How Do System Administrators Resolve Access-Denied Issues in the Real World?

Tianyin Xu, Han Min Naing, Le Lu, and Yuanyuan Zhou
University of California, San Diego

ABSTRACT

The efficacy of access control largely depends on how system administrators (sysadmins) resolve access-denied issues. A correct resolution should only permit the expected access, while maintaining the protection against illegal access. However, anecdotal evidence suggests that correct resolutions are occasional—sysadmins often grant too much access (known as security misconfigurations) to allow the denied access, posing severe security risks. This paper presents a quantitative study on real-world practices of resolving access-denied issues, with a particular focus on how and why security misconfigurations are introduced during problem solving. We characterize the real-world security misconfigurations introduced in the field, and show that many of these misconfigurations were the results of trial-and-error practices commonly adopted by sysadmins to work around access denials. We argue that the lack of adequate feedback information is one fundamental reason that prevents sysadmins from developing precise understanding and thus induces trial and error. Our study on access-denied messages shows that many of today's software systems miss the opportunities for providing adequate feedback information, imposing unnecessary obstacles to correct resolutions.

ACM Classification Keywords

H.1.2 User/Machine Systems: Human factors; D.4.6 Security and protection: Access controls; K.6.5 Management of computing and information systems: Security and Protection

Author Keywords

Security; access control; configuration; log messages

INTRODUCTION

Motivation

> "Access control doesn't work—40 years of experience says so. The basic problem [is]: its job is to say 'No.' This stops people from doing their work, and then they relax the access control. [There is] usually too much [access], but no one notices until there's a disaster." [31]
>
> —Butler Lampson, Perspectives on Security (2015)

CHI 2017, May 6-11, 2017, Denver, CO, USA.

DOI: http://dx.doi.org/10.1145/3025453.3025999

Access control is constantly reported to be "profoundly broken" in real-world system administration practices [33, 45, 57]. The blame is mainly placed on its side effects, *access-denied issues* where expected access is falsely denied. Anecdotal evidence suggests that in order to resolve such issues, system administrators (sysadmins) often grant too much access, known as *security misconfigurations*. Today, misconfigurations have become a major cause of security failures such as data thefts and system compromises [24, 45, 67]. For example, the misconfigurations of a database server at an insurance company allowed its data to be publicly accessible, causing massive breach of customers' account information in 2014 [21].

The challenge of correctly resolving access-denied issues lies in the requirement of precise configurations—those only permit expected access while maintaining the protection against illegal access. This is non-trivial in modern software systems, where access to resources is typically restricted by complex, heterogeneous access-control configurations at multiple layers. For example, to access a web page hosted on a web server, an HTTP request is typically checked through access control of firewalls, web authentication/authorization, file permissions, and OS kernel security modules. Correct resolutions need to understand not only already complex access-control configurations of each system component, but also their interactions.

Currently, resolving access-denied issues is mainly through human-centered problem-solving, in which sysadmins reason about missing permissions and devise resolutions. Insecure resolutions are notoriously hard to detect, as security misconfigurations do not manifest through symptoms of failures or anomalies [71]. A few existing tools focus on either predefined bad values [19, 37] or inconsistencies among settings [8, 9, 15, 78]. However, as noted in [15], given frequent configuration changes and the ad-hoc, one-off nature, it is very difficult for automated tools to deduce the exact and complete list of security misconfigurations. The correctness of access-control configurations mainly relies on sysadmins' practices.

Equipping sysadmins to correctly resolve access-denied issues requires the understanding of their problem-solving practices: how do sysadmins resolve access-denied issues? what are the obstacles to correct resolutions? how are security misconfigurations introduced? Without answering these fundamental questions, it is difficult for research and development efforts to focus on effective directions and build practical solutions. Unfortunately, few studies have provided answers. Sometimes, we simply lay the blame on sysadmins' attitude and hope for better education, which rarely worked in the past [31, 40].

Contribution

This paper presents a quantitative study on real-world practices of resolving access-denied issues, with a particular focus on how and why security misconfigurations were introduced during problem-solving. The objective is to identify and understand the fundamental obstacles faced by sysadmins, as well as the implications for better design of software systems and tool support for access-control configurations.

To this end, we study 486 real-world access-denied issues from four widely-used software systems collected from mailing lists and web forums. For each case, we examine the symptom, root cause, and problem-solving practices. Specially, we deduce the security impact of the configurations introduced by sysadmins in the field. We flag security misconfigurations if the new settings permit too much access.

The study shows that security misconfigurations were commonly introduced as workarounds for access-denied issues—they occurred in 38.1% of the studied cases in different software systems. The most common security misconfigurations were completely disabling access-control modules, voiding all the protection against malice. Ironically, 49.7% of the security misconfigurations introduced in the field failed to work around the issues, mostly because the misconfigurations, as blind attempts, were irrelevant to root causes of the denials.

One contributing factor of such significant prevalence is that security misconfigurations were frequently suggested as potential resolutions on web forums and mailing lists; 48.6% of these suggested security misconfigurations were adopted by sysadmins who were eager to allow the denied access. We observe that voting-based moderation, though commonly used by web forums, is ineffective in suppressing insecure suggestions.

We show that many security misconfigurations were the results of trial-and-error practices commonly adopted for problem-solving: sysadmins tried to allow the denied access by relaxing certain access-control configurations, without precise understanding of root causes—the missing permissions. We show that 70.3% of these misconfigurations were associated with sysadmins' futile configuration attempts that were irrelevant to the root causes. These useless configurations included working on wrong security modules and wrong parameters.

We argue that the lack of adequate feedback is one fundamental reason that prevents sysadmins from developing precise understanding and thus induces trial and error. Typically, the feedback information (mainly via log messages) is too coarse-grained or acts as generic notifications, which is unhelpful for reasoning out the missing permissions.

To quantitatively understand existing feedback information, we study access-denied messages in six mature, widely-used software projects. We show that many of these software systems miss the opportunities for providing specific, precise, and complete feedback to assist correct resolutions. First, access denials occasionally have specific messages: many of them come with generic notifications. Second, root-cause configurations are rarely included in message content, even when the software holds precise knowledge. Third, many messages lack critical details such as subject, access type, and object.

MySQL server fails to start with the following error messages:

```
[ERROR] Can't find file: './mysql/plugin.frm' (errno: 13)
[ERROR] Can't open the mysql.plugin table. Please run
        mysql_upgrade to create it.
```

attempts:	suggestions:
1. restore `plugin.frm` ("This file does exist and is readable!"); [useless]	1. upgrade MySQL version (run `mysql_upgrade`); [useless]
2. restart the server; [useless]	2. check if startup scripts override the configurations. [useless]
3. change all MySQL system files to be world readable, writable, and executable. [harmful] [useless]	

root cause & correct resolution:

The file access was denied by AppArmor (a Linux kernel security module) and returned `errno 13`. The resolution is to configure the AppArmor policies to allow MySQL to access the file.

(a) Access denied by an OS security module

Apache web server rejected legitimate access (403 forbidden) with the following error message:

```
[authz_core:error][client:127.0.0.1:38873] AH01630:
client denied by server configuration: /home/remix/
```

attempts:	suggestions:
1. set "`allow from all`"; [useless]	1. check directory permission; [useless]
2. allow IPs in `/etc/hosts`; [useless]	2. change `/home/remix` to be world executable. [harmful] [useless]
3. restart Apache via new user `remix` (changed from `nobody`); [useless]	
4. change `/home/remix` to be world executable. [harmful] [useless]	

adopted

root cause & correct resolution:

The client access is denied by Apache's internal authorization module (mod_access_compat). To resolve the issue, use the `require` configuration parameter to grant necessary access.

(b) Access denied by Apache's internal configurations

Figure 1. Real-world examples of resolving access-denied issues that introduced security misconfigurations in MySQL and Apache [52, 64].

We hope these results to raise the awareness of the importance of assisting access-denied issues, and motivate better feedback and tool support towards usable access control.

Real-world Examples

Figure 1 illustrates two representative real-world examples of resolving access-denied issues that introduced security misconfigurations (selected from our study). In both cases, the sysadmins applied the trial-and-error approach for problem solving—trying various configurations to allow the denied access, without understanding of the root-causes of the denials. In Figure 1(a), the sysadmin failed to reason out the root cause to be the configurations of AppArmor (a Linux kernel security module). Even after the sysadmin recursively configured the most permissive file permissions for MySQL's system files, the expected access was still denied. In Figure 1(b), the access was denied by Apache's internal authorization configurations. However, most of the attempted and suggested configurations targeted on wrong access-control modules (e.g., network-level access control and file permissions) and thus were useless.

Note that the feedback information given by the systems was inadequate in both of the examples. In Figure 1(a), the message content did not even clearly point out access control as

the root cause (except the `errno`). In fact, `errno 13`[1] merely means "`permission denied`" and does not reveal any specific information. In Figure 1(b), the error message "`client denied by server configuration`" was too generic to be helpful—it did not pinpoint which security modules and configuration parameters denied the access. Due to the large, complex configuration space, it is not surprising that the sysadmins failed to shot the root-cause configurations in the dark via trial and error.

METHODOLOGY

We study 486 randomly sampled, access-denied cases reported on four software systems collected from mailing lists and sysadmin-oriented web forums. Each case starts with the descriptions of a real-world issue, followed by online discussions and collaborative problem-solving efforts. In order to resolve the reported issues timely, sysadmins were willing (and required [41]) to describe detailed information in the field, including the software versions, system feedback, configurations related to access control, diagnostic/fixing attempts, etc. This provides the opportunity for understanding the reported issues: their root causes, the field practices of problem-solving, and the security impact of the attempted configurations.

For each case considered, two inspectors independently investigated the case report and recorded the information of interest, including the root-cause configurations, observed system feedback, configurations introduced during problem-solving, suggestions of resolutions, etc. The information was recorded in specially designed spreadsheets with detailed written classifications. The two inspectors then compared their results with each other until consensus was reached. For some tough cases, we checked the source code and reproduced the issues to ensure correct understanding. We observed inconsistent results in less than 4% of cases, with early setups of the definitions, categorization, and examples of security misconfigurations. Most inconsistencies were due to one inspector missing security risks of the new configurations; after the other inspector explained the risks, consensus was reached.

Target users: system administrators

Our study focuses on system administrators (sysadmins), a special type of software users [4]. We follow the definition in [5], where *sysadmins* are broadly referred to as "*those who use their technical, social, and organization skills to architect, configure, administer, and maintain computer systems, including operating systems, networks, security systems, databases, web servers, and applications*." Note that the definition is not limited to full-time IT professionals; thus we also include sysadmins of other occupations (e.g., developers) who managed their own infrastructure consist of the studied software.

Compared with ordinary users, sysadmins are known to be technical inclined [5, 28]. Specially, they are expected to take an active role in managing system security, as one primary concern of the systems they manage. Moreover, unlike end users whose access is restricted to avoid disclosure of sensitive information (e.g., internal access-control policies), sysadmins are the policy makers and implementers, with highest privileges.

[1]Linux's manpage lists more than three cases that could result in `errno 13` (`EACCES`) and does not include kernel security modules.

Software	Description	Lang.	Studied cases	
			Period	Total (#)
Apache	Web server	C	2001–2016	126
MySQL	Database	C++	1999–2016	117
Hadoop	Data processing	Java	2009–2016	101
CentOS	Linux distribution	Multiple	2005–2016	142

Table 1. The studied software systems, and the period; total number of real-world access-denied cases studied for each system in this paper.

Also, it is known that today's sysadmins have diverse backgrounds and skill sets—"*there is no single path to becoming a system administrator [...] many system administrators enter the field by self-taught*" [73]. In fact, even within one organization, security administration is often distributed across multiple sysadmins, each being responsible for different aspects of it, e.g., web, databases, operating systems [10].

Target software: server and operating systems

The studied issues were reported on four open-source software systems: Apache, MySQL, Hadoop, and CentOS[2] (c.f., Table 1). We selected these systems for the following reasons. First, all these systems are server and operating systems that are usually managed by sysadmins instead of ordinary end users. Second, all the systems require intensive access-control configuration, as security is one primary aspect of the system administration—the impact of security misconfigurations is severe. Third, all these software systems are mature, actively maintained, and widely used; their access control represents the state-of-the-art design and implementation. Fourth, we intend to cover diverse types of systems in terms of functionalities, programming languages, and system architectures.

Data collection

We collected real-world access-denied cases from the official mailing lists of the target software systems [2, 13, 23, 39] and sysadmin-oriented web forums including ServerFault [46] (for all the studied systems), Database Administrator [16] (only for MySQL), StackOverflow [58] (only for Hadoop[3]), and CentOS official forum [12]. As each studied system had hundreds of thousands of cases, in order to efficiently collect relevant cases, we used keywords related to access denials, including "access denied", "access control", "security", "permission", "privilege", and their variations. From the thousands of cases that contained at least one keyword, we *randomly* selected a thousand case reports with clear and detailed descriptions. Then, we manually examined them to make sure that the cases were really access-denied issues. We only selected *resolved* cases (those confirmed by the original reporters) to ensure correct understanding of the root causes and resolutions. To ensure the representativeness of sysadmins, we collected *one case per reporter* (identified by user IDs and emails).

[2]CentOS is selected over other OS distributions (e.g., Windows and Mac OS) because it is an enterprise-class distribution commonly used as servers. Few CentOS cases are about kernel configurations; most are related to configurations of system utilities (e.g., `mount`) and installed program packages. There are 64 unique programs appearing in the CentOS cases, providing a large, diverse OS software set.

[3]StackOverflow is the primary forum for Hadoop administration (likely because many Hadoop sysadmins are developers).

Table 1 shows the period and number of cases included in this study for the target software. All the cases are public available at: https://github.com/tianyin/access_denied_datasets.

Threats to Validity

As with all characteristic studies, there is an inherent risk that our study may be specific to the target software systems. While we cannot establish representativeness categorically, we have taken care to select diverse software projects that are mature and widely used (Table 1). Note that our study only focuses on system software; we do not intend to draw any conclusion about other types of software such as desktop and mobile apps.

Another concern is the representativeness of the studied access-denied issues. First, novice sysadmins are more likely to report problems, compared with experts. Unfortunately, it is hard to objectively differentiate between novices and experts. In fact, with new and major revisions of software being deployed in the field, there are always novice sysadmins. Second, we only collect cases that were reported by sysadmins and resolved at the time of the study. It is possible that the non-reported and non-resolved cases might have different characteristics.

Another potential source of bias is that the case reports may not record every single configuration action done by the sysadmins. This may affect the statistics of security misconfigurations and useless configurations. According to our experience, most of the sysadmins who reported the access-denied issues described their problem-solving efforts in details to help pin down the root causes efficiently. Therefore, we believe that the reported numbers well represent the general characteristics.

Still, we remind our readers that the results and findings should be taken with the target software and data sources in mind. At the same time, we urge the readers to focus more on the overall trends and implications rather than the absolute numbers.

PROFILES OF ACCESS-DENIED ISSUES

Access-control configurations

Both application- and OS-level access-control configurations are included in the studied cases. Many of the studied software systems implement access-control mechanisms to protect application-specific objects. For example, MySQL provides 32 different privileges for defining access to different objects such as database, table, columns, view, etc. Apache has 15 different authentication and authorization modules which can be used in combination by `RequireAny` and `RequireAll` directives. Besides their own access control, the software programs interact with OS-level access control when requesting OS resources. The common OS-level access control in our study includes file permissions, SELinux, and firewalls (e.g., iptables).

Notably, all access-control configurations in our study were under discretionary or mandatory models. None of the cases used role-based access control (RBAC). SELinux is perhaps the only one that supports RBAC, but in most Linux distributions it is not enforced by default and is rarely enabled.

Patterns of denied access

The denied access falls in two patterns: (1) the software program denied incoming access (e.g., issued by end users' SQL queries) based on its internal access-control configurations; and (2) the software program tried to access OS resources (e.g., files) but was denied by OS-level access controls, e.g., when the Apache web server was fetching HTML files to serve users' HTTP requests. Both the first and second patterns contribute to significant percentages (42.6% and 57.4% respectively).

Unlike application-level access-control configurations which was specific to each application software, 88.2% of the OS-level access-denied issues were related to file access; 67.9% of the denied file access was not caused by the basic three permission triads (denoted as `rwxrwxrwx`) of the accessed files. One common mistake was lacking search permissions on one parent directory of the accessed file (many sysadmins only checked the permissions of the accessed files). Besides file permissions, in many cases the access was denied by other OS security modules (e.g., SELinux and AppArmor) or security policies (e.g., Windows 7 disallows application-level programs to create symbolic links by default [60]).

Log messages as main feedback information

Sysadmins mainly rely on the software's log messages to understand the root causes of denied access. Among all the 486 case reports, 87.0% of the sysadmins examined the logs of the studied software and included them in the case reports (including log files and console log messages); 91.3% of them extracted the log messages specific to the denied access from the entire logs. Besides the logs of the studied software, 5.6% of the sysadmins also examined OS logs for certain types of issues. A common case was checking `audit.log` for SELinux-related issues. Guided by the log messages, most of the sysadmins also examined access-control configurations in configuration files or returned by status-checking commands (such as "`ls -lZ`" for file permissions and security contexts).

Only eight (1.6%) sysadmins checked the software's source code during problem solving (all of them were from Hadoop cases); none used any debuggers (e.g., `GDB`) to examine execution. This again shows that sysadmins, even with the developer occupation, are *users* of the systems they manage. They mainly rely on the system's feedback information instead of trying to understand the implementation or runtime execution.

Tool support for problem-solving

In the studied access-denied cases, the problem-solving was mostly through manual configuration efforts with few automated tooling support. Very few of the studied software applications provided tool support for automatically pinpointing the missing permissions or suggesting resolutions—sysadmins changed access-control configurations by editing configuration files or using commands (e.g., `GRANT` in MySQL). Similarly, all the file permissions were done with Unix shell commands, such as `chmod`, `chown`, `setfacl`, etc. The only tool support observed in our study was from SELinux (a Linux kernel security module). In certain cases related to SELinux, the sysadmins used `sealert` and `audit2allow` to automatically troubleshoot and generate SELinux policy configurations, which could permit the denied access recorded in audit logs. Note that utilities like `audit2allow` are not the panacea for SELinux denial issues [68]—the common resolutions were still manually editing security labels and boolean parameters.

SECURITY MISCONFIGURATIONS AS INCORRECT RESOLUTIONS

To understand real-world security misconfigurations, we deduce security impact of the access-control configurations introduced by the sysadmin in the studied cases. We determine security misconfigurations if the introduced configurations granted too much access—more than what is necessary and should be permitted. Note that we do *not* measure the absolute security of the configuration values; instead, we leverage the original configurations as the baseline to check if the newly introduced configurations produced appropriate access.

Security misconfigurations are commonly introduced

Our study reveals significant prevalence of security misconfigurations in the field—they were introduced as resolutions among in 38.1% of the studied access-denied cases across different software systems (c.f., Table 2). These results advocate for the importance of studying the usability of access control from the aspects of access-denied issues.

Essentially, access-control configurations are not a one-time effort, but need constant changes to adapt the dynamic nature of information sharing. The enforcement of "security by default" is insufficient to maintain protection of access control in the long term, if sysadmins cannot resolve access-denied issues with correct, secure configurations. Unfortunately, our study shows that the real-world practices of resolving access-denied issues are prone to security misconfigurations.

Common types of security misconfigurations

We categorize the security misconfigurations into three types based on how the access-control configurations were incorrectly loosened. Table 3 gives the definitions of each type and provides real-world examples selected from our study. Table 4 shows the distribution of different types of security misconfigurtions among the studied cases containing misconfigurations.

Disabling the protection by access control of a specific security module was the most common type of security misconfigurations: 74.6% of the cases of security misconfigurations fall in this category. This is likely because disabling the entire module was considered as a quick way to work around the denials (compared with setting correct configurations).

The distribution of the other two types of security misconfigurations mainly depends on the access-control model of the studied software. For example, MySQL implements privilege-based access control—to access a database or a table, the database users need to be granted corresponding privileges. Therefore, over-granting privileges was the common security misconfigurations in MySQL. Apache, as a web server, intensively interacts with the filesystems to fetch files such as web pages, images, and Javascripts. Downgrading file permissions was a common misconfiguration to obtain file access.

Blind misconfigurations cannot permit the denied access

Ironically, 49.7% of the security misconfigurations failed to work around the reported access-denied issues: they were not only harmful (imposing security risks) but also useless. Such lose-lose situation was typically caused by security misconfigurations performed on wrong security modules or configuration parameters which were irrelevant to the root causes of the denials, as exemplified by Figures 1. We discuss the useless configurations in details in the later sections.

Software	# Cases incl. security misconfigurations		Total
	Performed	Suggested	
Apache	42.9% (54)	18.3% (23)	126
MySQL	32.5% (38)	20.5% (24)	117
Hadoop	36.6% (37)	15.8% (16)	101
CentOS	39.4% (56)	6.3% (9)	142
Total	38.1% (185)	14.8% (72)	486

Table 2. Prevalence of security misconfigurations. The percentage (number) of access-denied cases in which security misconfigurations were introduced by the sysadmins and suggested as potential resolutions.

Type	Definition & Example
Disable protection	**Def.:** Disable all the protection of the security module
	Ex.1: Set file permissions to `rwxrwxrwx` [49, 53]
	Ex.2: Disable OS security modules (e.g., SELinux) [55]
	Ex.3: Remove firewalls (e.g., ModSecurity) [50]
	Ex.4: Turn off application-level permission checks [59]
Over-grant privileges	**Def.:** Grant unnecessary privileges to the subject
	Ex.5: Grant users the access to all the database tables (`"GRANT ALL *.* TO $user"` in MySQL) [54]
	Ex.6: Add regular users in the administrator group [48]
Downgrade permissions	**Def.:** Downgrade permissions of the object unnecessarily
	Ex.7: Give up one type of file permissions (e.g., execution permission by `"chmod a+x $file"` [47])
	Ex.8: Allow any groups (*) to submit jobs to Hadoop only for allowing one specific user [62]

Table 3. Definitions and examples of different types of security misconfigurations that lead to more access than what is necessary and should be permitted.

Software	Disable protection	Over-grant privileges	Downgrade permissions	Total
Apache	66.7% (36)	9.3% (5)	35.2% (19)	54
MySQL	63.2% (24)	52.6% (20)	2.6% (1)	38
Hadoop	81.1% (30)	13.5% (5)	5.4% (2)	37
CentOS	85.7% (48)	3.6% (2)	12.5% (7)	56
Total	74.6% (138)	17.3% (32)	15.7% (29)	185

Table 4. Distribution of different types of security misconfigurations. The three types are defined in Table 3. Note: some cases introduced more than one types of security misconfigurations.

Security misconfigurations are frequently suggested

As shown in Table 2, among 14.8% of the studied cases security misconfigurations were suggested as potential resolutions on the web forums and mailing lists across the studied systems. Worse, 48.6% of these suggested security misconfigurations were adopted by sysadmins who were eager to allow the denied access on these systems. This partially explains the prevalence of security misconfigurations.

Unlike misconfigurations that result in functionality or performance issues, security misconfigurations are not manifested immediately; also their impact is hard to measure. It could be challenging to understand the security impact of the suggested resolutions, especially when the configurations are complex or unfamiliar to the sysadmins. This calls for future research to help sysadmins evaluate and validate the security of suggestions from untrusted sources.

Interestingly, the characteristics of insecure suggestions are quite different across web forums, as shown in Table 5. Among the studied cases, CentOS Forums and DBA had no security

Web sites	% Cases w/ misconfiguration suggestions (adoption rate of misconfigurations)
StackOverflow [58]	22.4% (76.8%)
Mailing lists [2, 13, 23, 39]	16.7% (14.4%)
ServerFault [46]	14.7% (60.5%)
DBA [16]	0.0% (0.0%)
CentOS Forums [13]	0.0% (0.0%)

Table 5. The percentage of cases where security misconfigurations were suggested (and their adoption rate) on different web sites.

misconfiguration suggestions, mainly because in these two sites the reported issues were answered and moderated by a small number of experts (e.g., the designated moderators in CentOS Forums) who were cautious in giving suggestions and were responsible for editing suggestions posted by other users.

On the other hand, we find that voting-based moderation [65], though commonly adopted by web forums, is ineffective in suppressing insecure suggestions. Among all the suggestions that contained security misconfigurations on ServerFault and StackOverflow, only 6.6% of them were down-voted, while 50.8% of them had up-votes.[4] Worse, 32.7% of these misconfiguration suggestions had at least two up-votes, i.e., these insecure suggestions had already influenced other sysadmins besides the reporter. Security misconfiguraions were up-voted because they can work around the denials, and thus considered useful. This again shows that completely relying on sysadmins to correctly evaluate security risks cannot be expected.

Security misconfigurations through trial and error

We further examine sysadmins' problem-solving practices and observe that many of the introduced security misconfigurations were the results of trial-and-error based problem-solving: sysadmins tried to allow the denied access by loosening certain access-control configurations, without precise understanding of the root causes—the missing permissions. In one case [63], the sysadmin commented on the workaround,

> "This was a terrible work around and I do not recommend it. ~~but~~ I changed the permission of the `/home/hadoop` directory to `777`[5] and it works. I couldn't find where `TokenizedMapper[.class file]` resided in my user file. [I know] This is a terrible option but I am accepting it in order to close the question."

Trial and error is featured by useless configuration efforts that are irrelevant to the root causes and thus cannot resolve the issues. In 70.3% of the studied cases that contained security misconfigurations, sysadmins attempted futile configurations, indicating that the sysadmins failed to reason out the precise root causes of the denials. The following comments describe the typical difficulties encountered by sysadmins during the trial-and-error process,

> "I'm totally stuck. I'm reading other guides and trying random configurations here and there, but I'm clueless."

[4]ServerFault and StackOverflow encourages their registered users to up or down vote the suggested solutions. Only votes from users with enough "reputation scores" are counted for the posted vote scores.

[5]A file permission of `777` (i.e., `rwxrwxrwx`) allows everyone in the system to read, write, and execute the file.

Software	# Cases incl. useless configs		Total
	Performed	Suggested	
Apache	50.8% (64)	48.4% (61)	126
MySQL	41.9% (49)	30.8% (36)	117
Hadoop	32.7% (33)	28.7% (29)	101
CentOS	54.2% (77)	40.8% (58)	142
Total	45.9% (223)	37.9% (184)	486

Table 6. The percentage (number) of studied cases that include useless configurations attempted by sysadmins and suggested by the other users.

Type	Definition & Example
Blind restore	**Def.:** Blindly attempt irrelevant things
	Ex.1: The sysadmin rebooted the system or reinstalled the software [53]
Wrong problem	**Def.:** Fail to recognize access control as the root cause
	Ex.2: The sysadmin was confused by the log messages showing `ClassNotFoundException`, but the root cause was insufficient file permissions [61].
Wrong module	**Def.:** Work on wrong security modules
	Ex.3: The sysadmin changed filesystem permissions, but the file access was denied by SELinux [64]
Wrong param.	**Def.:** Work on wrong configuration parameters
	Ex.4: The sysadmin changed `SecRequestBodyLimit` setting (in `ModSecurity`), but the request was denied by `SecRequestBodyAccess` [51]
Wrong setting	**Def.:** Set wrong access-control configuration values
	Ex.5: The sysadmin granted the `ALL` privilege to the database user, but "`LOAD DATA`" required the `FILE` privilege (`ALL` does not include `FILE`) [17]

Table 7. Definitions and examples of different types of useless configurations that failed to resolve the access-denied issues.

We shall point out that *trail and error is a terrible problem-solving practice for any security issues* because of its solution-oriented and non-optimal nature—trial and error makes no attempt to discover why a solution works (merely that it is a solution); trial and error only attempts to find a working solution, rather than the best (secure) one. This explains why trial and error often leads to security misconfigurations. To make the matter worse, many of the security-misconfiguration "trials" are not easy to revert—the new configurations could overwrite previous settings or permanently change the system states, e.g., recursively `chmod` a directory in Figure 1(a).

Unfortunately, trial and error appears to be a common problem-solving practice adopted by sysadmins for access-denied issues. As shown in Table 6, a significant percentage (45.9%) of *all* the studied access-denied cases included useless configuration attempts. In a similar percentage (37.9%) of the cases, useless configurations were suggested as potential resolutions by the other users on mailing list and web forums. This shows that precisely understanding the root causes of access-denied issues is a common difficulty.

There are other security misconfigurations due to sysadmins' being incapable of access-control configurations. Typically, the security modules were originally installed and configured by other sysadmins or by default. In one of such cases [50], the sysadmin disabled ModSecurity (a web application firewall) and commented: "*There seems to be a frustratingly large amount of ModSecurity documentation, but I don't have weeks to read through hundreds of documentation [pages] just to find out how to make it play nice with SVN.*"

Software	Blind restore	Wrong problem	Wrong module	Wrong parameter	Wrong setting	Total
Apache	7.8% (5)	7.8% (5)	48.4% (31)	34.4% (22)	9.4% (6)	64
MySQL	10.2% (5)	24.5% (12)	57.1% (28)	12.2% (6)	14.3% (7)	49
Hadoop	9.1% (3)	12.1% (4)	48.5% (16)	42.4% (14)	6.1% (2)	33
CentOS	5.2% (4)	15.6% (12)	49.4% (38)	28.6% (22)	22.1% (17)	77
Total	7.6% (17)	14.8% (33)	50.7% (113)	28.7% (64)	14.3% (32)	223

Table 8. Distribution of different types of useless configurations (defined in Table 7). Note: some cases had more than one types of useless configurations.

Useless configuration attempts

We categorize the useless configurations based on the error patterns. Table 7 describes the definitions and gives examples of each category. Note that the categories are *exclusive*—wrong configuration parameters means correct security modules and wrong settings indicates correct parameters. Each useless configuration belongs to only one category; but some cases included multiple different types of useless configurations.

Table 8 shows that working on wrong modules and wrong parameters contribute to most of the useless configurations. They were introduced in 50.7% and 28.7% of the studied cases, respectively. In comparison, wrong configuration values was less common. Moreover, in 14.8% of the cases, the sysadmins even did not realize the root causes were related to access control, but chased completely wrong directions, e.g., manipulating the Java classpath for the access-denied issues (Table 7: Ex2). The results show that sysadmins had difficulties in locating the root-cause modules and configuration parameters, not to mention the concrete settings.

Discussion

We argue that one fundamental reason behind the wide adoption of trial and error for problem-solving lies in the lack of adequate feedback information. Typically, the feedback information (mainly via log messages) is too coarse-grained or merely acts as generic notifications. This imposes significant obstacles for sysadmins to develop precise understanding of the root causes and to reason out the missing permissions. In the studied cases, it not uncommon to see sysadmins complaining the feedback messages, for example,

> "Apache's error logs simply say I don't have permission to access the file, [and] isn't telling me anything more useful, and my searches have all told to ensure the permissions are set correctly."

> " I don't even see a mention of the group in this exception message. Do group permissions just not work?"

We believe good feedback messages is essential to assist correct resolutions for access-denied issues. Designing good log messages and improving existing ones would require software developers and tool builders to carefully evaluate the message content. Different from debugging and auditing logs, access-denied messages should be designed from the perspectives of resolving access-denied issues with secure configurations.

INSPECTION OF LOG MESSAGES

To quantitatively understand the feedback information provided by today's software systems, we study access-denied log messages in the latest versions of six mature, widely-used system software projects (Apache, MySQL, Lighttpd, VSFTP, Hadoop, and HBase). The study is based on inspecting the source code via two steps: (1) identifying the program points that deny access in the source code of the studied software program, and the corresponding log messages; (2) evaluating whether the message content satisfies the design principles.

We include the two common access-denied patterns: (1) the software program denies incoming access based on its internal access-control configurations; and (2) the access to external resources (e.g., files) is denied by OS-level access-control configurations. For access to OS resources, we focus on *file* access which dominates external access-denied issues, as revealed in the profiles of access-denied issues.

Note that few work has studied access-denied log messages from the aspect of correctly resolving the issues (prior studies mainly focus on logs for auditing [1, 14] and failure diagnosis [26, 76, 77, 79]). Therefore, we start with presenting the methodology of identifying and evaluating the log messages.

Methodology

Identify access-denied log messages

We identify access-denied log messages by searching the program signatures of access-denied points in the source code; we then validate each point by inspecting related code.

Internal access denials. As different software programs have different access-control implementations, there is no common signature of access denials across software. Fortunately, within each software, denials are manifested through globally defined error code or exceptions. For example, in Apache the error code that would result in 403 Forbidden HTTP status includes `HTTP_FORBIDDEN`, `AUTHZ_DENIED`, and `AUTH_DENIED`; Hadoop throws `AccessControlException` when denying access. These error code and exceptions can be viewed as program signatures of access denials. Starting from the program points that assign the error code or throw the exceptions, we record the log messages specific to the denied access by tracking the propagation of the error code or exception.

File access denials. Access to files is through system/library APIs such as `open`, `fopen`, and `FileInputStream`, where error code (`EACCES`) and exceptions (`IOException`) would be returned for the denied access. We start with file APIs in `libc` and Java SDK. We only include APIs related to access control configurations, and exclude file read/write APIs. In our inspection, we observe that developers seldom check return values and `errno` of two access patterns: (1) deleting files created by them; (2) the file access is checked early (e.g., via `stat` to check accessibility and file types). We exclude these two patterns (including them would result in larger number of unlogged cases).

Most of the studied systems implement customized file APIs that wrap around basic system/library APIs with same semantics and error code, such as `apr_file_open`, and `my_open` for `open`

```
/* hadoop: hadoop-hdfs-project/.../server/namenode/FSPermissionChecker.java
   (commit 0398db5...48896fa) */
throw new AccessControlException(
-       "Permission denied");
+       "Permission denied. user="
+       + user + " is not the owner of inode=" + inode
);
```

(a) The specific and precise principles: the message was improved by explaining the denial and including root-cause configurations.

```
/* httpd: server/request.c (revision 99781; revision 1301504) */
} else if (APR_STATUS_IS_EACCES(rv)) {
  ap_log_rerror(APLOG_MARK, APLOG_ERR, rv, r,
                "access to %s denied ",
+               "(filesystem path '%s') ",
+               "because search permissions are missing "
+               "on a component of the path",
-               r->uri);
+               r->uri, r->filename);
  return r->status = HTTP_FORBIDDEN;
}
```

added in v2.4.2

added in v2.4.1

(b) The complete principle: the log message was improved by including details of the denied access

Figure 2. Demonstration of three design principles of access-denied messages (real-world postmortem improvements for usability issues).

in Apache and MySQL. Thus, besides the basic APIs (defined in `fcntl.h`, `unistd.h`, `stat.h`, `stdio.h`, `java.io`, and `java.nio`), we include customized file APIs for each studied system and track their error returns. Further, if the return values and error code are propagated to higher-level APIs, we recursively treat these APIs as customized file APIs. The method is similar as error-code propagation analysis [44], but with the special focus on identifying the log messages for the denied access.

Principles of access-denied messages

We study access-denied messages based on the following three principles. In essence, the messages should present specific, precise and complete information to help sysadmins understand missing permissions and resolve issues correctly.

Specific: Every access denial should have specific messages for explaining the root causes of the denial. Generic notification messages such as "`permission denied`" are not helpful for problem-solving [34, 75].

Precise: When denying access, the software should print precise messages to pinpoint root-cause configurations (e.g., the parameters or missing permissions). This is considered the key to enabling sysadmins to resolve configuration problems [70, 72, 74, 79].

Complete: The messages should include critical details of the denied access to enable sysadmins or automated tools to reproduce the issue and reason out the root causes. To deduce the missing permissions, one needs to know "'who' can do 'what' to 'which' things [14, 30, 32].

Figure 2 demonstrates these three principles using real-world examples where the messages were improved by the developers as postmortem improvements for usability issues.

Denied access occasionally has specific messages

Table 9 shows the percentage of access denials that has specific explanatory messages. In the studied C/C++ systems, only 66.5% of the program points that deny access have specific

Software	Internal denials		File access denials	
	w/ specific logs	Total	w/ specific logs	Total
Apache	63.7% (56)	88	88.2% (180)	204
MySQL	85.1% (40)	47	80.9% (313)	387
Lighttpd	63.2% (12)	19	71.4% (50)	70
VSFTP	44.0% (11)	25	78.4% (29)	37
Hadoop	100.0% (49)	49	88.0% (271)	308
HBase	100.0% (36)	36	86.7% (26)	30

Table 9. Program points of access denials that have specific log messages.

```
/* httpd-2.4.20: modules/aaa/mod_authz_core.c */
authz_status method_check_authorization(request_rec *r, ...) {
  ...                          ◄ check access w/ method-based ACL configs
  return AUTHZ_DENIED;         ◄ no log in this function
}

authz_status ip_check_authorization(request_rec *r, ...) {
  ...                          ◄ check access w/ IP-based ACL configs
  return AUTHZ_DENIED;         ◄ no log in this function
}

authz_status host_check_authorization(request_rec *r, ...) {
  ...                          ◄ check access w/ host-based configs
  return AUTHZ_DENIED;         ◄ no log in this function
}

int authorize_user_core(request_rec* r, ...) {
  ...                          ◄ call the above check functions
  if (auth_result == AUTHZ_DENIED) {
    ...
    ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, APLOGNO(01631)
                  "user %s: authorization failure for %s:",
                  r->user, r->uri);
}
```

Figure 3. An examples of denying access without giving specific messages in Apache: access denied with different causes (`method`, `IP`, and `host`) all comes with the same generic notification.

```
/* mysql-5.7.13: storage/innobase/row/row0import.cc */
FILE* file = fopen(name, "rb");
if (file == NULL) {
  /* If there is no cfp file, return directly */
  import.m_cfp_missing = true;
  return DB_SUCCESS;
} ...
```

Silently ignoring the denied access

Figure 4. An example of denied file access without any log messages in MySQL: the denial is treated as file non-existence and is silently ignored.

messages to explain the denials; the remaining ones typically fall into generic notifications, such as "`authorization failure`" in the example of Figure 3. As shown in the study of real-world access-denied issues, such generic notification messages cause difficulties for sysadmins to nail down the root-cause configurations, and thus lead to trial-and-error based problem-solving. In the studied Java systems, the denied access is manifested through exceptions such as `AccessControlException` in Hadoop and `AccessDeniedException` in HBase. The software developers are enforced to write a specific message in order to initialize the exception objects (required by the constructor of the exception class). As a result, all the access-denied points in the studied Java programs have specific messages.

The majority (83.9%) of file access have log messages upon denials, while the rest silently ignore the denied access (c.f., Table 9). Most of the unlogged file access denials are treated as file non-existence, as exemplified in Figure 4. Some of them are attributed to the underlying library APIs. In fact,

Software	Pinpoint root-cause config	Internal denials
Apache	87.5% (49)	56
MySQL	17.5% (7)	40
Lighttpd	33.3% (4)	12
VSFTP	0.0% (0)	11
Hadoop	38.8% (19)	49
HBase	13.9% (5)	36
Total	41.2% (84)	204

Table 10. The percentage of messages that pinpoint root-cause configurations when the software denies access.

Software	Error code	Subject	Operation	Object	Total
Apache	71.1% (128)	0.0% (0)	54.4% (98)	81.1% (146)	180
MySQL	56.2% (176)	0.0% (0)	62.0% (194)	81.8% (256)	313
Lighttpd	76.0% (38)	0.0% (0)	46.0% (23)	90.0% (45)	50
VSFTP	3.4% (1)	0.0% (0)	48.3% (14)	65.5% (19)	29
Hadoop	59.8% (162)	0.0% (0)	93.4% (253)	95.2% (258)	271
HBase	92.3% (24)	0.0% (0)	100.0% (26)	96.2% (25)	26
Total	60.9% (529)	0.0% (0)	70.0% (608)	86.2% (749)	869

Table 11. Characteristics of log messages for denied file access. Many log messages miss critical details such as error type, subject, file operation, and file object.

```
/* hadoop-2.7.2: hadoop-hdfs-project/.../server/datanode/DataNode.java */
throw new AccessControlException(
    "Can't continue with getBlockLocalPathInfo() " +
    "authorization. The user " + currentUser + " " +
-   "is not allowed to call getBlockLocalPathInfo");
+   "is not configured in " +
+   "dfs.block.local-path-access.user");
```

Root-cause parameter

Figure 5. An example of messages that do not pinpoint the root-cause configurations, and our patch to improve it (accepted by the developers).

```
/* hadoop-2.7.2: hadoop-common-project/.../util/RunJar.java */
System.err.println("Not a valid JAR: " +
                   file.getCanonicalPath());
```

(a) Missing error code: it is hard to infer the root cause of the denied access.

```
/* vsftpd-3.0.3: logging.c */
die("chroot");
```

(b) Missing file object: the log message does not record the target root directory.

```
/* mysql-5.7.13: plugin/innodb_memcached/.../daemon/memcached.c */
log(EXTENSION_LOG_WARNING, NULL, "Failed to open %s: %s\n",
                temp_portnumber_filename, strerror(errno));
```

(c) Missing access type (file operation): the message is printed only when the file is not *writable*.

Figure 6. Different types of inadequate log messages for the denied file access in the studied software programs.

several Java's built-in libraries and APIs equalize denied access and file non-existence. For example, in Java's built-in library `java.io`, all the denied access is manifested through `FileNotFoundException`; APIs like `exists` (in both `java.io` and `java.nio`) do not differentiate between permission errors and file non-existence. Such practice commonly results in confusion when the access is denied for files that do exist [38].

Root-cause configurations are rarely pinpointed

Next we study the content of each specific access-denied message. We focus on the log messages when the software denied access internally. In this case, the software has the opportunity to pinpoint the root-cause configurations (either the relevant parameters or the missing permissions). However, only 41.2% of log messages follow this principle, as shown in Table 10. One main reason behind such inadequate messages is that software developers do not design message content with helping sysadmins in mind; instead, they often assume that sysadmins understand the implementation, as exemplified by Figure 5.

Some of the log messages, though referring to specific denied access, are still in the form of generic notifications. For example, all the denied access to tables in MySQL come with the following notorious message:

> "`%s command denied to user %s for table %s`"

As shown in our study, many database administrators cannot figure out the exact missing permissions when their operations are denied, and consequently choose to grant `ALL` privileges as workarounds. This is quite understandable, given that MySQL has 32 distinct privileges and 12 of them apply to tables.

Messages for denied file access miss critical details

We then study the log messages for each denied file access. Assisting resolutions for file access denials needs complete information of the subject, access type, and file object. Moreover, as the denied file access is manifested through `errno` or exceptions in the same way as other types of I/O errors such as disk errors (the file `open` system call can return 21 different `errno`, each representing a unique error type), the log messages should include `errno` or exceptions. Unfortunately, our evaluation shows that many of the log messages for denied access miss these critical details.

Table 11 characterizes the log messages for file access denials. Only 60.9% and 70.0% of messages include `errno`/exceptions and access types, respectively. Compared with C/C++ systems, the studied Java systems have more complete messages because many Java file APIs throw exceptions with stack traces upon the denied access, and thus automatically include the file objects and the operations. In C/C++ systems all these messages were manually written by the software developers. However, Java also provides C/C++ style APIs such as `mkdir` and `renameTo` (`java.io.File`) that use `false` returns to indicate all types of errors. Most of the studied log messages include the denied file objects in the content (but still 13.8% inadvertently miss the information). Figure 6 shows the three types of inadequate file-access messages in the studied software.

Notably, none of the file access denied messages include the subject (against whom access is denied), likely because the subject is not concerned in the other types of access errors (e.g., invalid file names, insufficient memory, and disk errors). However, subject information is critical for access control. In our study, there are several cases in which the sysadmins had difficulties in obtaining subject information, especially when the programs use `setuid` or `setgid`.

IMPLICATIONS

Improving usability of access control requires efforts of both software systems and tooling support. Our results show that current design and implementation of access control are far from adequate to assist correct, secure configurations. One

fundamental cause of this situation is that software designers and developers often assume that sysadmins are so expert that the usability of access control is not a concern for them. Our study invalidates this assumption—without adequate feedback and tooling, sysadmins introduce misconfigurations and pose security risks. Our study draws following design implications.

Design of software systems. Our results show that current system feedback via log messages is far from adequate to assist resolving access-denied issues with correct configurations. The feedback messages should be designed with the goal of helping sysadmins to correctly resolve the denied access; they should follow the usability principles—being specific, precise, and complete. It is important to educate developers and raise the awareness that access-denied messages should be designed differently from log messages for other usages (e.g., debugging logs are mostly used by developers themselves). We hope that our work can serve this purpose.

Automated tools for evaluating and enhancing access-denied messages can significantly improve the quality of feedback information. Most of the existing log-enhancement techniques focus on reconstructing execution paths [77] and failure-site information [76] to improve software diagnosability. There are techniques for detecting inadequate log messages for configuration errors that result in functionality issues [79]. These approaches can be extended to detect inadequate access-denied messages based on the usability principles of access control.

Design of tool support. The ultimate tool support is to automate the resolutions for access-denied issues, thus avoiding human errors. This is not impossible: though setting complicated access-control configurations is difficult for human administrators, it is straightforward for automated programs. In fact, SELinux's utilities like `audit2allow` have demonstrated the feasibility to automatically generate correct access-control configurations to allow denied access. In our study of CentOS cases (CentOS enables SELinux by default), we observe that `audit2allow` significantly improves the usability and increases the adoption of SELinux. This approach should be generalized and implemented in other software systems.

Furthermore, tool support for articulating security impact of access-control configurations is highly desired. Our study reveals that one common difficulty of sysadmins is to understand the impact of configuration changes, e.g., who obtains more access after the change. Informing and visualizing such information would help sysadmins make more reasonable and cautious access-control decisions. For common resources such as Unix files and SQL databases, generic tools can be made based on their semantics, while application may need specific implementations for their own access-control mechanisms.

RELATED WORK

The previous studies on access control mainly focus on authoring and visualizing access-control policies. Bauer et al. examined the physical access-control policies and found that users' intended policies were better matched with a flexible access-control system [6]. In another study, Bauer et al. identified three sets of requirements either ignored or inadequately addressed for implementing access-control policies [7]. They reported that policy makers were often distinct from policy implementers; the separation raises challenges of ensuring that the actual configurations correctly implement policies. Sinclair et al. studied access control practices of three organizations and reported that access control was profoundly broken in real-world practices due to over-simplified or wrong assumptions [56,57]. The consequences include frequent over-entitlement (granting too much access) and constant under-entitlement that leads to access-denied issues. Reeder et al. argued that the traditional list-of-rules model is deficient in displaying policies when multiple rules interact, as it cannot convey the interactions amongst rules to policy authors [42]. In another study, Reeder et al. demonstrated that the conflict-resolution method for ALLOW and DENY rules had significant usability impact on the implementation of access-control policies [43]. Adage is one of the first authorization services that place usability at the center of its design [80].

This paper concerns about a complementary aspect of access-control usability—resolving access-denied issues. Certainly, the ultimate solutions is creating perfect configurations to allow access to all authorized requests. However, due to a number of practical challenges, in reality many sysadmins still have to manually resolve access-denied issues on an as-needed basis. As shown in this paper, resolving access-denied issues is critical to the correctness of access control.

Other works have studied access control of various end-user applications, including personal/home data sharing [35, 36], web applications [3, 11], social network services [18, 27], and mobile apps [20]. End-user access control is completely different from security administration of system software studied in this paper. It mainly focuses on protecting the privacy and security of the individuals who are less technical and less motivated [25, 69]; end users do not manage server/OS software in which security is one primary concern. Also, the feedback information for end users needs to balance usability and disclosure of sensitive information [29]; this is less a concern for sysadmins who are policy makers and implementers and usually have highest privileges. Lastly, GUI-based techniques designed for end users are not applicable to system software which typically are not managed via GUIs [5, 22, 66].

CONCLUDING REMARKS

This paper advocates the importance of assisting correct resolutions for access-denied issues—a critical but often ignored aspect of access control usability. We show that the common problem-solving practices based on trial and error are prone to security misconfigurations, and are inefficient in allowing the denied access. Consequently, security misconfigurations are commonly introduced as workarounds in the field, imposing significant security risks. Our study reveals that security misconfigurations are often the results of lacking precise understanding of the root causes (the missing permissions), due to inadequate feedback information and tool support provided by today's software systems. Our future work includes exploring automated approaches to resolving access-denied issues under different access-control models, and deducing security implications of sysadmin's resolutions (e.g., unauthorized access incorrectly granted by their configurations).

REFERENCES

1. Sepehr Amir-Mohammadian, Stephen Chong, and Christian Skalka. 2016. Correct Audit Logging: Theory and Practice. In *Proceedings of the 5th International Conference on Principles of Security and Trust (POST'16)*. Eindhoven, The Netherlands.
2. Apache Users Mailing List. 2016. http://mail-archives.apache.org/mod_mbox/httpd-users/. (2016).
3. Dirk Balfanz. 2003. Usable Access Control for the World Wide Web. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03)*. Las Vegas, NV, USA.
4. Rob Barrett, Yen-Yang Michael Chen, and Paul Maglio. 2009. System Administrators are Users, Too: Designing Workspaces for Managing Internet-Scale Systems. In *Proceedings of the 27th ACM Conference on Human Factors in Computing Systems (CHI'03)*. Boston, MA, USA.
5. Rob Barrett, Eser Kandogan, Paul P. Maglio, Eben Haber, Leila A. Takayama, and Madhu Prabaker. 2004. Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW'04)*. Baltimore, MD, USA.
6. Lujo Bauer, Lorrie Faith Cranor, Robert W. Reeder, Michael K. Reiter, and Kami Vaniea. 2008. A User Study of Policy Creation in a Flexible Access-Control System. In *Proceedings of the 26th ACM Conference on Human Factors in Computing Systems (CHI'08)*. Florence, Italy.
7. Lujo Bauer, Lorrie Faith Cranor, Robert W. Reeder, Michael K. Reiter, and Kami Vaniea. 2009. Real Life Challenges in Access Control Management. In *Proceedings of the 27th ACM Conference on Human Factors in Computing Systems (CHI'09)*. Boston, MA, USA.
8. Lujo Bauer, Scott Garriss, and Michael K. Reiter. 2008. Detecting and Resolving Policy Misconfigurations in Access-Control Systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT'08)*. Estes Park, CO, USA.
9. Lujo Bauer, Scott Garriss, and Michael K. Reiter. 2011. Detecting and Resolving Policy Misconfigurations in Access-Control Systems. *ACM Transactions on Information and System Security (TISSEC)* 14, 1 (May 2011), 1–28.
10. David Botta, Rodrigo Werlinger, André Gagné, Konstantin Beznosov, Lee Iverson, Sidney Fels, and Brian Fisher. 2007. Towards Understanding IT Security Professionals and Their Tools. In *Proceedings of the 2nd Symposium on Usable Privacy and Security (SOUPS'07)*. Pittsburgh, PA, USA.
11. Xiang Cao and Lee Iverson. 2006. Intentional access management: making access control usable for end-users. In *Proceedings of the 2nd Symposium on Usable Privacy and Security (SOUPS'06)*. Pittsburgh, PA, USA.
12. CentOS Forums. 2016. https://www.centos.org/forums/. (2016).
13. CentOS Mailing Lists. 2016. https://lists.centos.org/pipermail/centos/. (2016).
14. Anton Chuvakin and Gunnar Peterson. 2010. How to Do Application Logging Right. *IEEE Security & Privacy* 8, 4 (Jul. 2010), 82–85.
15. Tathagata Das, Ranjita Bhagwan, and Prasad Naldurg. 2010. Baaz: A System for Detecting Access Control Misconfigurations. In *Proceedings of the 19th USENIX Security Symposium (USENIX Security'10)*. Washington, DC, USA.
16. Database Administrators. 2016. http://dba.stackexchange.com. (2016).
17. Database Administrators #127003. 2016. Can't access MySQL database with created user through dll. http://dba.stackexchange.com/questions/127003/cant-access-mysql-database-with-created-user-through-dll. (Jan. 2016).
18. Serge Egelman, Andrew Oates, and Shriram Krishnamurthi. 2011. Oops, I Did it Again: Mitigating Repeated Access Control Errors on Facebook. In *Proceedings of the 29th ACM International Conference on Human Factors in Computing Systems (CHI'11)*. Vancouver, BC, Canada.
19. Daniel Farmer and Eugene H. Spafford. 1990. *The COPS Security Checker System*. Technical Report CSD-TR-993. Purdue University, Department of Computer Science.
20. Adrienne Porter Felt. 2012. *Towards Comprehensible and Effective Permission Systems*. Ph.D. Dissertation. University of California at Berkeley, Electrical Engineering and Computer Sciences.
21. Jeff Goldman. 2014. Misconfigured Server Causes Massive Data Breach at MBIA. http://www.esecurityplanet.com/network-security/misconfigured-server-causes-massive-data-breach-at-mbia.html. (Oct. 2014).
22. Eben M. Haber and John Bailey. 2007. Design Guidelines for System Administration Tools Developed through Ethnographic Field Study. In *Proceedings of the 1st ACM Symposium on Computer Human Interaction for Management of Information Technology (CHIMIT'07)*. Cambridge, MA, USA.
23. Hadoop User Mailing Lists. 2016. http://mail-archives.apache.org/mod_mbox/hadoop-user/. (2016).
24. Hewlett Packard Enterprise. 2015. HP Cyber Risk Report 2015. http://www8.hp.com/h20195/v2/GetPDF.aspx/4AA5-0858ENN.pdf. (Feb. 2015).

25. Adele E. Howe, Indrajit Ray, Mark Roberts, Malgorzata Urbanska, and Zinta Byrne. 2012. The Psychology of Security for the Home Computer User. In *Proceeding of the 33rd IEEE Symposium on Security & Privacy (S&P)*. San Francisco, CA, USA.
26. Weihang Jiang, Chongfeng Hu, Shankar Pasupathy, Arkady Kanevsky, Zhenmin Li, and Yuanyuan Zhou. 2009. Understanding Customer Problem Troubleshooting from Storage System Logs. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST'09)*. San Francisco, CA, USA.
27. Maritza L. Johnson. 2012. *Toward Usable Access Control for End-users: A Case Study of Facebook Privacy Settings*. Ph.D. Dissertation. Columbia University, Computer Science.
28. Eser Kandogan and Eben M. Haber. 2005. Security Administration Tools and Practices. *Security and Usability, O'Reilly Media, Inc.* (Aug. 2005).
29. Apu Kapadia, Geetanjali Sampemane, and Roy H. Campbell. 2004. Know Why Your Access Was Denied: Regulating Feedback for Usable Security. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*. Washington, DC, USA.
30. Butler Lampson. 2009. Usable Security: How to Get It. *Commun. ACM* 52, 11 (Nov. 2009), 25–27.
31. Butler Lampson. 2015. Perspectives on Security. SOSP History Day. (Oct. 2015). http://sigops.org/sosp/sosp15/history/.
32. Butler W. Lampson. 1974. Protection. *ACM SIGOPS Operating Systems Review* 8, 1 (Jan. 1974), 18–24.
33. Butler W. Lampson. 2004. Computer Security in the Real World. *IEEE Computer* 37, 6 (Jun. 2004), 37–46.
34. Paul P. Maglio and Eser Kandogan. 2004. Error Messages: What's the Problem? *ACM Queue* 2, 8 (Nov. 2004), 50–55.
35. Roy A. Maxion and Robert W. Reeder. 2005. Improving User-Interface Dependability through Mitigation of Human Error. *International Journal of Human-Computer Studies* 63, 1-2 (Jul. 2005), 25–50.
36. Michelle L. Mazurek, J. P. Arsenault, Joanna Bresee, Nitin Gupta, Iulia Ion, Christine Johns, Daniel Lee, Yuan Liang, Jenny Olsen, Brandon Salmon, Richard Shay, Kami Vaniea, Lujo Bauer, Lorrie Faith Cranor, Gregory R. Granger, and Michael K. Reiter. 2010. Access Control for Home Data Sharing: Attitudes, Needs, and Practices. In *Proceedings of the 28th ACM International Conference on Human Factors in Computing Systems (CHI'10)*. Atlanta, GA, USA.
37. Microsoft Baseline Security Analyzer. 2016. http://www.microsoft.com/technet/security/tools/MBSAHome.mspx. (2016).
38. MySQL Bug #80542. 2016. Should notify users when ignoring their option files. http://bugs.mysql.com/bug.php?id=80542. (Feb. 2016).
39. MySQL General Discussion Mailing List. 2016. https://lists.mysql.com/mysql. (2016).
40. Peter G. Neumann. 2015. Reminiscences on the 25th SOSP's History Day Workshop. http://sigops.org/sosp/sosp15/history/12-neumann.pdf. (2015).
41. Eric Steven Raymond and Rick Moen. 2014. How To Ask Questions The Smart Way. http://www.catb.org/~esr/faqs/smart-questions.html. (2014).
42. Robert W. Reeder, Lujo Bauer, Lorrie Faith Cranor, Michael K. Reiter, Kelli Bacon, Keisha How, and Heather Strong. 2008. Expandable Grids for Visualizing and Authoring Computer Security Policies. In *Proceedings of the 26th ACM Conference on Human Factors in Computing Systems (CHI'08)*. Florence, Italy.
43. Robert W. Reeder, Lujo Bauer, Lorrie Faith Cranor, Michael K. Reiter, and Kami Vaniea. 2011. More than Skin Deep: Measuring Effects of the Underlying Model on Access-Control System Usability. In *Proceedings of the 29th ACM International Conference on Human Factors in Computing Systems (CHI'11)*. Vancouver, BC, Canada.
44. Cindy Rubio-González, Haryadi S. Gunawi, Ben Liblit, Remzi H. Arpaci-Dusseau, and Andrea C. Arpaci-Dusseau. 2009. Error Propagation Analysis for File Systems. In *Proceedings of the 30th Annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'09)*. Dublin, Ireland.
45. Bruce Schneier. 2009. Real-World Access Control. https://www.schneier.com/blog/archives/2009/09/real-world_acce.html. (2009).
46. ServerFault. 2016. http://serverfault.com. (2016).
47. ServerFault #235376. 2011. Apache/PHP writing to a user's home folder. http://serverfault.com/questions/235376/apache-php-writing-to-a-users-home-folder. (Feb. 2011).
48. ServerFault #287508. 2011. Apache Permission denied problem on OS X. serverfault.com/questions//apache-permission-denied-problem-on-os-x. (Jul. 2011).
49. ServerFault #396036. 2012. Apache httpd permissions. http://serverfault.com/questions/396036/apache-httpd-permissions. (Jun. 2012).
50. ServerFault #401115. 2012. Apache Mod SVN Access Forbidden. http://serverfault.com/questions/401115/apache-mod-svn-access-forbidden. (Jun. 2012).
51. ServerFault #402630. 2012. HTTP Error: 413 Request Entity Too Large. http://serverfault.com/questions/402630/http-error-413-request-entity-too-large. (Jun. 2012).

52. ServerFault #418101. 2012. Apache "Client denied by server configuration", despite allowing access to directory (vhost configuration). http://serverfault.com/questions/418101/apache-client-denied-by-server-configuration-despite-allowing-access-to-direc. (Aug. 2012).
53. ServerFault #421633. 2012. Cannot write file with apache despite 777 permissions on Directory. http://serverfault.com/questions/421633/cannot-write-file-with-apache-despite-777-permissions-on-directory. (Aug. 2012).
54. ServerFault #507914. 2013. Mysql working remotely but not localy. http://serverfault.com/questions/507914/mysql-working-remotely-but-not-localy. (May 2013).
55. ServerFault #65362. 2009. Is there a work around for the "Permission denied" error from urllib2.urlopen? http://serverfault.com/questions//is-there-a-work-around-for-the-permission-denied-error-from-urllib2-urlopen. (Sep. 2009).
56. Sara Sinclair. 2013. *Access Control In and For the Real World*. Ph.D. Dissertation. Dartmouth College, Department of Computer Science.
57. Sara Sinclair and Sean W. Smith. 2010. What's Wrong with Access Control in the Real World? *IEEE Security & Privacy* 8, 4 (Jul. 2010), 74–77.
58. StackOverflow. 2016. http://stackoverflow.com. (2016).
59. StackOverflow #13542384. 2012. Setting permissions for cloudera hadoop. http://stackoverflow.com/questions/13542384/setting-permissions-for-cloudera-hadoop. (Nov. 2012).
60. StackOverflow #22664268. 2014. Error on map reduce example of Hadoop 2.2.0. http://stackoverflow.com/questions/22664268/error-on-map-reduce-example-of-hadoop-2-2-0. (Mar. 2014).
61. StackOverflow #29438893. 2015. Hadoop jar execution failing on class not found. http://stackoverflow.com/questions/29438893/hadoop-jar-execution-failing-on-class-not-found. (Apr. 2015).
62. StackOverflow #30926357. 2015. Oozie on YARN - oozie is not allowed to impersonate hadoop. http://stackoverflow.com/questions/30926357/oozie-on-yarn-oozie-is-not-allowed-to-impersonate-hadoop. (Jun. 2015).
63. StackOverflow #33358339. 2015. Hadoop "error while writing TokenizerMapper permission denied". http://stackoverflow.com/questions/33358339/hadoop-error-while-writing-tokenizermapper-permission-denied. (Jun. 2015).
64. StackOverflow #473789. 2013. mysql doesn't start after relocating data dir. http://serverfault.com/questions/473789/mysql-doesnt-start-after-relocating-data-dir. (Jan. 2013).
65. StackOverflow Help Center. 2016. Why is voting important? http://stackoverflow.com/help/why-vote. (2016).
66. Leila Takayama and Eser Kandogan. 2006. Trust as an Underlying Factor of System Administrator Interface Choice. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI EA'06)*. Montréal, Québec, Canada.
67. The Open Web Application Security Project. 2013. OWASP Top 10: The Ten Most Critical Web Application Security Risks. http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf. (2013).
68. Dan Walsh. 2013. Audit2allow should be your third option not the first. http://danwalsh.livejournal.com/63137.html. (2013).
69. Ryan West. 2008. The Psychology of Security. *Commun. ACM* 51, 4 (Apr. 2008), 34–40.
70. Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, You Have Given Me Too Many Knobs! Understanding and Dealing with Over-Designed Configuration in System Software. In *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15)*. Bergamo, Italy.
71. Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. 2016. Early Detection of Configuration Errors to Reduce Failure Damage. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. Savannah, GA, USA.
72. Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. 2013. Do Not Blame Users for Misconfigurations. In *Proceedings of the 24th Symposium on Operating System Principles (SOSP'13)*. Farmington, PA, USA.
73. Tianyin Xu and Yuanyuan Zhou. 2015. Systems Approaches to Tackling Configuration Errors: A Survey. *ACM Computing Surveys (CSUR)* 47, 4 (Jul. 2015).
74. Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram, and Shankar Pasupathy. 2011. An Empirical Study on Configuration Errors in Commercial and Open Source Systems. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*. Cascais, Portugal.
75. Ding Yuan. 2012. *Improving Failure Diagnosis with Better Logging Design and Analysis*. Ph.D. Dissertation. University of Illinois Urbana-Champaign, Department of Computer Science.
76. Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M. Lee, Xiaoming Tang, Yuanyuan Zhou, and Stefan Savage. 2012. Be Conservative: Enhancing Failure Diagnosis with Proactive Logging. In *Proceedings of the*

10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12). Hollywood, CA, USA.

77. Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. 2011. Improving Software Diagnosability via Log Enhancement. In *Proceedings of the 16th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS'11)*. Newport Beach, CA, USA.

78. Jiaqi Zhang, Lakshmi Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. 2014. EnCore: Exploiting System Environment and Correlation Information for Misconfiguration Detection. In *Proceedings of the 19th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS'14)*. Salt Lake City, UT, USA.

79. Sai Zhang and Michael D. Ernst. 2015. Proactive Detection of Inadequate Diagnostic Messages for Software Configuration Errors. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA'15)*. Baltimore, MD, USA.

80. Mary Ellen Zurko, Rich Simon, and Tom Sanfilippo. 1999. A User-Centered, Modular Authorization Service Built on an RBAC Foundation. In *Proceedings of the 20th IEEE Symposium on Security and Privacy (S&P)*. Oakland, CA, USA.