Improving Logging to Reduce Permission Over-Granting Mistakes (Supplementary Materials)

1 Measuring Efforts to Annotate ACC functions and Result Check Functions

Methodology To further measure the manual efforts in annotation for SECLOG, we recruited two graduate students in computer science (no co-authors of this paper) 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.

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

Table 1: **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.

Results: The annotation results are shown in Table 1. From the study, we find that even for developers who have less experience and knowledge of the software, they are able to find the ACC functions and annotate the result check functions with an average coverage of 92.4% compared to the total ACC functions used in the evaluation. And the average time spent on each application is 24.8 minutes. The manual efforts required for software maintainers who are familiar with the source code could be even smaller.

2 Measuring Manual Efforts to Write Log Messages with SECLOG's Outputs

Methodology We conducted 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 were provided with the four applications' source code with specific versions that SECLOG evaluated on, including Apache, PostgreSQL, vsftpd and mSQL. We randomly selected 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 recorded 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.

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

Table 2: **The time (in minutes) spent to write log statement based on the SECLOG's outputs.**

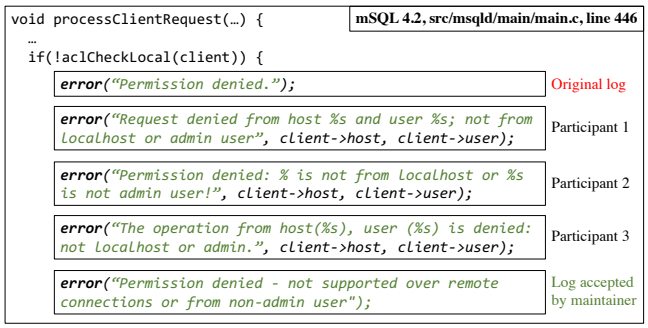

Figure 1: **The log statements written by the participants in mSQL-1 with SecLog's outputs.** Comparing with the log statement accepted by the software maintainers, all the log messages contain the same information, but are syntactically different.

Results In total, we recruited three developers to participate in our study. On average, the time required to write one log statement is 4.33 minutes, with minimum of 1.0 minute and maximum of 9.7 minute. The detailed results are shown in Table 2. 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 provided by the same set of SecLog-identified variables. For example, Figure 1 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.

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 5.

Case 1 (Apache): Figure 2 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

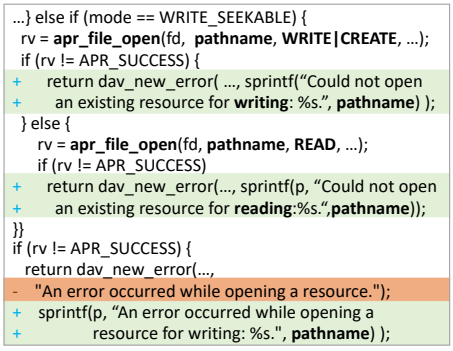

Figure 2: An accepted Apache patch based on SecLog-identified information adds the filename and action into log messages.

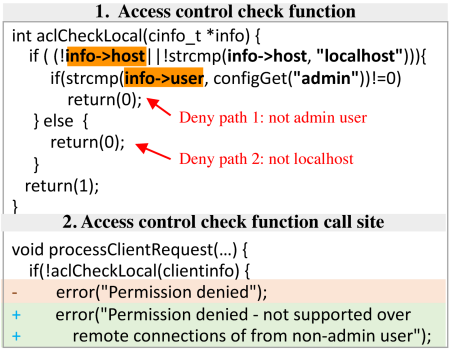

Figure 3: **The final accepted patch in mSQL adds information related to the subject.** SecLog finds the user (`info->user`) and host name (`info->host`) from two different deny paths. We reported the log message by adding SecLog-identified variables. The maintainers revised the wording based on our patch as shown above.

ServerFault [1] - 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 3 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 patch as shown in Figure 3.

Case 3 (vsftpd): Figure 4 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, so we added the configuration entry in the log message. We conducted the user study using this example (vsftpd-1).

An accepted patch which corrects the original log message by adding two access-deny reasons: The patch is shown in Figure 5. SecLog detects two different access-deny return points and constructs two slices with backwards

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 4: **An log enhancement for vsftpd, which is used in the user study problem vsftpd-1 in the user study problems.** SECLOG identifies relevant configuration variable `tunable_deny_file` and includes the configuration entry `deny_file` in the log message.

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 5: **Collecting access-deny information inside the access control check function.** With the result check function specifies `retval != ret_ok`, 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`.

slicing. The first sliced path extracts `mode` whereas the second one is related to `perm`. SECLOG enhance the log statement by completing both possible reasons of denial.

Global variables example in Figure 6: 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.

4 User study survey question example

4.1 Survey question for helpfulness of SECLOG-identified variables

To further understand the helpfulness of SECLOG-identified variables in diagnosing access-deny issues, we conducted a

```
vsf_privop_pasv_listen(struct session* p_sess){
  static struct vsf_sysutil_sockaddr* s_p_sockaddr;
  minport = max(1024, tunable_pasv_min_port);
  maxport = min(65535, tunable_pasv_max_port);
  ...
  the_port = random(minport, maxport);
  vsf_set_port(s_p_sockaddr, the_port);
  retval = vsf_util_bind(p_sess->fd, s_p_sockaddr);
  if (vsf_is_error(retval)) {
+    vsf_log_line_fail("Bind failed, port %d is in use,
+          check configuration entry:
+          pasv_max_port and pasv_min_port", the_port));
    die("vsf_util_bind");
  } ...
```

Data dependency

Figure 6: **An example of new log insertions that adds configuration entries of port range.** SECLOGtracks the data dependency of the arguments of the ACC functions at the call sites. This allows SECLOGadd unique variables per different call sites.

survey to quantitatively evaluate the helpfulness of the information in SECLOG-identified variables as discussed in Section **??**. We choose three commonly used applications, Apache, PostgreSQL and vsftpd, and randomly selected 10 access-deny program points enhanced by SECLOG. Only four scenarios are randomly drawn for each respondent.

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

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	○	○	○	○	○

4.2 User study survey question for helpfulness of whole log messages

As discussed in §**??**, after the participants finish all the questions, they can optionally participate in a short survey to re-

view and rate the original and SECLOG-enhanced log messages in each problem. 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	○	○	○	○	○

References

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