

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

The participants are asked to write the log statements with the following requirements: (1) The log statements should utilize the information in the variables. (2) The log statements should be human-readable and provide useful information for the system administrators. We do not require the participants to write log statements with the specific log functions in each application; instead, the participants only need to focus on the content of log messages and write the log message with `printf()` function. We record the time that each participant spent on each task. After the participant finish all the problems, we conduct a semi-structured interview to ask how they utilize the provided information to write the log statements and whether the information is easy to use.

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.

```

void processClientRequest(...) {
    ... mSQL 4.2, src/msqid/main/main.c, line 446
    ...
    if(!aclCheckLocal(client)) {
        error("Permission denied.");
    }
    error("Request denied from host %s and user %s; not from
          Localhost or admin user", client->host, client->user); Participant 1
    error("Permission denied: % is not from Localhost or %
          is not admin user!", client->host, client->user); Participant 2
    error("The operation from host(%s), user (%s) is denied:
          not Localhost or admin.", client->host, client->user); Participant 3
    error("Permission denied - not supported over remote
          connections or from non-admin user"); Log accepted
    by maintainer
}

```

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. We recorded the time and log statements written by the participants at each program location. On average, the time required to write one log statement is 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 all guided by the 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-denied 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 con-

```

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

```

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

```

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

2. Access control check function call site
void processClientRequest(...) {
    if(!aclCheckLocal(clientinfo))
        - error("Permission denied");
        + error("Permission denied - not supported over
          + remote connections of from non-admin user");
}

```

Figure 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.

firmed our suggestion and accepted our patch into its main branch. This patch can directly help an access-denied issue in 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 and includes the configuration entry in the log message. We conducted the user study using this example (vsftpd-1).

An accepted patch with two access-denied reasons: The patch is shown in Figure 5. SECLOG detects two different

```

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;
}

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`.

access-deny return points and constructs two slices with backwards slicing. The first sliced path extracts `mode` whereas the second one is related to `perm`. SECLOG enhance the log statement by adding two reasons for the denied access.

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-denied location.

```

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); ← Data dependency
    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");
    }
    ...
}

```

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

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>.