Improving Logging to Reduce Permission Over-Granting Mistakes

Bingyu Shen, Tianyi Shan, Yuanyuan Zhou
Access Control Protects User Data on Cloud
Prevalent misconfigurations cause newsworthy data leaks

Oct 6, 2021, twitch suffered from a major data breach including source code and creator payouts, because a hacker accessed the company’s servers due to a server configuration change.

Mar 9, 2022, 70% of tested servicenow instances were vulnerable because misconfigured customer-managed Access Control Lists (ACLs) and overprovisioning of permissions to guest users.

Apr. 12, 2022, 13 million users’ records were found public because the misconfiguration of Content Management System (CMS) database.
Misconfigurations introduced in solving access-denied issues

Users’ legitimate access falsely denied by servers

Errors: The requested URL could not be retrieved

Diagnose and make a configuration change

Left unnoticed until...
Misconfigurations may be introduced in solving access-denied issues

Months later, find the vulnerability and gain access*

Left unnoticed until ...

Diagnose and make a configuration change

* https://www.varonis.com/blog/data-breach-response-times/
Log messages are the main feedback for sysadmins

- Log messages help sysadmins develop the precise understanding of the root cause.

  - 87.0% of sysadmins check logs
  - only 1.6% of sysadmins looked at source code

*How Do System Administrators Resolve Access-Denied Issues in the Real World?, CHI’17*
Poor logging practice in server software

1. The log messages do not contain specific/complete information

```c
status ip_check_authorization(request *r, ...) {
    ... // <- check request’s IP w/ config.
    return AUTHZ_DENIED;
}
status method_check_authorization(request *r, ...) {}
status host_check_authorization(request *r, ...) {}
...
if (auth_result == AUTHZ_DENIED) {
    ap_log_rerror(APLOG_ERR, r, APLOGNO(01631)
        "%s: authorization failure for \%s\": ", r->user, r->uri);
}
/* httpd-2.4.46: modules/aaa/mod_authz_core.c */
```
Poor logging practice in server software

1. The log messages do not contain specific/complete information.
2. Developers may miss access-deny locations.

/* Cherokee-1.2.103: cherokee/config_reader.c*/
dir = cherokee_opendir (path->buf);
if (dir == NULL)
    return ret_error;

Silently ignoring the denied access
Poor logging practice in server software

1. The log messages do not contain specific/complete information
2. Developers may miss access-deny locations.

```c
/* 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;
```

How good are the logging practices in the server software?
Understanding real-world server software

Q1: How often do server software have log messages when the access is denied?

Q2: Do the log messages have complete information related to the denied access?

Q3: How do software developers add or update access-deny log messages?
Study preparation

• Find access-denied locations and analyze the log messages

• Target on server software
  • 5 server programs: web server, database, FTP, NFS
Q1: How often do server software have no log messages when the access is denied?

A: Even in mature software, 14.0% to 70.5% of cases have no log messages when an access is denied.

<table>
<thead>
<tr>
<th>Application</th>
<th>No logs at default level</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache httpd</td>
<td>63</td>
<td>35.4%</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>107</td>
<td>22.2%</td>
</tr>
<tr>
<td>Vsftpd</td>
<td>10</td>
<td>14.0%</td>
</tr>
<tr>
<td>NFS-ganesha</td>
<td>43</td>
<td>70.5%</td>
</tr>
<tr>
<td>Proftpd</td>
<td>209</td>
<td>58.7%</td>
</tr>
</tbody>
</table>
Specific information for a request

• Q2: Do the log messages have complete information related to the denied access?

<table>
<thead>
<tr>
<th>subject</th>
<th>action</th>
<th>object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web server</td>
<td>Web server user</td>
<td>GET, POST</td>
</tr>
<tr>
<td>Database</td>
<td>DB user</td>
<td>SELECT, UPDATE</td>
</tr>
</tbody>
</table>
Finding 2

Q2: Do the log messages have complete information related to the denied access?

A1: Servers have subject, action, or object at different levels.

<table>
<thead>
<tr>
<th>Application</th>
<th>Subject</th>
<th>Action</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache httpd</td>
<td>13</td>
<td>90</td>
<td>97</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>99</td>
<td>132</td>
<td>267</td>
</tr>
<tr>
<td>Vsftpd</td>
<td>2</td>
<td>41</td>
<td>15</td>
</tr>
<tr>
<td>NFS-ganesha</td>
<td>15</td>
<td>24</td>
<td>18</td>
</tr>
<tr>
<td>Proftpd</td>
<td>0</td>
<td>146</td>
<td>146</td>
</tr>
</tbody>
</table>
Finding 2

Q2: Do the log messages have complete information related to the denied access?

A1: Servers have subject, action, or object at different levels.

A2: The information is available at the same function at the denied location.

<table>
<thead>
<tr>
<th>Application</th>
<th>Subject</th>
<th>Action</th>
<th>Object</th>
<th>In same func.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache httpd</td>
<td>13</td>
<td>90</td>
<td>97</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>11.3%</td>
<td>78.3%</td>
<td>84.3%</td>
<td>97.3%</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>99</td>
<td>132</td>
<td>267</td>
<td>357</td>
</tr>
<tr>
<td></td>
<td>26.5%</td>
<td>36.9%</td>
<td>71.4%</td>
<td>95.5%</td>
</tr>
<tr>
<td>Vsftpd</td>
<td>2</td>
<td>41</td>
<td>15</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>3.3%</td>
<td>67.2%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>NFS-ganesha</td>
<td>15</td>
<td>24</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>60.0%</td>
<td>96.0%</td>
<td>68.0%</td>
<td>68.0%</td>
</tr>
<tr>
<td>Proftpd</td>
<td>0</td>
<td>146</td>
<td>146</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Q3: How do software developers add/update access-deny log messages for sysadmins?

A: Access-deny logging practice is ad-hoc and many existing log messages are added as afterthoughts.

<table>
<thead>
<tr>
<th>Application</th>
<th>Add logs</th>
<th>Revise logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache httpd</td>
<td>40</td>
<td>511</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>165</td>
<td>1728</td>
</tr>
<tr>
<td>Vsftpd</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NFS-ganesha</td>
<td>7</td>
<td>344</td>
</tr>
<tr>
<td>Proftpd</td>
<td>33</td>
<td>1209</td>
</tr>
</tbody>
</table>
Can we automatically improve access-deny logging?
Our solution: SecLog

• To help developers to enhance existing log messages and insert missing log messages
• To provide guidance for sysadmins to fix access-deny issues without over-granting permissions
Challenge 1: How to log

• Manual solution
  • Rely on developers to manually log
    • Cons 1: Many logging locations
    • Cons 2: Manually examine call chains to collect info

• SecLog solution
  • Automated static analysis tool to assist the logging process
  • Can be integrated in development process

CI / CD
Challenge 2: Where to log (1)

1. Where to find access control check (ACC) locations?

```c
// ACC function in Cherokee web server
ret = cherokee_mkdir_p_perm(&path, 0775, W_OK);
if (ret != ret_ok) {
    LOG_CRITICAL("Cannot create the ‘%s’ directory", path);
    return ret_error;
}

// ACC function in PostgreSQL
aclresult = pg_class_aclcheck(TableID, GetUserId(), ACL_SELECT);
if (aclresult != ACLCHECK_OK)
    aclcheck_error(aclresult, ACL_SELECT, TableID);
```

SecLog solution:
- Use ACC functions to find access-control check locations (require developer annotation)
Challenge 2: Where to log (2)

2. Where to place the log statements
   - Inside the ACC function
   - At the ACC function’s call site

<table>
<thead>
<tr>
<th></th>
<th>Inside the ACC function</th>
<th>At the ACC function’s call site</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td>• Low analysis cost</td>
<td>• More comprehensive info</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td>• May miss the info at call site</td>
<td>• More analysis cost</td>
</tr>
</tbody>
</table>
Challenge 2: Where to log (2)

2. Where to place the log statements
   • Inside the ACC function
   • At the ACC function’s call site

<table>
<thead>
<tr>
<th></th>
<th>Inside the ACC function</th>
<th>At the ACC function’s call site</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td>• Low analysis cost</td>
<td>• More comprehensive info</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td>• May miss the info at call site</td>
<td>• More analysis cost</td>
</tr>
</tbody>
</table>
Challenge 3: What to log

• The critical information to insert into log statements
  1. Inside the ACC function
  2. At the ACC function’s call site

• Approach
  • Extract relevant variables with static analysis
Challenge 3: What to log

1. Inside the ACC function
   • From the return value, perform backward slicing
   • Extract the reasons for each deny

```
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;
    }
    /* dir exist, check permissions */
    ret = cherokee_access(dir->buf, perm);
    if (ret != ret_ok)
        return ret_deny;
    return ret_ok;
}
```
Challenge 3: What to log

1. Inside the ACC function
   • From the return value, perform backward slicing
   • Extract the reasons for each deny

Access control check function

```c
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;
  }
  /* dir exist, check permissions */
  ret = cherokee_access(dir->buf, perm);
  if (ret != ret_ok)
    return ret_deny;
  return ret_ok;
}
```
Challenge 3: What to log

1. Inside the ACC function
   • From the return value, perform backward slicing
   • Extract the reasons for each deny

2. At the function’s call site
   • From the relevant variables, perform data dependency analysis

```c
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;
}
/* dir exist, check permissions */
ret = cherokee_access(dir->buf, perm);
if (ret != ret_ok)
    return ret_deny;
return ret_ok;
```
Evaluation

• Log improvement
  • Overall results
    • Helpfulness of identified variables
• User study
  • Performance overhead
• Adoption efforts
Evaluation

- In total, SecLog inserts 380 new access-deny log statements and enhances 550 existing ones
- Currently, **70 of 114 submitted log improvements have been accepted**

<table>
<thead>
<tr>
<th>App</th>
<th>Existing Logs</th>
<th>New Logs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td># new vars</td>
</tr>
<tr>
<td>Apache httpd</td>
<td>93</td>
<td>5.62</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>203</td>
<td>2.61</td>
</tr>
<tr>
<td>vsftpd</td>
<td>39</td>
<td>1.67</td>
</tr>
<tr>
<td>NSF-granshea</td>
<td>8</td>
<td>3.00</td>
</tr>
<tr>
<td>Proftpd</td>
<td>145</td>
<td>4.03</td>
</tr>
<tr>
<td>Postfix</td>
<td>8</td>
<td>8.75</td>
</tr>
<tr>
<td>HAProxy</td>
<td>3</td>
<td>3.33</td>
</tr>
<tr>
<td>Cherokee</td>
<td>11</td>
<td>2.18</td>
</tr>
<tr>
<td>redis</td>
<td>11</td>
<td>1.82</td>
</tr>
<tr>
<td>mSQL</td>
<td>19</td>
<td>2.00</td>
</tr>
</tbody>
</table>
Evaluation: User study

- User study
  - Recruit 32 professionals
  - 3 problems each for vsftpd/PostgreSQL
  - The only difference is the log message

<table>
<thead>
<tr>
<th># (%) of insecure solutions</th>
<th>Problem 1</th>
<th>Problem 2</th>
<th>Problem 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>SECLOG</td>
<td>Original</td>
</tr>
<tr>
<td>vsftpd</td>
<td>1 (6.25%)</td>
<td>0 (0.0%)</td>
<td>6 (37.5%)</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>8 (50.0%)</td>
<td>0 (0.0%)</td>
<td>4 (25.0%)</td>
</tr>
</tbody>
</table>

SecLog-enhanced log group 3x faster
Conclusion

• An observation study on existing access-deny logging practice
  • Better logging for access control administration

• Design choices for improving access-deny logging
  • How to log
  • Where to log
  • What to log
Limitations

• Imprecise intra-procedure static analysis
• SecLog can not automatically generates natural language log messages
• SecLog does not fix the problems for sysadmins.
Thank you

Q&A