathname, READ, …);
      if (rv != APR_SUCCESS)
+     return dav_new_error(…, sprintf(p, “Could not open
+       an existing resource for reading:%s.“,pathname));
}}
if (rv != APR_SUCCESS) {
   return dav_new_error(…,
-    "An error occurred while opening a resource.");
+    sprintf(p, “An error occurred while opening a
+              resource for writing: %s.", pathname) );
```

Figure 2.8: An accepted Apache patch based on SECLOG-identified information adds the filename and action into log messages.

1. Access control check function

```
int aclCheckLocal(cinfo_t *info) {
   if ( (!info->host| | !strcmp(info->host, "localhost"))){
      if(strcmp(info->user, configGet("admin"))!=0)
         return(0);          <-- Deny path 1: not admin user
   } else  {
      return(0);             <-- Deny path 2: not localhost
   }
  return(1);
}
```

2. Access control check function call site

```
void processClientRequest(…) {
   if(!aclCheckLocal(clientinfo) {
-       error("Permission denied");
+       error("Permission denied - not supported over
+          remote connections of from non-admin user");
```

Figure 2.9: The final accepted patch in mSQL adds information related to the subject.

1. Access control check function

```
int vsf_access_check_file(struct mystr* filename){
  str_copy(&s_access_str, tunable_deny_file);
  if(vsf_filename_match(filename, &s_access_str)){
    return 0;        <-- Access denied.
  }
```

2. Access control check function call site

```
void handle_cwd(struct vsf_session* p_sess){
  ...
  if (!vsf_access_check_file(&p_sess->ftp_arg_str)){
    vsf_cmdio_write(FTP_NOPERM, "Permission denied.");
+   vsf_cmdio_write(FTP_NOPERM, “Permission denied
+           because of configuration: deny_file”);
```

Figure 2.10: An log enhancement for vsftpd related to configuration, which is used in the user study problem vsftpd-1.

1. Access control check function

```
cherokee_mkdir_p_perm(buffer_t* dir, mode, perm){
  re = cherokee_stat(dir->buf, &foo);
  if (re != 0) {/*if not exist, create the dir.*/
    ret = cherokee_mkdir_p(dir, mode));
    if (ret != ret_ok)
      return ret_error;   <-- Deny path 1 return value
  }
  /* dir exist, check permissions */
  ret = cherokee_access(dir->buf, perm);
  if (ret != ret_ok)
    return ret_deny;      <-- Deny path 2 return value
  return ret_ok;}
```

variables on path1
variables on path2

2. Access control check function call site

```
cherokee_handler_rrd(…){ //...
  ret = cherokee_mkdir_p_perm(img, 0775, W_OK);
  if (ret != ret_ok) {
-   LOG_CRITICAL(“Cannot create the ‘%s’ directory”,img);
+   LOG_CRITICAL(“Cannot create the ‘%s’ directory; or
+    the directory doesn't have write permissions”, img);
  return ret_error;
}
```

Figure 2.11: Collecting access-deny information inside the access control check function.

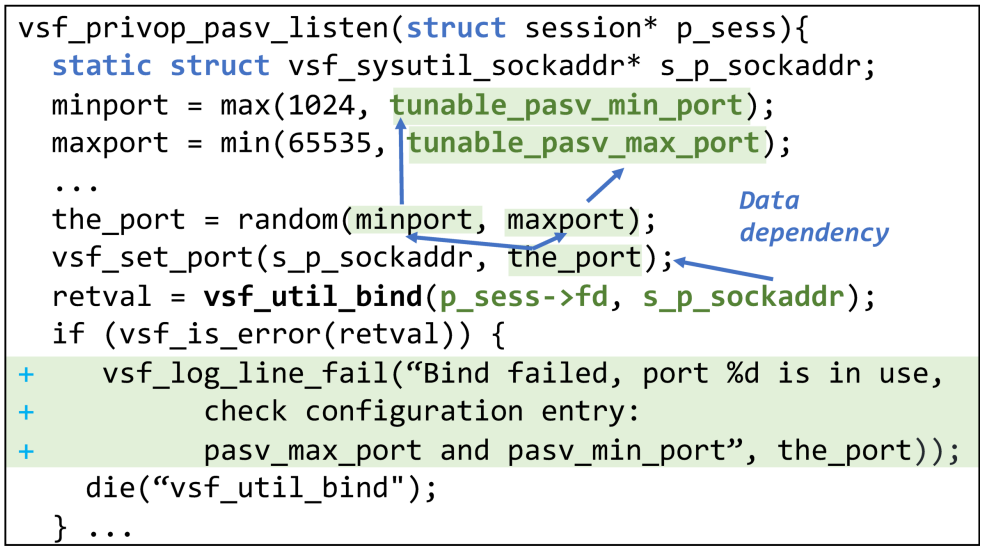

Figure 2.12: An example of new log insertions that adds configuration entries of port range.

Case 1 (Apache): Figure 2.8 shows that SECLOG enhanced the log statements to provide object and action information to guide sysadmins in fixing possible unintended access-deny issues. Originally the message "An error occurred while opening a resource" was generic with no diagnostic information while the permission can be denied for several different reasons. Such messages provide very little for sysadmins to understand what resources and which operations are denied. We submitted a patch to log the operation and the file name for all the possible failed places. The Apache developers confirmed our suggestion and accepted our patch into its main branch. This patch can directly help an access-deny issue in ServerFault [14] - when `RewriteEngine` is used, the uploaded directory may not be the original URL. Sysadmins need to know the real path and denied operation to fix it with a lower chance of over-granting permissions.

Case 2 (mSQL): Figure 2.9 shows that mSQL may deny requests from remote hosts or non-admin users. The original log only records a general error as "Permission denied" with no information. By analyzing the access-check function `aclCheckLocal`, SECLOG identifies two possible reasons on two deny paths that (1) the client is on localhost but not admin user or (2) the client is not on localhost. SECLOG improves the logging by adding the client's user and host information. The maintainers acknowledged our patch and revised the wording based on our

Table 2.15: Six real-world access-deny issues from vsftpd and PostgreSQL evaluated in our user study.

Problem	Description
vsftpd-1	Users could not download the txt file because of misspelled regex patterns in configuration `deny_file`
vsftpd-2	User could not login because the user customized config. file is not owned by root.
vsftpd-3	User could not retrieve directory listing in the passive mode because the allocated port is being used by other application.
Postgres-1	The SQL query is denied because the user lacks `USAGE` privilege on the schema and `INSERT` privilege on the table.
Postgres-2	The SQL query is denied because the user lacks `EXECUTE` privilege to run the functions in the schema.
Postgres-3	The SQL query is denied because the user lacks `REFERENCES` privilege when creating a table with foreign key reference.

patch as shown in Figure 2.9.

Case 3 (vsftpd): Figure 2.10 shows that vsftpd may deny access to certain files if the file name matches records in configuration entry `deny_file`. SECLOG identifies this configuration variable by analyzing the deny paths inside the ACC functions and includes the configuration entry in the log message. We conducted the user study using this example (vsftpd-1 §3.6.2).

An accepted patch with two access-deny reasons: The patch is shown in Figure 2.11. SECLOG detects two different access-deny return points and constructs two slices with backwards slicing. The first sliced path extracts `mode` whereas the second one is related to `perm`. SECLOG enhance the log statement by adding two reasons for the denied access.

Patch with global variables at the call site: In Figure 2.12, from the function parameters, SECLOG traces back to the global variables of the configuration settings, and adds them as relevant information in the access-deny location.

2.5.4 User Study on SECLOG

To evaluate the effectiveness of SECLOG in helping sysadmins fix issues accurately without introducing permission over-granting security mistakes, we conducted a controlled user study. Our study was approved with an IRB exempt status.

We crawled the real-world problems related to the software from sysadmin forums (e.g., stackexchange, stackoverflow) and the software's administration mailing list, and further filtered the access control related cases with keywords filtering. We examined the cases and found the common ones as our user study cases. Finally, we used six real-world problems from two server software (vsftpd and PostgreSQL) as shown in Table 2.15. For example, problem 3 in vsftpd is related to the common port range settings in FTP the server that was posted in many threads [12, 13]. We also use the real-world issue in Figure 1.1 as problem 2 in vsftpd.

We recruited our participants from each server's user mailing list and Slack workspace, and sysadmins Reddit. In total, we recruited 32 work professionals. The participants were compensated for their time with a $30 giftcard.

Ethical considerations. All our study involving human subjects (i.e., questionnaire survey and user study) were approved by the university's Institutional Review Board (IRB) with an IRB exempt status. We took several measures to protect the participants' rights. First, the researchers who conduct the study were trained with ethics for user research courses before the study. Second, the participants were informed of the purpose, rights and risks before the study and the study is totally voluntary. The participants were informed that they can opt out anytime and they still get compensated for their time in the study. Third, no personal identifiable information is collected or stored during the study.

Methodology. The study is conducted remotely using Zoom for communication due to COVID-19. The participants ssh to a remote server to resolve the designed problems. Each user is asked to resolve six problems in Table 2.15. We randomly allocate the participants to the group with SECLOG-enhanced log messages (group A) and the group with original software (group B).

The **only** difference between the groups is the information in the log messages.

Before the study for each server application, we give the participants 30 minutes to read a detailed guide on how to manage the servers with the specific sections in the official manual related to our study. Then we ask the participant to solve a *warm-up* case for each server program to help them get familiar with the server environment, such as the location of configuration files and commands to modify the settings. The warm-up case is not included in our results.

After the warm-up question, we give them three problems from each server application. The order of the problems is randomized. Each problem is given 30 minutes. If they cannot finish during the given time, we count it as 30 minutes, which penalizes the enhanced group since many participants with the original software could not resolve the problem within the maximum time. We also record how they resolve the problem by recording their commands and modifications, and then evaluate whether the fixes over-grant the permissions and introduce security issues.

After all problems are finished, participants can optionally participate in a short interview. We show the participants both the original and enhanced log messages. We ask the participants whether and how they would change their solutions and reasons for changes; And the participants will rate the usefulness of both log messages on a 5-point Likert scale. Here we present the example survey question for problem vsftpd-2.

Example (vsftpd-2) *How useful is the following log message in Problem Vsftpd 2?*

Log message 1: *500 OOPS: config file not owned by correct user, or not a file.*

Log message 2: *500 OOPS: The config file is not owned by root, or not a file: /etc/vsftpd_user_conf/ftpuser1*

	Not helpful	Slightly helpful	Moderately helpful	Very helpful	Extremely helpful
Log message 1	○	○	○	○	○
Log message 2	○	○	○	○	○

Insecure fixes. We record all the changes made by the participant during the study. We regard the solution as *secure* if the solution only grants necessary privileges to solve the

Table 2.16: The number and percentage of insecure fixes in each problem that over-granted permissions and introduced security issues, in the group with original logs and SECLOG-enhanced logs.

# (%) of insecure solutions	Problem 1		Problem 2		Problem 3	
	Original	SECLOG	Original	SECLOG	Original	SECLOG
vsftpd	1 (6.25%)	0 (0.0%)	6 (37.5%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
PostgreSQL	8 (50.0%)	0 (0.0%)	4 (25.0%)	1 (6.25%)	8 (50%)	0 (0.0%)

Table 2.17: The number and percentage of participants in each problem who cannot resolve the problem in the 30-minute time period in the group with original logs and enhanced logs.

# of unfinished participants	Problem 1		Problem 2		Problem 3	
	Original	SECLOG	Original	SECLOG	Original	SECLOG
vsftpd	1 (6.25%)	0 (0.0%)	9 (56.25%)	0 (0.0%)	4 (25.0%)	0 (0.0%)
PostgreSQL	0 (0.0%)	0 (0.0%)	2 (12.5%)	0 (0.0%)	1 (6.25%)	0 (0.0%)

access-deny issue in each problem; If the solution grants excessive privilege to the denied user or other users, it is regard as insecure. For example in PostgreSQL problems, the solution is insecure if the participant grants more privileges than required to execute the original SQL. In problem Postgres-3, only `REFERENCES` privilege is required; however, half of the participants in group with original log messages fix the problem by granting `ALL` privileges on the table to the denied user. This may be because `REFERENCES` is a less common privilege, and the original log messages only tell sysadmins a generic warning of "Permission denied" but did not explicitly tell them what privilege it lacks. As such, some participants just gave random tries like granting `SELECT`/`CREATE` privileges. When it did not work, they just chose to grant `ALL` to resolve the problem.

We show the detailed solution description for the secure fixes in Table 2.18. All the secure fixes only grant the necessary privileges to solve the access-deny issue.

We also show the details and consequences of insecure fixes, as well as the number of occurrences in Table 2.19. All these insecure fixes make intrusive changes to the system which grant excessive privilege to the denied user or more users.

Table 2.16 shows the number of insecure fixes (e.g. fixes that over-grant permissions

Table 2.18: Description for the secure fixes for the problems in user study.

Problem	Secure solution description
vsftpd-1	Delete the wrong regex pattern that matches with the download file in `deny_file`
vsftpd-2	Change the owner of `ftpuser1`'s config file `/etc/user_config_dir/ftpuser1` to the current process id, i.e., `root`
vsftpd-3	Change the port range under passive mode, `pasv_min_port` and `pasv_max_port` to a wider range
PosgreSQL-1	Grant `USAGE` privilege on the schema to user and Grant `INSERT` privilege on the table to user
PosgreSQL-2	Grant `EXECUTE` privilege on the function to user
PosgreSQL-3	Grant `REFERENCES` privilege on the table to user

and introduce security issues) introduced by the participants during the study. We find that the group with original log messages (group B) introduces 27 insecure fixes in total. In contrast, the group with SECLOG-enhanced log messages (group A) only has one insecure fix (in problem Postgres-2). The result shows that diagnostic information added by SECLOG) are effective to reduce the number of insecure fixes by sysadmins.

Diagnosis time. Besides the number of incorrect fixes, we find that with the help of SECLOG enhanced log messages, the participants are more likely to have a much faster diagnosis and fix time. As shown in Table 2.17, all participants in group A completed the task. In contrast, around one-fifth (18/96) of the tasks in group B were not completed within the time limit. For example, 9 out of 16 participants in group B could not finish task vsftpd-2. The original log message only gives a generic warning of "The config file is not owned by root", but does not pinpoint where the config file is, which makes the participants mistakenly look into the default config file. Excluding all the insecure fixes, Figure 2.13 shows that group A is 2.79x faster in completing the tasks than group B. The differences between group A and group B are all significant (p<0.05, Mann-Whitney U test) excluding problem vsftpd-1.

Post-study helpfulness survey of whole log messages. The participants can optionally

Table 2.19: Description for the insecure fixes for the problems in user study.

Problem	Solution description	Consequence	Number of occurrences
vsftpd-1	1. Delete other configuration options related to `userlist`	Previous settings limit that only users on `userlist` can login; after the change every user can login.	1
vsftpd-2	1. Delete configuration option `user_config_dir`	All the configurations customized by each user under `user_config_dir` do not take effect.	4
	2. Change permission of config file `/etc/user_config_dir/ftpuser1` to 777	All users in the system can read/write/execute the file.	1
	3. Delete the configuration option `userlist` and other options.	Previous settings limit that only users on `userlist` can login; after the change every user can login.	1
PostgreSQL-1	1. Grant `ALL` privileges on the table to user and grant all privileges on the schema to user.	The user has more privileges than required to execute the query, including `DELETE` or `UPDATE` privileges.	3
	2. Grant `ALL` privileges on the schema to user.	The user has excessive privileges on the schema, e.g., with the `UPDATE` privilege, the user can modify the schema.	3
	3. Grant `ALL` privileges on the table to user.	The user has excessive privileges on the table, e.g., with the `DELETE` privilege, the user can delete the table.	1
	4. Grant `INSERT` on all tables in the schema to user.	The user has `INSERT` privileges on all the tables, so the user can modify other tables.	1
PostgreSQL-2	1. Grant `ALL` privileges on all functions in the schema to user.	The user can execute all the defined functions in the schema.	2
	2. Grant `EXECUTE` on all functions in the schema to user.	The user can execute all the defined functions in the schema.	1
	3. Grant `ALL` privileges on all tables in the schema to user and grant `ALL` privileges on all functions in the schema to user.	The user has all the privileges to the tables and functions in the schema.	1
PostgreSQL-3	1. Grant `ALL` privileges on the table to user.	The user has more privileges than required `REFERENCES` on the table.	8

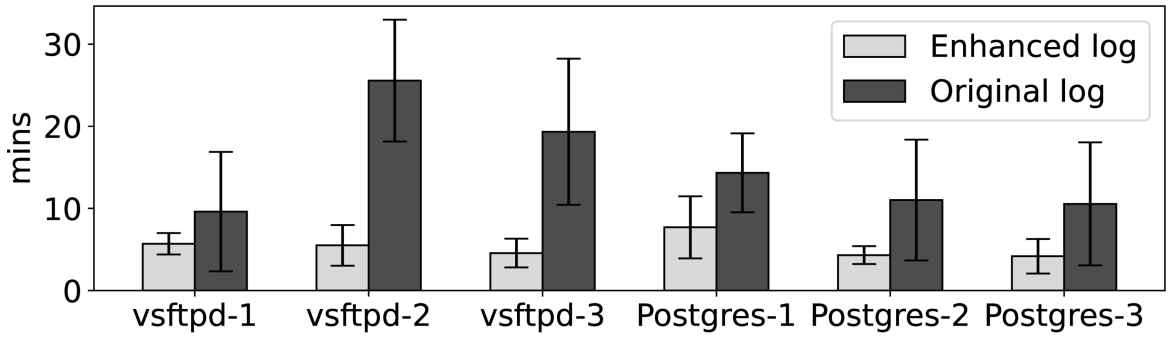

Figure 2.13: The comparison of average completion time excluding insecure fix cases.

Table 2.20: Helpfulness ratings of original and enhanced log messages in the problems.

Helpfulness Rating	Original			SecLog-Enhanced		
	Avg.	Median	Distribution	Avg.	Median	Distribution
vsftpd-1	1.80	2		4.36	4	
vsftpd-2	2.28	2		4.64	5	
vsftpd-3	1.32	1		4.80	5	
Postgres-1	2.40	2		4.52	5	
Postgres-2	2.44	2		4.56	5	
Postgres-3	2.16	2		4.60	5	

participate in a short interview to review the original and SecLog-enhanced log messages and rate the helpfulness. One survey question example is shown in Appendix **??**. In total, 11 participants in group A and 11 in group B participated in the interview. For participants who chose to grant `ALL` privileges in the PostgreSQL problems, they noticed that excessive privileges were granted and would choose to give only necessary privilege if they had seen the enhanced message.

25 participants finished the helpfulness ratings of log messages in the six problem on a 5-point Likert scale. The results are shown in Table 2.20. We find that the enhanced log messages are rated as more helpful than the original messages. In particular, the original log message in problem vsftpd-3 was only rated as 1.32 (close to "not helpful"). The log message only shows "vsf_sysutil_bind". As one participant commented that *"the old log messages were only written*

Table 2.21: Performance overhead of Apache httpd with enhanced logs. The data in gray are characteristics of the workload. "Deny" is the percentage of accesses with access-denied status. "Slowdown" is the total execution time to replay the workload; "Log size" is the size of error log.

Workload	# Access	Deny %	Slowdown %	Log size increase %
`Course`	42K	0.6%	0.0%	0.3%
`Group`	250K	13.0%	4.7%	23.8%

so that the original developers of the software would understand." In comparison, the enhanced log messages give more detailed information about the errors, with a rating of 4.80.

Limitation. The user study has limitations as it is conducted in a controlled environment and users may have different levels of experience and expertise. We tried several things to reduce the bias: (1) We recruited 32 users with relevant server management experiences; (2) We randomly and evenly distributed them to two groups; (3) We provided each with a guide and related manual sections and gave them 30 mins to read these guides; (4) We gave them warm-up questions to get them more familiar with the software and environment.

2.5.5 Performance and Overhead

We first measure the performance of SECLOG's static analysis. The result (Table 2.7) shows that the analysis time is almost proportional to the number of ACC functions' call sites in the application. We optimize the analysis time by allowing SECLOG to re-use the analysis result of functions via summaries. SECLOG is single-threaded but it can be further parallelized by analyzing function summaries concurrently.

Next, we measure the performance overhead of Apache httpd's SECLOG-enhanced version. All enhanced/inserted log messages have `ERROR` log level which is enabled by default. We use two real-world workloads from a university CS department's websites. `Course` hosts courses pages and `Group` hosts one research group's website. The result is shown in Table 2.21. (1) For the throughput, we find that on the `Group` workload, the throughput was downgraded by 4.7%. This

may because the percentage of the access-denied requests is higher. (2) For the error log size, it increased by 26% on the `Group` workload. We also calculated the average size of access-deny log messages caused. The performance overhead may be larger with a higher percentage of access denial. For potential optimizations, developers may need to improve the common path, i.e., reduce the logging overhead on common types of access-deny errors.

2.6 Discussions and Limitations

Logging Templates. Logging templates are adopted in many applications to report error messages for specific type of errors. One potential approach to improve logging quality is to enforce the logging practice via logging template. With the logging templates, developers only need to fill in the variables required by the template to generate the final log message. However, if the template lacks detailed information, the logs will systematically lack information in all places. Therefore, it is promising to improve the logging templates to help developers write better log messages. SECLOG can be used to detect the missing information in the logging templates and developers may use the detected missing information to improve the logging templates, too.

Generating human-readable log messages. SECLOG can not generate human-readable log messages automatically. Instead, SECLOG can insert variable names and values in the log messages. To make the log messages understandable by sysadmins, the semantics of log messages still need to be processed by the developers. Developers can also design utility functions to automatically process the semantics of certain types of variables to readable strings. Moreover, SECLOG can show all the identified locations and variables to help developers to know where to enhance or add access-deny log statements. SECLOG can be used as a plugin in programming IDE and suggest developers with the potential log locations and information. Some recent works [50] may be combined with SECLOG to generate human-readable log messages.

Coverage of ACC locations and relevant information. SECLOG may miss ACC locations if developers do not provide a complete list of ACC functions, or the software performs

ad-hoc checks without ACC functions. As discussed in §2.5.1, the new developers can find ACC functions with high coverage in a short amount of time; for software maintainers, the effort could be less. The developers may also refactor some code to adopt general ACC functions instead of ad-hoc checks.

SECLOG may also miss relevant information that is far in the call chain. However, this should not be common since Finding 2 shows – most (70.8%-100%) of the relevant information is usually available in the same function. Besides, tracking further in the call chain may include too much irrelevant information that can confuse sysadmins.

Security and privacy concerns. Server logs contain valuable information for developers and administrators in the troubleshooting process, which may be leveraged by attackers to gain system details. Therefore, various laws and standards have been proposed to enforce the compliance requirement in log storage, analysis and disposal [26, 47]. SECLOG assumes that the server logs should be kept secure from attackers.

As a static analysis tool, SECLOG cannot differentiate what information is considered sensitive and what should not be visible to sysadmins. Existing works on automatic private information filtering techniques [28, 102, 92, 121] can be combined with SECLOG to filter variables that may potentially leak private user information. In addition, when the SECLOG-identified information is processed by developers, developers can decide whether the information is sensitive and should be included in the log message.

Providing Secure fixes. Even though SECLOG can provide more information for sysadmins to understand unintended access-deny issues correctly, SECLOG cannot solve the problems for sysadmins. This means that SECLOG cannot eliminate the chance of sysadmins making mistakes. As we have shown in our user study, while SECLOG can significantly reduce the number of insecure fixes from 27 to 1, there is still one insecure fix even with SECLOG enhanced log messages.

2.7 Acknowledgement

Chapter 2, in full, is a reprint of the material as it appears in the 32nd USENIX Security Symposium, 2023. Bingyu Shen, Tianyi Shan, Yuanyuan Zhou. “Improving Logging to Reduce Permission Over-Granting Mistakes”. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Finding Blind Spots in Access-Deny Issues Diagnosis with Multiview

3.1 Introduction

3.1.1 Motivation

Access control is critically important for many enterprises to protect the systems resources and users' data from unauthorized access [55, 97]. As the business grows and evolves, the organization's access-control configurations require modifications 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 [108].

Inaccurately fixing access-deny issues often results in permission over-grant mistakes. According to a recent study [108] 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

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 3.1: A simplified access-deny issue with root cause in the file permission.

given on the forum are ill-advised as they lead sysadmins to introduce permission over-grant mistakes. In an Apache server access-deny issue [84], the answer that received the most upvotes is to completely disable the access control of Apache and allows all users, including attackers, to access the resources. Sysadmins can easily create access-control misconfigurations through trial and error or randomly adopt online advice when they encounter an access-deny issue, thus making permission over-grant mistakes.

Permission over-granting mistake is one of the major triggers of security incidents [57, 115, 95, 39, 78, 67, 94]. In general, users only notify sysadmins when their requests are denied, and rarely notice or complain about excessive permissions. In recent reports [52], over-granted permissions were only discovered after a security incident when the system has been exploited by malicious attackers [76]. Indeed, many enterprises have suffered from security incidents caused by permission misconfiguration, including data breaches, ransomware and system compromises [97, 82, 42]. From 2018 to 2019, the number of records lost by misconfiguration rose by 80% as did the cost to the associated companies [42]. In 2021, the non-profit security community OWASP also chose broken access control as one of the top 10 vulnerabilities in web applications [68].

In a perfect world, sysadmins can always find the solution that fits the security context. However, this is hard in real-world scenarios because each access-deny issue may have multiple

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

directions to relax the security configuration to solve the issue. Each direction may have different security consequence. 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 3.1 and 3.2). Each figure includes the request information, related access-control configuration and different possible directions.

Figure 3.1 shows an example of a web request being denied due to lack of access to the file system [49]. The root-cause is that the process owner `apache` lacks execute permission on upper-level directories. There are many directions, each has different security implications, and many of them have potential security vulnerabilities. One solution is relatively safe by granting `apache` execute permission while keeping original owner `Alice`'s permission. However, many real-life cases [49] with similar root causes often resolve the issues by disabling the protection (`chmod -r 777`) recursively from the home directory, as suggested on online forums [87]. Part of the reason is that they do not know which directory is causing the problem, and aren't sure which permissions are required for that directory. Another common risky suggestion we can often see on the forums [17, 91, 85, 86] is to disable all execute protection (`chown +x`) on the directory and now anyone can traverse this directory. Another way is to make `apache` the owner of the directory

which hurts the availability of `Alice`. This suggestion needs to be evaluated by sysadmins based on the context. All directions can resolve the issue, but they have different implications which may not fit the security context.

Figure 3.2 is adopted from a real-world access-deny issues [84] caused by server configuration. We added more context for audiences to understand the security consequence. 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 generate a more specific directory block. Moreover, sysadmins may misunderstand user's role and add Alice to the group `staff` which over-grants `Alice` permissions. This case is also used in our user study to evaluate whether participants can make less risky resolutions with our tool's diagnosis.

These examples show that access-deny issues can have multiple directions 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 for resolving access-deny issues. Their blind spots can prevent them from finding the direction that fits the security context. Without an 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 highly important to relieve sysadmins' burden of exploring all the different possible directions and assessing the security implications of each to reduce insecure fixes for access-deny issues. Unfortunately, at present, the general method involves no tooling support. Most previous works on misconfiguration diagnosis [24, 34, 116, 62, 63, 54] cannot help sysadmins explore possible fixing directions. Some previous works [89, 117] provide recom-

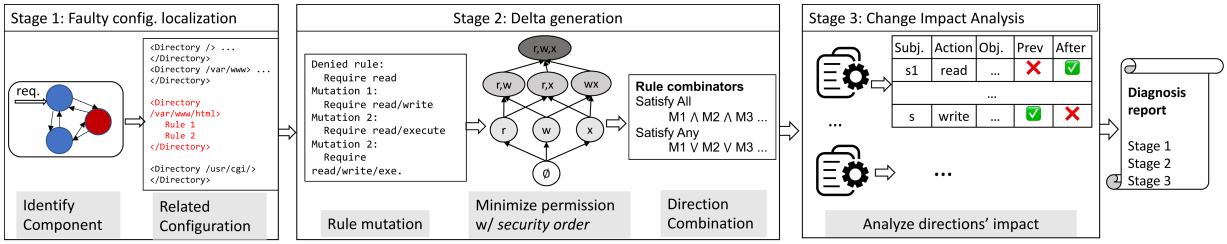

Figure 3.3: High level workflow of Multiview.

mendations 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 [89]. Sysadmins might blindly trust the solution provided by the tool and unknowingly introduce security risks.

3.1.2 Our Contributions

Our work focuses on helping sysadmins find possible directions and understand the security impact in order 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 directions to allow the legitimate request and evaluate each direction's impact. 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 on each direction with the pre-defined partial security order and only presents the directions with least permissions in the security order.

Multiview faces three major challenges in resolving the 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 on each direction? (3) How to systematically evaluate each direction's impact on the global access-control state to help sysadmins find the one 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 helps to narrow down the access-control configurations related to the request. Multiview can help diagnose the issues in multiple server applications through customized APIs to understand the syntactic and semantic meanings of the configuration files. These APIs can be implemented by the server 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 each direction grants. Then Multiview collects all the non-comparable directions which each contains least permission changes (§3.4).

To address the third challenge, Multiview needs to compare the system-level impacts of each direction by revisiting all the data in the system after each direction's change has been applied. 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 64 real-world cases from four widely used systems and server applications including UNIX file system, Apache HTTPD, Nginx and Vsftpd. Multiview can successfully find the directions to change the configuration on 45 out of 64 cases. For all the cases, Multiview finishes its diagnosis within 1 minute.

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%.

3.2 Overview

3.2.1 Multiview Workflow

Multiview aims to help sysadmins diagnose access-deny issues by providing possible directions and the security impact of each direction. Multiview's workflow is shown in Figure 3.3. 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, such as the rules in the configuration block as shown in Figure 3.4 (§3.3). 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 (§3.4). Last, Multiview analyzes the impact on the global access-control state for different directions. Multiview synthesizes the requests based on the available roles and resources and replay 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 (§3.5).

The generated diagnosis report includes the detected configuration rules, the possible directions to allow the request along with each direction's impact on the access-control state of the entire system. But still, sysadmins need to determine which direction best fits the security context.

3.2.2 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 log file which contains the information related to the denied request, or manually specify such information. Multiview performs the analysis in a packed virtualized environment which contains the same settings as in a production environment (§3.3.3). The following analysis process is automatic and generates a diagnosis report for the access-denied issue in the end.

To make one piece of software fits 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 location of the software's configuration file; (2) the information related to the denied access, or the software's log which can find such related information; (3) all subjects, actions and objects in the application, e.g., the directory to host HTML files and the users in the system, for Multiview to perform the change impact analysis.

3.3 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 as *toggle analysis*, as it mimics how

sysadmins narrow down the causes of access-deny issue in real-world scenarios.

3.3.1 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 an 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 web server needs to pass the check of the file system on behalf of the end-user in order 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. For example, SELinux provides APIs to temporarily disable the checks by one single command `setenforce 0`.

3.3.2 Identify Request Related Configurations

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 [21]. 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

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 3.4: A simplified Apache HTTPD configuration example.

request, and then proceed to probe whether the user lacks permissions for each object. To process a single request, the system often needs to access multiple objects to complete the request. For example, when a user wants to read a file in the file system, the user must also have the search permission on upper-level directories [49]. 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 3.4. 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. Depending on the configuration, the user needs to satisfy all matching blocks or only the longest matching block [61].

To identify the related objects, Multiview first extracts the original object in the denied request by extracting the access log information. Based on the characteristics of the denied object, Multiview maps the requested information to the objects in the system that the user needs to have permission with the APIs provided by the developers. Multiview leverages the domain-specific characteristics to collect all the objects the user may need to access in each component. To probe

whether the user lacks permission for each object, it continues to use toggle analysis to change the access control rules of any object to see if the change affects the result of the request.

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 components. Unix file systems use 9 permission bits to represent access control for owner, group and others. Common web applications use directives which can place 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 3.4, multiple access-control rules exist in the configuration block, and the result is other allowed (marked in green) or denied (marked in red) for each rule. (We will discuss more about the combination operators in §3.4.4.)

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.

3.3.3 Toggle Analysis Implementation

Multiview APIs for Toggle Analysis. Multiview requires some APIs to be implemented for each software to perform the toggle analysis at different levels, including components, objects and specific rules. Multiview defines an API `toggle_component()` to toggle the access control result in a single component. The application developers can implement the function as a one-time effort. For general server software configurations, we can try to remove all the rules related to the access control and add one rule to ensure all users will be allowed by default.

To find the access control configurations related to the issue on the object-level, Multiview

Algorithm 2: Toggle Analysis

```
Function ToggleAnalysis(components):
    faulty_comp ← None
    faulty_objects ←[]
    related_rules ←[]
    for comp ∈ compnents do
        if toggle_component(comp)=allowed then
            faulty_comp = comp
            break;
    for obj ∈ faulty_comp do
        if toggle_object(obj)=allowed then
            faulty_objects.append(obj)
    for obj ∈ faulty_objects do
        if toggle_rules(obj)=allowed then
            related_rules.append(<obj, get_rules(obj)>)
    return related_rules
```

provides an API named `toggle_objects(component)`, in which the software's developers can specify how systems find the related objects. Due to the nature of the file system hierarchy, the logic is usually to require all objects with matched prefixes or just the longest matching. Last, Multiview provides `toggle_rules(object)` to test each access-control rule within the object is to allow or to deny. For example, in the Apache configuration as shown Figure 3.4, developers can implement APIs to examine the result of each rule by only keeping each rule in the block.

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 from each component. On the object and rule level, the system checks each resource's permission respectively to achieve the final access-control result.

Safety Considerations. When Multiview performs the toggle analysis to find related configurations in the component, Multiview needs to analyze the results by applying changes and replaying requests. 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 it is hard to ensure separate environment to be consistent with the production.

Multiview tackles this challenge in two ways. 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 uses copy-on-write techniques to share the same copy of data and configurations as production [59, 72]. When a modification happens, Multiview will create a copy of the original data and the modification is only applied on the modified version without affecting other production systems.

3.4 Delta Generation

3.4.1 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. Therefore, Multiview aims to help sysadmins find as many problem-solving ideas as possible by systematically modifying access-control rules and combining them as possible directions.

Multiview finds different directions to resolve the access denial, thereby loosening the access control on denied data, and minimizing the granted permissions in each direction. 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 denied request. Therefore, each possible direction 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 in each direction and grant as few

Table 3.1: Definitions and examples of different types of modification that can relax the access control and permit the access-deny issues.

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.

permissions as possible, we developed a new partial order metric called the *security order* to measure and compare the number of 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 directions. Multiview provides directions, not solutions, as we only want to provide more insight on possible ways to solve the access-deny issue. To find the best suitable solution, sysadmins should further confirm and determine the direction that fits the security context of the organization.

3.4.2 Access-Control Rule Mutation

We propose a method to systemically search for possible directions to resolve the issue. At this point, Multiview has found objects for which the user does not have enough permission, and the access-control rules on each object that deny the request. Multiview comes to help by mutating 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 (§3.4.3).

First, we classify methods of changing access control into three types based on how access-control rule is relaxed to resolve access-deny issues. Table 3.1 gives the definitions of each type of mutation and provides real-life examples. We denote the subject, action and object

in the access-denied request as s_{deny}, a_{deny} and o_{deny}. s_{deny} contains the subject's information such as the user's role in the organization and the subject's ip address, etc. Multiview can relax the restriction on s_{deny} by assigning user to different roles in the access-control configuration or allow user-specific attributes. Taking file systems as an example, each system user can have roles of owner, group or other, to different files. Each role implies different privileges. By changing s_{deny} to a role with more privileges, Multiview relaxes the system's restrictions on s_{deny}. Second, Multiview can relax the restrictions on a_{deny} by granting s_{deny} with more privileges to perform actions on the same object. For example, the file system has read, write and execute permissions for each role, and granting different permissions would allow different actions on the object. Last, Multiview can relax the restrictions on o_{deny} by allowing s_{deny} to access new objects (including the denied object). This can be done by adding new access-control rules for the denied objects. Because each category changes a different aspect of the access-control rule, we can relax the access-control to different degrees by a combination of these three categories.

Multiview needs to understand the possible ways to mutate s_{deny}, a_{deny} and o_{deny}. From the results of faulty configuration localization, Multiview is able to 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 applied on 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 first 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 based on the replayed request's status. 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. For configuration settings with no user authentication, Multiview checks if there are other attributes related to the subject, such as IP or hostname to concretely describe the request characteristics and make relaxations accordingly.

3.4.3 Minimize Permissions with Security Order

Multiview may produce multiple rule mutations to change the result of the access-control rule, where each mutates the same aspect of the access-control configuration. 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 permission may introduce security risks, Multiview should prune these rule mutations to avoid misleading sysadmins. Multiview's directions should grant as less permissions as possible and allow the denied request.

We developed a partial security order to compare different directions. This can be served to early terminate mutation process, if one mutation of the rule can already resolve the access-deny issue, 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*

$R(c_1) \subseteq R(c_2)$.

When there are 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., $\alpha_{c_1}(s_{deny}, a_{deny}, o_{deny}) = \alpha_{c_2}(s_{deny}, a_{deny}, o_{deny}) = 1$, we rely on the partial security order to compare them. In particular, if $c_1 \preceq c_2$, the set of access granted by c_1 is a subset of those granted by c_2. Hence, c_1 will be a more secure solution if we regard less access as more secure. For example, in webserver configuration, the request is denied because one of the IP subnets in the blacklist contains the IP of the denied user. Following the partial security order, it holds $c_{\text{ip}} \preceq c_{\text{ip subnet}}$. Therefore, it allows less access by allowing a single IP than deleting the entire subnet restriction.

Handle incomparable possible rule mutations. Note that the ordering for the access status images of configuration is a partial order. There are situations where two mutations are not comparable, i.e., when the symmetric difference of two sets $R(c_1)$ and $R(c_2)$ are not empty, we can not determine which one grants fewer permissions 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 to access the object. Two different mutations can allow the request, but because the members of the two roles are different, the access status images cannot be subsets of each other and they bring different impact on the access control result. Therefore, Multiview collects all the non-comparable rule mutations to generate the final directions for the sysadmins to determine which one to choose based on the security context.

3.4.4 Generate Final Directions

Multiple rules may exist in an object's ACL to represent the access control policy. When multiple rules of the object deny the request, Multiview first generates possible rule mutations to resolve each rule and then combines the mutations based on the relationship among the rules. For example, in the web server configuration scenario, there could be multiple rules associated

with the configuration block to provide access control on the subject or actions, as shown in Figure 3.4. The user's group may not be allowed to view data under the directory or the user's IP is blocked by misconfigured IP ranges. Certain methods (or actions) like POST, DELETE may not be allowed.

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 three combination operators for the directive's result, Satisfy All($\wedge$), Satisfy Any ($\vee$), and Negate ($\neg$). The rules' result can be combined with the above operators to represents one 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 to make the request to be allowed are $\mathbb{C} = \bigtimes_{C_x \in \{C_{p_i} | r_i = True\}} C_x$, 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 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 are to make the request to be allowed are $\mathbb{C} = \bigtimes_{C_x \in \{C_{p_i} | r_i = False\}} C_x$, which represents all the rules needs to be denied to make one final direction.

The combination orders can be retrieved from the filtered configurations based on keywords. For example in Apache configuration (Figure 3.4), the keyword `RequireAll` represents Satisfy All, and the keyword `RequireAny` represents Satisfy Any [60]. 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.

Generating exception. Multiview also generates a direction works as an exception to allow the denied request for sysadmin's reference. There are situations where Multiview can generate a more restrictive direction. Multiview takes all the request's attributes to generate a specific exception to only permit this user to access the single request object. When sysadmins need to quickly work around the situation but do not know what is the best way to change the configuration and fit it into the security context, sysadmins can adopt this restrictive direction as a hot patch to only permit the single request. This exception may not be desired as it may add more burden on future maintenance.

3.5 Change Impact Analysis

Multiview further inspects the impact of each configuration change on the overall 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 these changes in order to judge whether these changes are intended based on the organization's security goals. To help sysadmins evaluate whether the generated configuration change is intended, Multiview applies change impact analysis to analyze the impact of configuration changes on the global access control state. Furthermore, sysadmins can compare different directions suggested by Multiview by examining the access-control state changes, where Multiview can not directly compare them with the defined security order.

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

Table 3.2: The example subject, object and action in different software systems.

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

configuration change can be represented as a set of tuples (s,a,o,r,r') where $r = \alpha_{c_{original}}(s,a,o)$ and $r' = \alpha_{c_{direction}}(s,a,o)$ 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 3.2.

Comprehensive requests generation To characterize the global access-control state comprehensively, the impact analyzer needs to collect all the subjects, objects, and actions that sysadmins are interested in. 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 that they are interested in. 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 configuration changes to reduce the number of related subjects and objects.

As described in Section 3.4, the generated configuration change is composed of one or more mutations on configuration entries. Based on the specific mutation on each entry, we can find the related subjects and objects. For example, if the mutated file permission is only related to a specific file path, only the access control state of files under this directory will be affected, therefore the impact analyzer only needs to walk through the directory to collect the objects.

Table 3.3: Results for 11 representative real-world access-deny issues.

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
	3	Even with the correct configuration file, the Apache server is still unable to process the request because it cannot access one .htaccess file.	Yes	2
	4	The Nginx master process user cannot access data shared with other services because it is not added to the ‘www-data‘ group.	Yes	4
	5	After enabling custom configuration, users cannot access through FTP because FTP cannot read the custom configuration in the file system.	Yes	1
Apache	6	Anonymous requests are blocked because the default configuration of Apache blocks the request method.	Yes	2
	7	Authenticated users cannot access internal web pages due to improper implementation of role access-control configuration in Apache.	Yes	4
	8	The company machines cannot access the internal pages of Apache server due to an incorrect IP whitelist in the configuration.	Yes	1
	9	After several failed attempts to change the configuration file, the sysadmin is unable to allow the user to access the directory because the changes were placed in the wrong location that did not match the request.	Yes	2
Nginx	10	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	11	The configuration entry `deny_file` contains an incorrect settings that matches all filenames, causing all FTP commands to be access denied.	Yes	1

3.6 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 64 access-deny issues for four 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.

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.

3.6.1 Real-world Access-Deny Issues

Methodology. We choose popular open-source web applications as our targets including Apache HTTPD, Nginx, and vsftpd. We collect a set of access-deny issues from mailing lists and sysadmin online forums. 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. 272 cases are crawled, and we examine each case and exclude issues not related to access-deny. In the end, we have 93 access-deny issues.

Next we try to reproduce these cases based on the description and root causes by recreating their server configuration file and file system permission. We have to exclude 29 cases with no

concrete causes which we cannot reproduce. We reproduced 64 cases. Multiview can diagnose 45 cases of them. We analyzed the root causes of these issues and categorized them according to their causes. Because many cases had similar root causes, we select 11 common cases from all categories to present in Table 3.3 along with Multiview's report summary including related configuration, possible directions.

Results. For each access-deny issue in Table 3.3, 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 that may resolve the access-deny issue. For the change impact analysis, because we do not have the data from the original issue, we do not have results for the impact analysis. We will discuss more about the usefulness of impact analysis with our designed scenarios in the user study (§3.6.2).

For all the cases that Multiview can successfully locate to the configuration, Multiview can find the possible directions based on the mutations of subject, action and object. Multiview can successfully find 1-10 directions that can resolve the access-deny issue. Sysadmins need to select the direction based on the access-control context. Note that the directions found by Multiview also cover original post's direction that is secure based on the problem context. Case 2 has 10 directions because there are multiple denied data. Multiview combines different mutations for each data, resulting in 10 directions.

There are 19 cases that Multiview cannot diagnose. Among all, 11 of these issues are related to SElinux. SELinux has diagnosis tools such as `audit2allow`, we can also integrate Multiview with it in the future. Some of these cases are related to specific modules that only apply to specific applications, as we have one case for mod_ssl and three cases for mod_security. After locating the related rules, Multiview has no way to mutate the configuration. Also, Multiview could not handle syntax errors. there are four access-deny issues caused by the syntax change between the versions. With syntax errors, Multiview cannot correctly parse and locate the faulty configuration.

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

Figure 3.5: Multiview diagnosis report for case 1.

We use two case studies to show how Multiview's reports can help sysadmins diagnose the access-deny issues.

Case 1: File System. Figure 3.5 shows a common file system access-deny issues on forums and 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 users did not know that in the file system 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 may try to grant unnecessary privilege 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. 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 direction fits more to 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 can resolve the access-deny issues and grant fewer privileges compared with forums' answers.

Case 2: Apache Configuration. Figure 3.6 shows an issue caused by Apache server's

<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><pre><Directory /var/www/sales/stats/>
  <RequireAll>
    Require method GET POST
    <RequireAny>
      Require group admin ◂ not passing
    </RequireAny>
  </RequireAll>
</Directory></pre></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
<pre>admin: sysadmin, Alice</pre></td><td rowspan="2">Direction 3: Allow Alice's current role <code>sales</code> to access the block
<pre><RequireAny>
  Require group admin
  Require group sales
</RequireAny></pre></td></tr>
<tr><td>Direction 2: Allow Alice's current role <code>sales_manager</code> to access the block
<pre><RequireAny>
  Require group admin
  Require group sales_manager
</RequireAny></pre></td></tr>
<tr><td></td><td>Direction 4: Only allow Alice to access
<pre><RequireAny>
  Require group admin
  Require user Alice
</RequireAny></pre></td></tr>
</table>

Figure 3.6: Multiview diagnosis report for case 7.

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 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. These directions all can resolve the issue, but have different implications on the access-control state. Sysadmins need to determine which direction is most suitable based on the scenario with the help of change impact analysis.

3.6.2 User Study

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

Table 3.4: Problem descriptions of user study. All the problems were designed based on real cases. Each question represents a common category of access-deny issues.

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.

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 3.4. 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 §3.3, (2) the specific rules related to the denial §3.3, (3) the possible directions to resolve the problem §3.4 and (4) the change impact analysis for each possible direction §3.5.

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 then suggest possible measures to resolve the issue securely. After that, the expert evaluates the resolutions from the 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.

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.

Results. Table 3.5 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 that the percentage of participants in the treatment group that introduced insecure fixes was lower for each question compared to the control group.

Table 3.5: The percentage of participants who made security mistakes for each problem in user study.

	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%

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 [17, 91, 85, 86]. 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.

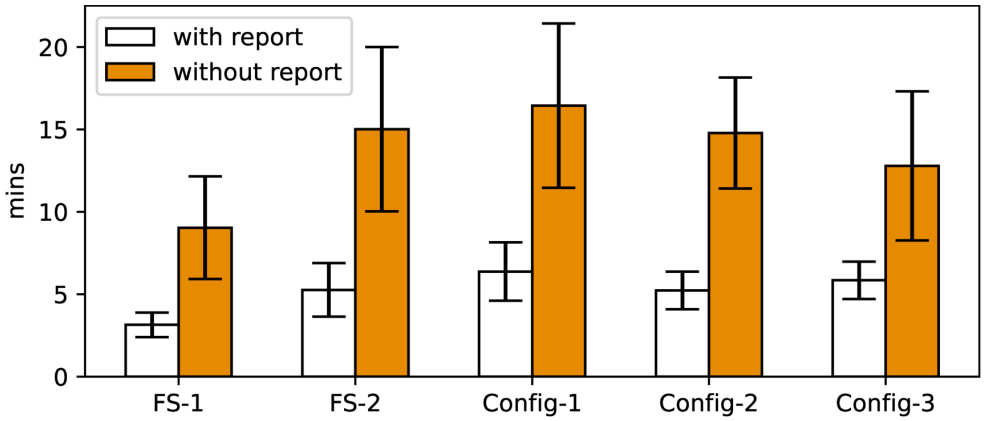

Figure 3.7: The average completion time for each user study problem. The error bar shows the 95% confidence interval.

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 smaller 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 less security mistakes.

3.6.3 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 annotating the security order for the access-control rules. 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 §3.6.2.

Table 3.6: Number of LOC needed to adopt Multiview for each application.

Application	# LOC
Multiview Framework	1967
File system	256
Apache	482
Nginx	364
Vsftpd	280

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.

3.7 Limitations and Discussion

Multiview's goal is to help sysadmins collect the possible directions and evaluate their impact to understand and fix the access-deny issue correctly. However, Multiview can not determine which direction should be taken 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 direction with proper models of the security properties.

Multiview takes a black-box approach in the faulty configuration localization (§3.3), in which Multiview mutates the result of access control at different levels to identify faulty configurations. This requires Multiview know all the related components and their configurations to start the mutations. However, some software's configuration options have default values that are not explicitly declared in the configuration file. 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 [88, 18, 19, 20] to analyze the implicit configuration options.

Besides, Multiview's mutation policies to generate possible directions require the software developers to provide the APIs for mutating each individual access-control rule, as well as the partial security order. Therefore, the directions 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.

Multiview performs the change impact analysis to find each direction's impact on the access-control state. The number of subjects and objects could be very 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 state changes as future work.

Lastly, 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.

3.8 Acknowledgement

Chapter 3, in part, is a reprint of the material under submission. Bingyu Shen, Tianyi Shan (co-first authors), Yuanyuan Zhou. “Multiview: Finding Blind Spots in Access-Deny Issues Diagnosis”. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Related Work

4.1 Empirical Studies

4.1.1 Access-deny Issues Characteristic study.

Xu et al. [108] studied the access-deny issues from a Human-Computer Interaction (HCI) perspective by looking at the practices of how sysadmins resolving access-deny issues from online forums. Even though their study shed light on the important problem that many current server programs do not provide adequate feedback information, but they did not study further into the details. However, *no solution was proposed and evaluated in their study*. Our work not only provides additional insights to their findings (e.g., Finding 3 on commit history) but also provides a solution to enhance access-deny log messages and insert new log statements. Multiview tackles the problem by finding precise information related to the issue. Besides, Multiview further perform delta analysis to find more directions to resolve the access-deny issue and evaluate their impact on the global access-control state.

4.1.2 Empirical studies on security practices of sysadmins.

There are a few empirical and HCI studies on some reasons or phenomenons of sysadmin mistakes. Fahl et al. [43] observed that a large percentage of sysadmins who operate on HTTPS websites used non-validating certificates deliberately, because of little tooling support and few affordable certificate options. Dietrich et al. [41] found that besides personal and environmental factors, the systems' poor support can cause misunderstanding and misconfigurations. Li et al. [58] found that sysadmins have faced challenges in updating server programs such as handling update-caused issues and deploying updates without disruptions. Continella et al. [31] identified misconfigurations that could cause unsecured Amazon s3 buckets and suggested stricter default policies and warnings for sysadmins to mitigate this issue. These works motivate for better support and assistance for sysadmins in managing server software. Our work is exactly along this direction.

4.2 Misconfiguration Detection and Diagnosis

4.2.1 Access-control Misconfiguration

Many works have been proposed to detect the inconsistencies in the access control policies with data mining [106, 24, 34, 116], testing [62, 63], and verification approaches [45, 54]. 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 [24] 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, Multiview 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. SECLOG is trying to help sysadmins gather useful information

understand an unintended access-deny issue and fix the setting correctly without over-granting permissions.

4.2.2 Other Types of Misconfiguration

Previous research [100, 104, 103, 88, 116, 113, 18, 19, 20, 107, 71, 117, 118] has applied various types of techniques to the configuration error detection and diagnosis, including black-box and white-box approaches. The black-box approaches [100, 104, 103, 88, 116] try to understand the dependency between configuration changes and software behavior by using the external observations of the systems, such as functional configurations or predicates defined by users.

In contrast, the white box approaches [18, 19, 20, 107, 71, 117, 118, 119] rely on the source code and compare the inside structures of software to automate misconfiguration troubleshooting. These works aim to find the related configuration entries by using static and dynamic analysis or runtime profiling, and compare the changes in the code execution path.

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, Multiview 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.

4.3 Improving Logging Messages

LogEnhancer [114] and its follow-up works [120, 112] improve logging to collect more diagnostic information for software developers to troubleshoot software bugs. As briefly discussed in §3.1, SECLOG differ from their work in several major ways: (1) They target to collect *intermediate variable* values to help *developers* narrow down the search paths for root cause analysis. Our work focuses on adding/enhancing access-deny log messages for *sysadmins* to understand and fix the reported access-deny issue correctly without over-granting permissions or introducing security issues. As sysadmins do not refer to source code or even have access to source code [108], logging intermediate variables is meaningless to sysadmins. (2) Different goals also lead to different technical challenges and solutions. Their work started from a failure or an error to backtrack important intermediate *control variables* values to help developers reduce the search space, whereas SECLOG starts from access check function call sites, and conduct both backward and forward data and control flow analysis to identify access-deny path, relevant information such as subject, object and action which *determine the outcome of an access check function*. In a way, while we also consider control-dependency, SECLOG looks more into data flow from the result of an ACC function, whereas their work focuses more on control flow analysis to narrow down the possible execution path for developers.

4.4 Access Control Management

4.4.1 Proactive Access Control Management

Access control matrix [55] is proposed in the early days to protect computer systems. Many access control models have been proposed to cope with different application context, including basic access control matrix [55], discretionary access control [70] for file systems, role based access control for databases [44], attribute based access control for Apache web server [51],

and numerous variants [99, 46]. The complex models as well as the different syntax and schema in various software make it hard for sysadmins to manage the policies [23]. These work facilitates proactive access control management with tools and frameworks, such as customizing access control frameworks based on usage context [32, 101, 48], simplifying the implementation of policies [30, 98] and automatically generating secure policies based on security goals [25, 27, 66]. Other works also looked at how to design usable interfaces to help sysadmins review policies [53]. However, the previous works are usually highly tailored to specific systems or software, and require administrators formally define security goals. SECLOG is orthogonal to these works by providing a general approach to improving the access-control logging which aims to help sysadmins gather useful information when an access-deny issue happens.

4.4.2 Least-privilege Policy Generation

A line of work has been proposed to use the audit log data to generate least-privilege policies in different environments [80, 64, 29, 74, 75]. Polgen [80] first proposed to generate SELinux policies based on the interaction patterns between different security contexts. Molly et al. propose to mine roles and identify redundant or unused roles from access usages logs with a learning approach [64, 29]. Sanders et al. conducted a series of work to generate least-privilege policies from audit logs with a counting-based model based on time window [73], and further propose Privilege Error Minimization Problem to minimize the over-privilege or under-privilege in RBAC [74] and ABAC models [75]. These works detect the over-privileged policy only *after* the over-privileged access is recorded in the access log. In contrast, SECLOG helping sysadmins to understand and fix unintended access-deny issues correctly in order to *prevent* insecure fixes and over-privileged user accesses.

4.4.3 Access-control Code Vulnerabilities

Vulnerabilities in access control code, aka bugs in authorization code, are dangerous as they allow attackers to bypass the entire protection. Various systems have been proposed to detect such bugs or enforce access control properties at runtime. There are quite some prior work on this topic, such as Nemesis [33], CLAMP [69], RESIN [110], Draco [93], SPACE [65], DetLogic [38] and others [90, 81]. SECLOG is complementary to theirs as they focus on software bugs, we focus on sysadmin mistakes (more specifically how to improve software logging to help sysadmins to reduce mistakes).

4.5 Acknowledgement

Chapter 4, in part, is a reprint of the material as it appears in the 32nd USENIX Security Symposium, 2023. Bingyu Shen, Tianyi Shan, Yuanyuan Zhou. "Improving Logging to Reduce Permission Over-Granting Mistakes". The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material under submission. Bingyu Shen, Tianyi Shan (co-first authors), Yuanyuan Zhou. "Multiview: Finding Blind Spots in Access-Deny Issues Diagnosis". The dissertation author was the primary investigator and author of this paper.

Chapter 5

Conclusion

This dissertation presents two approaches to improve the diagnosisbility of server systems. This dissertation presents SECLOG to automatically add missing access-deny log messages, and also enhance existing ones with relevant information to guide sysadmins to diagnose the access-deny issues. This dissertation also presents Multiview which takes a step further to automatically mutate the system configurations to explore possible directions and let each direction grant as few permissions as possible. Multiview provides a detailed diagnosis report to assist sysadmins during diagnosis.

SECLOG is a practical to help developers to improve access-deny log messages for sysadmins to fix access-deny issues correctly and safely. We first conducted a study on five large server programs, to understand the current status of access-deny logging practices. Based on our findings, we designed and implemented a tool called, SECLOG, that can help developers find missing log locations and identify relevant information at the log location. Our evaluation on ten server software and user study show that SECLOG is effective in helping developers to improve the quality of access-deny log messages and reducing the insecure fixes made by sysadmins. SECLOG's source code is publicly available at [15].

Multiview is a new diagnosis framework that can provide multiple directions 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 change impact of 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 45 out of 64 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.

5.1 Lessons Learned

In the end, I would like to talk about the lessons that I have working on the access control research which may be helpful for future directions.

Sysadmins as the target user. My first project during graduate school is the access control on mobile platforms, which directly faces the end users [77]. Many works try to help the end users configure the permissions on the mobile phones, in order to better protect users' privacy. However, when I work on the access control on the server platforms, things become very different — very few developers or academics care about the special software.

The server software has a very different target user group — system administrators. When the developers develop the software, they usually have false assumptions on sysadmins. They often assume that the sysadmins have very good knowledge of the server and provide little transparency for them (e.g., poor logging and documentation). Besides, many developers care more about the features, not considering enough for the security. Not even to mention the tools for detecting or diagnosing the access control issues. This dissertation aims to help sysadmins make configuration changes to resolve the access-deny issue. Future works can focus on detecting

the over-granted access during the configuration changes or access control auditing and forensics.

Evolving security goals. One assumption that researchers made is that the sysadmins clearly knows what is the desired security goal, which is the ground truth for verification or validation [105]. However, it is pretty hard to maintain a clear security goal in a constantly evolving environment even for system professionals. This may because of multiple factors, like the software does not support logging the events of configuration changes, or the miscommunication between engineers and management, or simply the previous sysadmins leave the job without enough documentation. Some of the existing works propose prototypes to model the access control policy with domain-specific languages which are hard to maintain for sysadmins who have little experience in formal models, or mine the security goals based on the access history which may be inaccurate and hard to verify. Therefore, future works may work on systems with ability to automatically construct or maintain the security goals based on configuration change events, which can be used for both auditing and forensics.

Bibliography

[1] Apache httpd web server. https://httpd.apache.org/.

[2] Cherokee web server. http://cherokee-project.com/.

[3] HAProxy. https://www.haproxy.com/.

[4] Mini SQL (mSQL). https://hughestech.com.au/products/msql/.

[5] NFS Ganesha File Server. https://fedoraproject.org/wiki/Changes/NFSGanesha.

[6] Postfix. http://www.postfix.org/.

[7] PostgreSQL. https://www.postgresql.org/.

[8] PostgreSQL official docs. https://www.postgresql.org/docs/12.

[9] ProFTPD. http://www.proftpd.org/.

[10] Redis. https://redis.io/.

[11] Vsftpd. https://security.appspot.com/vsftpd.html.

[12] vsftpd passive mode refused. https://serverfault.com/questions/344540/vsftpd-error-500-oops-vsf-sysutil-bind.

[13] vsftpd problem: 500 OOPS: vsf_sysutil_bind. https://www.linuxquestions.org/questions/linux-server-73/vsftpd-problem-500-oops-vsf_sysutil_bind-675699/.

[14] WebDAV getting 403 error when attempt to upload. https://serverfault.com/questions/20169/webdav-on-centos-getting-403-error-when-attempt-to-upload.

[15] SecLog's source code repo (anonymized for double-blind review). https://anonymous.4open.science/r/AceInstrument-F4BA/, 2022.

[16] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. Compilers, principles, techniques. *Addison wesley*, 7(8):9, 1986.

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

[18] Mona Attariyan, Michael Chow, and Jason Flinn. 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)*, pages 307–320, 2012.

[19] Mona Attariyan and Jason Flinn. Using causality to diagnose configuration bugs. In *USENIX Annual Technical Conference*, pages 281–286, 2008.

[20] Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *OSDI*, volume 10, pages 1–14, 2010.

[21] John Barkley. Comparing simple role based access control models and access control lists. In *Proceedings of the second ACM workshop on Role-based access control*, pages 127–132, 1997.

[22] Bradley Barth. Accentuate the negative: Accenture exposes data related to its enterprise cloud platform. https://www.scmagazine.com/home/security-news/data-breach/accentuate-the-negative-accenture-exposes-data-related-to-its-enterprise-cloud-platform/, Oct. 2017.

[23] Lujo Bauer, Lorrie Faith Cranor, Robert W Reeder, Michael K Reiter, and Kami Vaniea. Real life challenges in access-control management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 899–908, 2009.

[24] Lujo Bauer, Scott Garriss, and Michael K Reiter. Detecting and resolving policy misconfigurations in access-control systems. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):1–28, 2011.

[25] Matthias Beckerle and Leonardo A Martucci. Formal definitions for usable access control rule sets from goals to metrics. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, pages 1–11, 2013.

[26] Joel Brenner. Iso 27001 risk management and compliance. *Risk management*, 54(1):24–29, 2007.

[27] Seraphin Calo, Dinesh Verma, Supriyo Chakraborty, Elisa Bertino, Emil Lupu, and Gregory Cirincione. Self-generation of access control policies. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pages 39–47, 2018.

[28] Miguel Castro, Manuel Costa, and Jean-Philippe Martin. Better bug reporting with better privacy. *ACM SIGOPS Operating Systems Review*, 42(2):319–328, 2008.

[29] Suresh Chari, Ian Molloy, Youngja Park, and Wilfried Teiken. Ensuring continuous compliance through reconciling policy with usage. In *Proceedings of the 18th ACM symposium on Access control models and technologies*, pages 49–60, 2013.

[30] Yi Fei Chen. Usability of the access control system for openldap. Master's thesis, University of Waterloo, 2019.

[31] Andrea Continella, Mario Polino, Marcello Pogliani, and Stefano Zanero. There's a hole in that bucket! a large-scale analysis of misconfigured s3 buckets. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 702–711, 2018.

[32] Antonio Corrad, Rebecca Montanari, and Daniela Tibaldi. Context-based access control management in ubiquitous environments. In *Third IEEE International Symposium on Network Computing and Applications, 2004.(NCA 2004). Proceedings.*, pages 253–260. IEEE, 2004.

[33] Michael Dalton, Christos Kozyrakis, and Nickolai Zeldovich. Nemesis: Preventing authentication & [and] access control vulnerabilities in web applications. In *18th USENIX Security Symposium*, 2009.

[34] Tathagata Das, Ranjita Bhagwan, and Prasad Naldurg. Baaz: A system for detecting access control misconfigurations. In *USENIX Security Symposium*, pages 161–176, 2010.

[35] Jessica Davis. 63,500 patient records breached by New York provider's misconfigured database. `https://www.healthcareitnews.com/news/63500-patient-records-breached-new-york-providers-misconfigured-database`, Apr. 2018.

[36] Jessica Davis. Long Island provider exposes data of 42,000 patients in misconfigured database. `https://www.healthcareitnews.com/news/long-island-provider-exposes-data-42000-patients-misconfigured-database`, Mar. 2018.

[37] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

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

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

[40] BOB DIACHENKO. Document Management Company Left Credit Reports Online. `https://securitydiscovery.com/document-management-company-leaks-data-online/`, 2019.

[41] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. Investigating system operators' perspective on security misconfigurations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1272–1289, 2018.

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

[43] Sascha Fahl, Yasemin Acar, Henning Perl, and Matthew Smith. Why eve and mallory (also) love webmasters: a study on the root causes of ssl misconfigurations. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pages 507–512, 2014.

[44] David Ferraiolo, D Richard Kuhn, and Ramaswamy Chandramouli. *Role-based access control*. Artech house, 2003.

[45] Kathi Fisler, Shriram Krishnamurthi, Leo A Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering*, pages 196–205, 2005.

[46] Eric Freudenthal, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. drbac: distributed role-based access control for dynamic coalition environments. In *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 411–420. IEEE, 2002.

[47] J Frields. National industrial security program. operating manual supplement. Technical report, DEPARTMENT OF DEFENSE WASHINGTON DC, 1995.

[48] Mansura Habiba, Md Rafiqul Islam, and ABM Shawkat Ali. Access control management for cloud. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 485–492. IEEE, 2013.

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

[50] Pinjia He, Zhuangbin Chen, Shilin He, and Michael R Lyu. Characterizing the natural language descriptions in software logging statements. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 178–189, 2018.

[51] Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

[52] 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.

[53] Pooya Jaferian, Hootan Rashtian, and Konstantin Beznosov. To authorize or not authorize: helping users review access policies in organizations. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 301–320, 2014.

[54] Karthick Jayaraman, Vijay Ganesh, Mahesh Tripunitara, Martin Rinard, and Steve Chapin. Automatic error finding in access-control policies. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 163–174, 2011.

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

[56] Butler W Lampson. Computer security in the real world. *Computer*, 37(6):37–46, 2004.

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

[58] Frank Li, Lisa Rogers, Arunesh Mathur, Nathan Malkin, and Marshini Chetty. Keepers of the machines: Examining how system administrators manage software updates for multiple machines. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, 2019.

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

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

[61] Apache Official Manual. Apache configuration sections. https://httpd.apache.org/docs/2.4/sections.html, 2022.

[62] Evan Martin and Tao Xie. 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)*, pages 5–5. IEEE, 2007.

[63] Evan Martin and Tao Xie. A fault model and mutation testing of access control policies. In *Proceedings of the 16th international conference on World Wide Web*, pages 667–676, 2007.

[64] Ian Molloy, Youngja Park, and Suresh Chari. Generative models for access control policies: applications to role mining over logs with attribution. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 45–56, 2012.

[65] Joseph P Near and Daniel Jackson. Finding security bugs in web applications using a catalog of access control patterns. In *Proceedings of the 38th International Conference on Software Engineering*, pages 947–958, 2016.

[66] Qun Ni, Jorge Lobo, Seraphin Calo, Pankaj Rohatgi, and Elisa Bertino. Automating role-based provisioning by learning from examples. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 75–84, 2009.

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

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

[69] Bryan Parno, Jonathan M McCune, Dan Wendlandt, David G Andersen, and Adrian Perrig. Clamp: Practical prevention of large-scale data leaks. In *2009 30th IEEE Symposium on Security and Privacy*, pages 154–169. IEEE, 2009.

[70] Lili Qiu, Yin Zhang, Feng Wang, Mi Kyung, and Han Ratul Mahajan. Trusted computer system evaluation criteria. In *National Computer Security Center*. Citeseer, 1985.

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

[72] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *ACM Transactions on Storage (TOS)*, 9(3):1–32, 2013.

[73] Matthew Sanders and Chuan Yue. Automated least privileges in cloud-based web services. In *Proceedings of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, pages 1–6, 2017.

[74] Matthew W Sanders and Chuan Yue. Minimizing privilege assignment errors in cloud services. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 2–12, 2018.

[75] Matthew W Sanders and Chuan Yue. Mining least privilege attribute based access control policies. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 404–416, 2019.

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

[77] Bingyu Shen, Lili Wei, Chengcheng Xiang, Yudong Wu, Mingyao Shen, Yuanyuan Zhou, and Xinxin Jin. Can systems explain permissions better? understanding users' misperceptions under smartphone runtime permission model. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 751–768, 2021.

[78] 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.

[79] Sara Sinclair and Sean W Smith. What's wrong with access control in the real world? *IEEE Security & Privacy*, 8(4):74–77, 2010.

[80] Brian T Sniffen, David R Harris, and John D Ramsdell. Guided policy generation for application authors. In *SELinux Symposium*, 2006.

[81] Sooel Son, Kathryn S McKinley, and Vitaly Shmatikov. Fix me up: Repairing access-control bugs in web applications. In *NDSS*, 2013.

[82] 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.

[83] Tom Spring. Voter Database Leak Exposes 154 Million Sensitive Records. https://threatpost.com/voter-database-leak-exposes-154-million-sensitive-records/118901/, Jun. 2016.

[84] 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.

[85] 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.

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

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

[88] Ya-Yunn Su, Mona Attariyan, and Jason Flinn. Autobash: improving configuration management with operating system causality analysis. *ACM SIGOPS Operating Systems Review*, 41(6):237–250, 2007.

[89] Ya-Yunn Su and Jason Flinn. Automatically generating predicates and solutions for configuration troubleshooting. In *USENIX Annual Technical Conference*, 2009.

[90] Fangqi Sun, Liang Xu, and Zhendong Su. Static detection of access control vulnerabilities in web applications. In *USENIX Security Symposium*, volume 64, 2011.

[91] 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.

[92] Kunal Taneja, Mark Grechanik, Rayid Ghani, and Tao Xie. 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*, pages 201–211, 2011.

[93] Guliz Seray Tuncay, Soteris Demetriou, and Carl A Gunter. Draco: A system for uniform and fine-grained access control for web code on android. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 104–115, 2016.

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

[95] Biometric Update. 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.

[96] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009. http://www.globule.org/publi/WWADH_comnet2009.html.

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

[98] Artem Voronkov. *Usability of Firewall Configuration: Making the Life of System Administrators Easier*. PhD thesis, Karlstads universitet, 2020.

[99] Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 735–737, 2010.

[100] Helen J Wang, John C Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang. Automatic misconfiguration troubleshooting with peerpressure. In *OSDI*, volume 4, pages 245–257, 2004.

[101] Hua Wang, Yanchun Zhang, and Jinli Cao. Access control management for ubiquitous computing. *Future Generation Computer Systems*, 24(8):870–878, 2008.

[102] Rui Wang, XiaoFeng Wang, and Zhuowei Li. Panalyst: Privacy-aware remote error analysis on commodity software. In *USENIX Security Symposium*, pages 291–306, 2008.

[103] Yi-Min Wang, Chad Verbowski, John Dunagan, Yu Chen, Helen J Wang, Chun Yuan, and Zheng Zhang. Strider: A black-box, state-based approach to change and configuration management and support. *Science of Computer Programming*, 53(2):143–164, 2004.

[104] Andrew Whitaker, Richard S Cox, and Steven D Gribble. Configuration debugging as search: Finding the needle in the haystack. In *OSDI*, volume 4, pages 6–6, 2004.

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

[106] Chengcheng Xiang, Yudong Wu, Bingyu Shen, Mingyao Shen, Haochen Huang, Tianyin Xu, Yuanyuan Zhou, Cindy Moore, Xinxin Jin, and Tianwei Sheng. Towards continuous access control validation and forensics. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 113–129, 2019.

[107] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. Early detection of configuration errors to reduce failure damage. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 619–634, 2016.

[108] Tianyin Xu, Han Min Naing, Le Lu, and Yuanyuan Zhou. 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*, pages 348–361. ACM, 2017.

[109] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 244–259, 2013.

[110] Alexander Yip, Xi Wang, Nickolai Zeldovich, and M Frans Kaashoek. Improving application security with data flow assertions. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 291–304, 2009.

[111] Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U Jain, and Michael Stumm. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 249–265, 2014.

[112] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. Be conservative: Enhancing failure diagnosis with proactive logging. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 293–306, 2012.

[113] Ding Yuan, Yinglian Xie, Rina Panigrahy, Junfeng Yang, Chad Verbowski, and Arunvijay Kumar. Context-based online configuration-error detection. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, pages 28–28. USENIX Association, 2011.

[114] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. Improving software diagnosability via log enhancement. *ACM Transactions on Computer Systems (TOCS)*, 30(1):1–28, 2012.

[115] 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.

[116] Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. 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*, pages 687–700, 2014.

[117] Sai Zhang and Michael D Ernst. Automated diagnosis of software configuration errors. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 312–321. IEEE, 2013.

[118] Sai Zhang and Michael D Ernst. Which configuration option should I change? In *Proceedings of the 36th International Conference on Software Engineering*, pages 152–163, 2014.

[119] Sai Zhang and Michael D. Ernst. Proactive detection of inadequate diagnostic messages for software configuration errors. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, page 12–23, New York, NY, USA, 2015. Association for Computing Machinery.

[120] Xu Zhao, Kirk Rodrigues, Yu Luo, Michael Stumm, Ding Yuan, and Yuanyuan Zhou. Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 565–581, 2017.

[121] Rui Zhou, Mohammad Hamdaqa, Haipeng Cai, and Abdelwahab Hamou-Lhadj. Mobilogleak: A preliminary study on data leakage caused by poor logging practices. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 577–581. IEEE, 2020.