# CHAPTER SIXTEEN

# THE LOG ANALYSIS SCRIPT

## Global Variable Definition

First, we will define some **global Setup** variables that will govern the usage of this script. The reason that we do not include these variables in the general **web_store.setup** file is that there is the potential that on some servers, because the Setup file must be readable by the Web server, the Setup file is insecure.

Thus, we would not want to include the password there. Instead, you will hardcode the password in the script to provide the highest degree of security. At the same time, we will define other script specific variables as well.


Consider what would happen if a sneaky competitor studied up on this application and used her own Web browser to access a **log_analysis.setup** file (if one existed). If you left this file with the default filename, they would easily be able to type in a URL such as:

```
http://www.yourdomain.com/cgi-bin/Web_store/Library/
log_analysis.setup
```

If they did that, the Web server might serve up the Setup file as plain text (if your Web server is configured to allow that). With the Setup file in their browser window, they could easily scan down to determine the password. Then they could use your log analysis scripts to check up on your sales and customers!

Though it makes customizing the application a little less obvious, by placing the password in the script itself we make it impossible for a hacker to glean the password, because when she calls the script from the location window, she will execute the script rather than see its contents displayed.

- **$sc_password** is the password that you will need to enter on the Web form in order to receive a log analysis.
- **$sc_log_file_directory** is the path location of the directory containing the log files.
- **$sc_error_log_path** is the entire path name of the error log.
- **$sc_access_log_path** is the entire path name of the access log.
- **$sc_cgi_lib_path** is the location of **cgi-lib.pl**.
- **$sc_db_lib_path** is the location of **web_store_db_lib.pl**.
- **@sc_db_query_criteria** is the array of search string options you will use for searching the log files. For this default example, we will just allow string equality searching on every environment variable field and will use a text input field in the HTML form using **NAME = "keywords"**.


NOTE

If you do not understand the search definitions, you might want to read through the discussion of keyword searching in Chapter 5.

```
$sc_password = "selena";
$sc_log_file_directory = "./Admin_files";
$sc_error_log_path = "$sc_log_file_directory/error.log";
$sc_access_log_path = "$sc_log_file_directory/access.log";

$sc_cgi_lib_path = "./Library/cgi-lib.pl";
$sc_db_lib_path = "./Library/web_store_db_lib.pl";

@sc_db_query_criteria = ("keywords|0,1,2,3,4,5,6,7,8,9,10,11,12,13,
                          14,15,16,17,18,19,20,21,22|=|string");
```

# Main Routine

First, Perl is told to bypass its own buffer so that the information generated by this script will be sent immediately to the browser.

```
$| = 1;
```

Then, the HTTP header is sent to the browser. This is done early for two reasons. First, it will be easier to debug the script while making modifications or customizing because we will be able to see exactly what the script is doing. Secondly, the HTTP header is sent out early so that the browser will not "time out" in case the script takes a long time to complete its work.

```
print "Content-type: text/html\n\n";
```

Next, both of the supporting files are read in. **cgi-lib.pl** will be used for reading and parsing incoming form data and **web_store_db_lib.pl** will be used to search the log files for keywords.

```
require "$sc_cgi_lib_path";
require "$sc_db_lib_path";
```

Then, the incoming form data is read and parsed by **cgi-lib.pl** and **$log_file_in** and **$password_in** are defined according to the values coming in as form data. **$log_file_in** will be equal to the name and location of the log file that the script has been requested to analyze. **$password_in** will be equal to the password submitted via the form.

```
&ReadParse(*form_data);
$log_file_in = "$sc_log_file_directory/$form_data{'which_log'}";
$password_in = "$form_data{'password'}";
```

# Display Search Results

Now the script determines if the client submitting the password and log files has entered the correct values, because it will only display the correct error and access logs and will do so only if the client submits the right password.

```
if (($password_in eq "$sc_password") && (
     $log_file_in eq "$sc_error_log_path") ||
    ($log_file_in eq "$sc_access_log_path"))
  {
```

If the incoming form data passed the security check, the script opens the log file requested for reading and begins displaying the HTML response page using **log_analyzer_return_header** located at the end of this file.

```
open (LOG_FILE, "$log_file_in") || &CgiDie("Sorry,
      could not open the requested log file.  Please
      check the permissions and the path.");
&log_analyzer_return_header;
```

Next, the script goes through the log file one line at a time, splitting each line into its fields (every log file database row is pipe-delimited and each is separated by a newline). **$not_found** will also be set to zero so that we will be able to check at the end of this routine if we have actually found some hits in the log file, based on the client-submitted keywords.

```
  while (<LOG_FILE>)
    {
    @fields = split (/\|/, $_);
```

We start off stating that no criteria were found.

```
    $not_found = 0;
```

The information in the fields is then compared to the keywords submitted by the client. Each criteria in the **@sc_db_query_criteria** array is specifically applied to the database row for searching.

In the loop below, if any criteria is not found, the result is an incrementing of **$not_found**.

```
foreach $criteria (@sc_db_query_criteria)
  {
  $not_found += &flatfile_apply_criteria(
                $exact_match,
                $case_sensitive,
                *fields,
                $criteria);
      }
```

If the script found some hits, it displays them in table format.

```
    if ($not_found == 0)
      {
      print "<TR>";
      foreach $field (@fields)
        {
        print "<TD>$field</TD>";
        }
      print "</TR>";
      }
    }
```

Then, the script displays the HTML footer and exits.

```
  &log_analyzer_return_footer;
  close (LOG_FILE);
  }
```

# Display Query Form

If the client-supplied information does not pass the security check, or the script is being accessed for the first time, the script displays the form that can be used to submit keywords, a password, and a log file to analyze.

```
else
  {
  &log_analyzer_query_form;
  }
```

# log_analyzer_query_form Subroutine

**log_analyzer_query_form** is used to generate the form used by the administrator to select which log file to view and which keywords to filter with. It is called with no arguments:

```
sub log_analyzer_query_form
  {
  print qq~
  <HTML>
  <HEAD>
```

```
<TITLE>Web Store Log File View</TITLE>
</HEAD>
<BODY BGCOLOR = "FFFFFF" TEXT = "000000">
<CENTER><H2>Basic Log Analyzer</H2></CENTER>
<BLOCKQUOTE>
<FORM ACTION = "web_store_log_analysis.cgi"
      METHOD = "post">
<TABLE>
<TR>
<TH ALIGN = "left">Password (Required)</TH>
<TD><INPUT TYPE = "text" SIZE = "20" MAXLENGTH = "20"
           NAME ="password"></TD>
</TR>
<TR>
<TH ALIGN = "left">Log File to View</TH>
<TD><SELECT NAME = "which_log">
    <OPTION VALUE = "error.log">Error Log
    <OPTION VALUE = "access.log">Access Log
    </SELECT></TD>
</TR>
<TR>
<TH ALIGN = "left">Search Term
                   (None for entire file)</TH>
<TD><INPUT TYPE = "text" SIZE = "20" MAXLENGTH = "20"
           NAME ="keywords"></TD>
</TR>
</TABLE>
<CENTER>
<INPUT TYPE = "submit" NAME = "submit"
       VALUE = "View Log">
</CENTER>
</FORM>
</BLOCKQUOTE>
</BODY>
</HTML>~;
}
```

# log_analyzer_return_header Subroutine

**log_analyzer_return_header** is used to display the HTML header for the Log File View page. It is also called with no arguments:

```
sub log_analyzer_return_header
  {
  print qq!
  <HTML>
  <HEAD>
  <TITLE>Web Store Log File View</TITLE>
  </HEAD>
  <BODY BGCOLOR = "FFFFFF" TEXT = "000000">
  <TABLE BORDER = "1">!;
  }
```

# log_analyzer_return_footer Subroutine

**log_analyzer_return_footer** is used to display the HTML footer for the Log File View page. It is called with no arguments:

```
sub log_analyzer_return_footer
  {
  print "</TABLE></BODY></HTML>";
  }
```