

Using DLCGI to achieve single sign-on with The Diver Solution

In situations where authentication to DiveLine needs to be integrated with an existing authentication scheme, we provide "DLCGI", the DiveLine-Common Gateway Interface interfacing module. The "Common Gateway Interface" is a standard for interfacing external scripts and programs with a web server.

How DLCGI works

When `dlcgi.exe` is executed by the webserver, in the context of a user that the web server has already authenticated, it obtains a limited-lifetime one-time password from DiveLine. This password can be passed, via web page redirects, custom web page scripting, etc., to DivePort, NetDiver, or even ProDiver to allow the user to login.

The typical strategy for using DLCGI is:

1. Configure DiveLine to accept DLCGI requests from your webserver.
2. Install `dlcgi.exe` in a scripts directory (e.g. `/cgi-bin/`) on your CGI-compliant webserver (e.g. IIS, Apache). You configure the name of your DiveLine server and other parameters using `dlcgi.cfg` in the same directory as the executable.
3. Restrict access to this script so that the webserver will only execute it when the user has already authenticated (e.g. Domain account).

Typical uses

- DivePort: Users go to the DivePort site, and are redirected to another URL for authentication. That URL, which runs `dlcgi.exe`, redirects the user back to the DivePort URL with a one-use authentication token.
- ProDiver: When ProDiver connects to DiveLine, if configured with a DLCGI URL, it will access the URL in "raw" mode (see below) to obtain a parse-able result file containing a one-use DiveLine authentication token. This uses Internet Explorer components embedded in the Operating System for web server access (and cookie sharing).
- The templating support also permits "click on this link to start [client], authenticated" `<a>` elements on webpages.

Non-CGI environments

If you are operating in an environment where the web server controlling authentication does not support CGI scripts, you need to do something different. Essentially, you need to write a wrapper around dlcgi.exe that is compatible with your server platform.

You need to create a script / servlet / etc. which handles a URL and will act like the DLCGI URL we normally use. In its simplest form, the script must:

- Perform authentication (if not already handled by the platform at the point it is executing).
- Setup some environment variables (below) to emulate the CGI interface.
- Invoke dlcgi.exe.
- Send dlcgi's output unprocessed as the result of the request.

Advanced forms would include things like dynamically constructing the resulting page, rather than relying on dlcgi's templating and AJAX options.

Technical information about invoking dlcgi.exe

When dlcgi starts, it expects information to be passed to it in accordance with the standard Common Gateway Interface. In some cases, you may need to specify some of this information manually, depending on how dlcgi.exe is invoked. The following information is passed to DLCGI via environment variables:

- **REMOTE_USER** = name of DiveLine user to grant access to (usually matches username used to login on website).
- **QUERY_STRING** = used in advanced/debugging uses of dlcgi. "raw" will have dlcgi print the one-time username and password in a simple format which can be parsed by a program. It will also include debugging/configuration information.
- **PATH_INFO** = name of a template file to use. Templates are text files (usually .html) which dlcgi will substitute-in the username and password into, to allow execution of DivePort, NetDiver, etc. For instance, the DivePort template file just redirects the user to DivePort, passing the one-time authentication token (user/pass) in the process.

During development, it is possible to set these environment variables and run dlcgi.exe from a command-prompt to see what sorts of output it generates. The output is always a webpage to display, except in "raw" mode which produces a plain text file.

Wrapping the DL CGI script for other environments

Here are some hints on wrapping the execution of our CGI script to fit other execution environments. This would be needed anytime the webserver either isn't CGI compliant, or the built-in web server authentication is not being used, but rather some other system is in use. Note that these are not fully tested programs, just suggestions.

1. Create a server-side script in the technology appropriate to your webserver (e.g., ASP, ASP.NET, PHP, Java, etc.). This script will be executed by the web server in response to a request for a particular URL, defined by you, which we will call the "Start DivePort URL." For example: <https://yourserver/scripts/run-diveport.asp>

Be sure to put the script at an appropriate place in the file system on the web server to match the desired URL.

2. Make sure that access to the script, via its URL, is treated as protected content by your web server / portal system. That is, attempts to access the URL directly without logging in first will prompt you to login.
3. Your script should perform the following tasks:
 - A) Lookup the username associated with the current web server session. This is the part where you interface with your own authentication scheme.
 - B) Fetch the query string from the URL in the HTTP request that the script is being called to handle. We will pass this through unchanged to DL CGI via an environment variable when we execute it.

In ASP:

```
Request.ServerVariables("QUERY_STRING")
```

In PHP:

```
$_SERVER['QUERY_STRING']
```

In a Java Servlet:

```
javax.servlet.http.HttpServletRequest's getQueryString() method
```

- C) Fetch the extra path information from the URL in the HTTP request that the script is being called to handle. We will pass this through unchanged to DL CGI via an environment variable when we execute it.

In ASP:

```
Request.ServerVariables("PATH_INFO")
```

In PHP:

```
$_SERVER['PATH_INFO']
```

In a Java Servlet:

```
javax.servlet.http.HttpServletRequest's getPathInfo() method
```

D) Invoke the "dlcgi.exe" (or just "dlcgi" on Unix) executable. You need to set some environment variables for the execution of this script, in compliance with the W3C's Common Gateway Interface (CGI) specification (see following tips).

Namely:

REMOTE_USER - the username determined in step A.

QUERY_STRING - the query string determined in step B.

PATH_INFO - the extended path information determined in step C.

The output of running dlcgi (which goes to the standard output stream) is the HTML web page your script should produce.

Tips on running external executables with environment variables

In ASP:

Look into something like this pattern (or better yet, use ASP.NET's System.Diagnostics.Process)

```
Set shellObj = Server.CreateObject("WScript.Shell")
Set environment = shellObj.Environment("PROCESS")
environment("REMOTE_USER") = username
environment("QUERY_STRING") = query_string
environment("PATH_INFO") = path_info
Set cmdObj = shellObj.Exec("C:\Path\to\dlcgi.exe")
response.write cmdObj.StdOut.ReadAll()
```

In PHP, use this pattern:

```
putenv("REMOTE_USER=$username");
putenv("QUERY_STRING=$query_string");
putenv("PATH_INFO=$path_info");
system("C:\\path\\to\\dlcgi.exe");
```

In a Java servlet:

See `java.lang.Runtime.exec()`, or better, in Java 1.5 and newer, `java.lang.ProcessBuilder`. Both have mechanisms for passing environment variables to a program you wish to execute.

Testing DLCGI from the command-line

SETUP

- 1) Determine the name and port number of your DiveLine server, and the base URL for the Portal you wish to access.
(Example: DiveLine: <server>.org:2130
DivePort: https://<server>/PPA)
- 2) Configure that DiveLine to accept DLCGI "gateway" requests from the IP address of the machine you are testing from. Modify DiveLine's atlcfg.cfg -- put this value in the "main" section:

```
gateway_ips = {  
    "hostname-or-ip-address",  
},
```

- 3) Configure dlcgi.cfg.
Place dlcgi.exe in a directory and create a text file called dlcgi.cfg in the same directory with content similar to this:

```
version "1";  
// Computer generated object language file  
object 'DCFG' "main" {  
    diveline="<server>.org:2130",  
    template_directory="C:\\PATH\\TO\\TEMPLATEDIR",  
};
```

Correct the DiveLine entry. The template directory will be the directory that contains templates; it can be wherever you want, so long as dlcgi.exe can read it. Note the use of two backslashes in path elements.

- 4) Choose a username which is defined on DiveLine to test with.
Example: "di_tester".

RAW MODE

- 5) Set the username. From a command prompt, enter something like this:

```
SET REMOTE_USER=di_tester
```

- 6) Select "raw" mode, but no template. From the same command prompt:

```
SET QUERY_STRING=raw  
SET PATH_INFO=
```

- 7) Run dlcgi.exe. From the same command prompt:

```
dlcgi
```

You should get output in a computer-parsable format showing for example,

di_tester and a one-time password. You can now use this password, once, for the next 2 minutes, in ProDiver or DivePort to gain access as this user.

TEMPLATE MODE

8) Setup a template. For example, see "diveport.html" in DivePort's dlcgi directory. Note how it contains \$DL_USER and \$DL_PASSWORD. dlcgi will read the template in, perform \$ substitutions, and then write the result to standard output.

Put a template in the above-chosen templates directory. For example, put the supplied diveport.html there.

9) Set the username. From a command prompt, enter this:

```
SET REMOTE_USER=di_tester
```

10) Select the template mode, but not raw mode. From the same command prompt:

```
SET PATH_INFO=/diveport.html  
SET QUERY_STRING=
```

11) Run dlcgi.exe. From the same command prompt:

```
dlcgi
```

The result should be the contents of the diveport.html file with the one-time username and password plugged in. If this were returned to the user's web browser, the browser would redirect them to DivePort, passing the username and one-time password as parameters in the URL.

Note that other text files could be used; for instance, a JSON-style object with \$DL_USER and \$DL_PASSWORD, if you'd rather handle the redirect manually but not write code to parse the "raw" output.