Tivoli software

IBM

IBM Tivoli Access Manager for e-business 6.1: Writing External Authentication Interface Servers

White Paper

Ori Pomerantz (orip@us.ibm.com)

July 2010

Table of contents

Introduction

About this paper

This document explains the capabilities of the external authentication interface (EAI) for Tivoli® Access Manager for e-business, and how to write EAI servers that use that interface.

Audience

This paper is for people who need to write EAI servers to extend the capabilities of Tivoli Access Manager for e-business authentication. The assumption is that readers are programmers, but no knowledge of a particular programming language is assumed. The examples are in Perl and PHP, but the syntax is explained where it differs from other C based languages.

A secondary audience is architects whose job includes deciding whether EAI servers can solve a particular problem. The first part of the white paper explains the capabilities of EAI.

White paper

1 Reasons for using the external authentication interface

The external authentication interface (EAI) allows WebSEAL to outsource authentication decisions to a separate module. This technique enables additional functionality beyond what WebSEAL is designed to do.

1.1 EAI process flow

EAI is a mechanism to outsource the responsibility for authentication from WebSEAL to a third-party product. The way it works is shown in the following diagram.

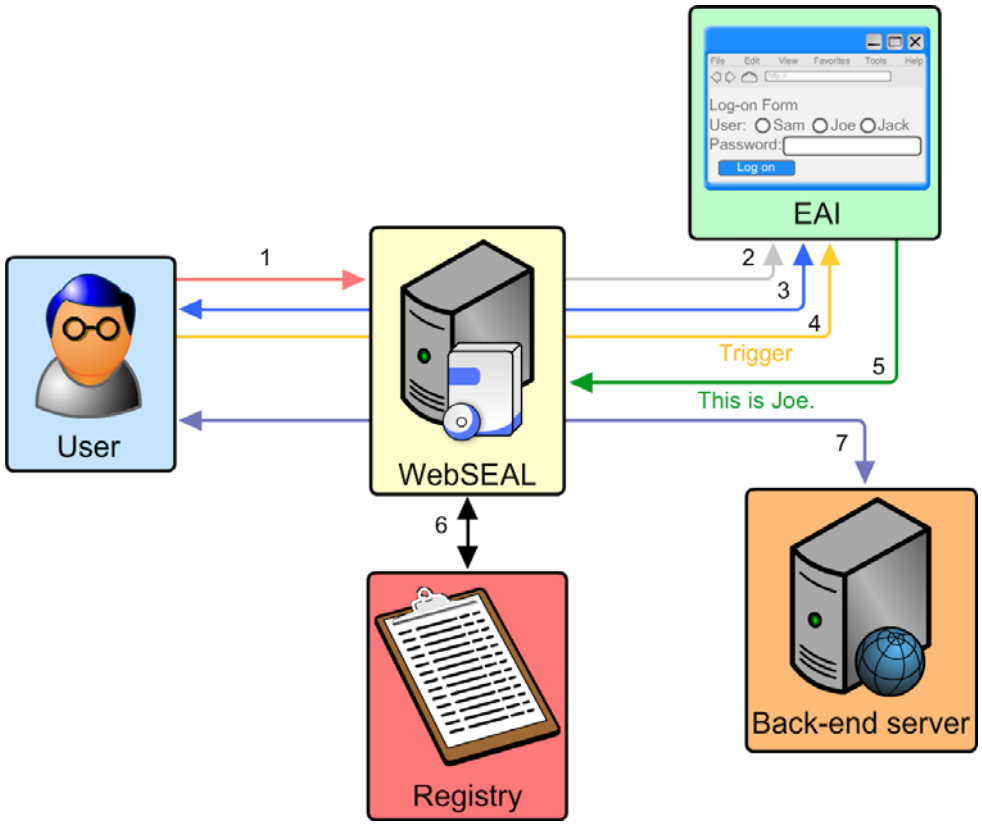

Figure 1: EAI process flow

The diagram describes the following process flow:

1. The user attempts to connect to the EAI server, which may be on a separate computer from WebSEAL.
2. WebSEAL allows unauthenticated access to the EAI server. This is necessary, because the user is not authenticated at this point.
3. The user and the EAI server communicate. This communication can be as long and as involved as necessary.
4. The user, based on an HTML page from the EAI server, retrieves a ***trigger URL***, which is a URL that is configured in WebSEAL as one that might contain the EAI output.
5. The EAI server sends back a reply, which has an HTTP header that contains the user identity and possibly additional information.
6. WebSEAL creates the credential for the user.
7. WebSEAL allows the user to access a back-end server.

1.2 Exotic authentication

IBM Tivoli Access Manager for eBusiness supports a large number of authentication mechanisms, such as passwords and RSA SecureID. If the organization uses a more exotic mechanism, negotiating with a third-party server is required. Using EAI, this mechanism can be as complex as necessary.

1.3 Extended attributes

In addition to providing the user identity, EAI can provide extended user attributes that WebSEAL can later use in authorization decisions.

1.4 Step-up authentication

EAI can also provide an authentication level. This level allows for ***step-up authentication***, a system in which some web pages require a stronger authentication mechanism than others.

2 Configuring WebSEAL for EAI

This section explains the steps that are required to configure WebSEAL for EAI support.

2.1 WebSEAL configuration file modifications

The WebSEAL instance configuration file (**/opt/pdweb/etc/webseald-default.conf** for the default instance under UNIX and Linux) requires some modifications for EAI.

2.1.1 Add the authentication mechanism library

The list of libraries used for authentication is in the **[authentication-mechanisms]** stanza of the WebSEAL configuration file. To enable EAI, add the following line (all on one line):

```
ext-auth-interface = /opt/pdwebrte/lib/libeaiauthn.so
```

2.1.2 The [eai] stanza

As with the other the WebSEAL authentication mechanisms, a setting within the stanza controls whether the mechanism can be used at all. To use EAI for HTTPS connections, use this line to set the **eai-auth** value:

```
eai-auth = https
```

Most of the remainder of the stanza specifies the names of fields in the HTTP header. Those fields are interpreted by WebSEAL when the EAI server responds with the answer. If you are writing your own EAI server, keeping the default values is the easiest approach.

2.1.3 The [eai-trigger-urls] stanza

This stanza specifies the trigger URLs. A ***trigger URL*** is a URL whose response can include the EAI server's reply in HTTP headers. Trigger URLs can also be specified using a wildcard pattern.

For example, if the junction is called **/eailogin** and the response that includes the authentication will be received from a Perl script that has a name starting with **eai** in the **cgi-bin** directory, the trigger may be specified as follows:

```
trigger = /eailogin/cgi-bin/eai*.pl
```

Important: After you are done editing the WebSEAL configuration file, remember to restart the WebSEAL instance.

2.2 Server junction and access control list

WebSEAL sees the EAI server as another HTTP server. To allow users to access this HTTP server, WebSEAL requires a junction.

Use the following **pdadmin** command to create the junction. Note that the command is all one line.

```
s t <instance>-webseald-<webseal computer> create -t tcp
      -h <eai computer> /eailogin
```

Users are unauthenticated while they are communicating with the EAI server. To allow unauthenticated access, run the following pdadmin commands. Ignore error message HPDAC0757E about ACL permissions when you get it.

```
acl create eaiacl
acl modify eaiacl set any-other Trx
acl modify eaiacl set unauthenticated Trx
acl attach /WebSEAL/<webseal computer>-<instance>/eailogin eaiacl
```


3 Simple EAI server

This section explains how to write a simple EAI server, how to debug it, and how to integrate it with WebSEAL.

3.1 The cgi-bin script

This section shows a simple EAI server, written in Perl. To use this server, put it in a **cgi-bin** directory on an Apache Web server (or IBM HTTP Server). Give it a descriptive name, such as **eai-simple.pl**. The file is as follows:

```
#! /usr/bin/perl
```

This line specifies the name of the interpreter. It is necessary for Linux scripts.

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

The first output line of a cgi-bin script provides the content type. In this case, the content is text-encoded in HTML.

```
print “am-eai-user-id: sam\n”;
```

This line provides the used identity, which must be a user ID that exists in IBM Tivoli Access Manager for e-business. This simple script hard codes the user identity, so it is always **sam**.

```
print “am-eai-redir-url: /ihs\n”;
```

This line specifies the URL to redirect the user after successful authentication. This URL is used when the EAI server is invoked directly, rather than by redirection after an unauthenticated user attempts to access a URL that requires authentication.

```
print “\n”
```

This line results in an empty line, which separates the fields in the HTTP header from the HTML according to the cgi-bin specifications.

```
print “<HTML><BODY>”;
print “<H2>Something Went Wrong</H2>”;
print “If the EAI was working correctly, you would not see this.”;
print “</BODY></HTML>”;
```

If the EAI server functions correctly, the user is authenticated by WebSEAL and redirected to a different Web page. This is HTML to display to users as an error message in case of error.

3.2 Running the EAI server manually

To check the EAI server, connect to it the way WebSEAL would. Most UNIX and Linux systems have the **netcat** command (**nc**) for that purpose. The syntax is as follows:

```
nc <eai computer> 80
```

Then, provide input similar to what a web browser or WebSEAL would provide. Make sure to send a blank line to signal the end of the HTTP request.

```
GET /cgi-bin/<script name> HTTP/1.1
Host: whatever
```

The output is identical to the output you would get from running the program on the server itself.

3.3 Using the EAI server

With all the pieces in place, you may now actually test the EAI server. To do this, open a new browser instance and type the following URL:

```
https://<webseal computer>/iealogin/cgi-bin/eai-simple.pl
```

If everything is working correctly, you are logged in automatically as the user (**sam**, unless you changed it), and redirected to another URL (**/ihs**, unless you changed it).

3.4 Automatic redirection

When unauthenticated users attempt to log on to a web page, they are normally asked for authentication. One method to automatically ask for EAI authentication is to enable forms authentication, disable basic authentication, and add the following lines to the login form file (for the default instance, that file is usually **/opt/pdweb/www-default/lib/html/C/login.html**).

```
<SCRIPT LANGUAGE=JavaScript>
window.location.replace(
      “https://<webseal computer>/iealogin/cgi-bin/eai-simple.pl”);
</SCRIPT>
```


4 Negotiating authentication

In real life, an EAI server cannot just specify a fixed-user identity. Users must provide their identities to the EAI server, usually by filling out a web form. A more realistic EAI module is one that asks the user for an identity and a password. It is composed of two files: **cgi-bin/eai-mkform.pl**, which creates the log on form; and **cgi-bin/eai-useform.pl**, which authenticates the user based on the response to that form.

4.1 The eai-mkform.pl script

This simple cgi-bin script displays a form with a list of users. Only the new parts are explained in this section.

```
#! /usr/bin/perl

@users = (“sam”, “joe”, “jack”);
```

This line creates a list called **@users**, and initializes it to the three user names.

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

There is no need for any HTTP header fields except the content type.

```
print “<HTML><HEAD><TITLE>Log-on Form</TITLE></HEAD>\n”;
print “<BODY>\n”;
print “<H2>Log-on Form</H2>\n”;
print “<FORM ACTION=\”eai-useform.pl\” METHOD=POST>\n”;
```

The last line specifies the beginning of an HTML form. It also specifies that the form result should be submitted to **eai-useform.pl** file (in the same directory as **eai-mkform.pl** file). The method is POST, meaning that the results are passed by the browser as HTTP header fields. The other method, GET, passes the results as part of the URL (that is the part after a question mark in a URL).

The \” notation escapes the quotation mark, so the Perl interpreter can understand that the quotation mark is part of the string to output, rather than the string terminator.

```
print “User:\n”;
foreach (@users) {
```

This line starts a ***foreach loop***. This kind of loop iterates on all the values in a list.

```
    print “<INPUT TYPE=RADIO NAME=uid VALUE=\”$_\”>”
```

This line specifies a ***radio button input***, an input field that allows the user to choose between multiple values. The value that the browser returns in the **uid** field if this particular radio button is selected is the one stored in **$_**, which is the current value. In this case, **$_** is the current value from the list in the **foreach** statement in the previous line, a member of the **@users** list.

```
    print “$_</INPUT>\n”;
```

The value between the **<INPUT>** and **</INPUT>** tags is presented to the user by the browser. Here it is also **$_**, the name of the current user in the list.

```
}
```

Perl uses the same block syntax as C, C++, and Java. Blocks start and end with curly braces (**{** and **}**).

```
print "</BR>Password: <INPUT TYPE=PASSWORD NAME=pwd \></BR>\n";
```

This line is a password field. It is a text field, where the typed value is replaced with symbols to preserve privacy.

```
print "<BUTTON TYPE=SUBMIT>Log on</BUTTON>\n";
```

When this button is clicked, the form is submitted by the browser.

```
print "</FORM>\n";
print "</BODY></HTML>";
```

4.2 The eai-useform.pl script

This is the script that uses the filled form to log on the user

4.2.1 Reading the form data

The first thing the script needs to do is read the form data.

4.2.1.1 POST data in cgi-bin scripts

When a form uses the POST method, the data is provided to cgi-bin scripts in their standard input, as one line. The various fields are separated by ampersand (**&**) characters, and the field name is separated from the value with an equal sign (**=**). Characters that have special meanings are encoded either at **%<*ASCII value in hexadecimal*>** or a plus sign (**+**) for space.

For example, if the user name is **joe** and the password is **joe's +password**, the input line is as follows:

```
uid=joe&pwd=joe%27s+%2Bpassword
```

4.2.1.2 Parsing form data in Perl

The code fragment in this section reads and parses the form data:

```
use URI::Escape;
```

This line specifies that the Perl program uses functions from the **URI::Escape** package. This line is similar to the **#include** directive in C.


```
$line = <STDIN>;
```

This line reads a line from the process standard input. The line is then stored in a variable called **$line**.

```
@vars = split(/&/, $line);
```

This line takes the **$line** variable, and splits it into an array called **@vars**. The **$line** variable is split according to the regular expression **/&/**, meaning that every item of **@vars** is one of the form variables.

```
foreach $var (@vars) {
```

This line starts a loop. This loop is executed once for each value in **@vars**, with the value stored in **$var** (if you do not specify a variable, the value is stored in **$_**).

```
    ($field, $value) = split (/=/, $var);
```

This line splits **$var** at the equal sign (=). In contrast to the line that split **$line**, in this line, the result array is assigned into two variables: **$field** for the first array entry and **$value** for the second one.

```
    $value =~ s/\+/ /g;
    $value = uri_unescape($value);
```

These two lines modify the **$value** variable to return to the original value submitted by the user. The first line changes plus signs (+) to spaces. The second uses the **uri_encode** function from the **URI::Escape** package to change the characters encoded as ASCII values back to their original form.

```
    $form{$field} = $value;
```

This line manipulates an ***associative array*** (an array indexed by strings rather than integers) called **%form**. It sets the value for the string **$field** to be **$value**. Associative arrays are identified by the curly braces (**{}**) surrounding the index.

```
}
```

This line closes the **foreach** block.

4.2.2 Verifying user identities

When you write the part of the script that verifies the user identity, it is important to remember that the code in cgi-bin scripts runs under the same user as the web server. In the case of Apache (and related servers, such as IBM HTTP Server), you can view the **User** and **Group** settings in the **conf/httpd.conf** location. The default value for both is **nobody**.

Listing all the various ways that users can be authenticated and explaining how to do them in Perl is not possible in this paper. In many cases, the easiest way to interface with a separate system is to download a module from CPAN (**http://www.cpan.org**).

4.2.3 The script itself

This section shows the complete eai-useform.pl script. Only the new parts are explained here.

```
#! /usr/bin/perl

use URI::Escape;

# Read the post data
$line = <STDIN>;
@vars = split(/&/, $line);

foreach $var (@vars) {
      ($field, $value) = split(/=/, $var);
      $value =~ s/\+/ /g;
      $form{$field} = uri_unescape($value);
}

# Verify user identity (in an extremely naive way)
if ($form{“pwd”} eq “object00”) {
```

The **eq** operator in Perl checks for string equality. In this case, it checks whether the password submitted by the user is equal to **object00**.

```
      $user = $form{“uid”};
}
```

In Perl, a block is required after an **if** statement. Even if there is only one command, as here, that command must be enclosed in curly braces (**{}**).

```
print “Content-type: text/html\n”;
if ($user ne “”) {
      print “am-eai-user-id: $user\n”;
}
```

The **ne** operator is used to check whether strings are *not* equal. In this case, it checks that **$user** is not equal to the default value, which is the empty string. If it is not equal, the script provides a user identity to WebSEAL. If **$user** is still empty, then no user identity is provided. In such a case, WebSEAL returns the HTML to the user.

```
print “am-eai-redir-url: /ihs\n”;
print “\n”;

# HTML header for authentication failure
```

When authentication succeeds, the user is redirected to the **am-eai-redir-url** value and does not see this HTML message.

```
print “<HTML><HEAD><TITLE>Authentication Failure</TITLE></HEAD>”;
print “<BODY>\n”;
print “<H2>Authentication Failure</H2>\n”;

print “If you really are $form{‘uid’}, “;
```


The string is enclosed in double quotation marks (**"**). One method to use quotation marks inside such a string, for example for an index to an associative array, is to use single quotation marks (**'**).

```
print "<A HREF=\"eai-mkform.pl\"> try again.</A>\n";
```

Another method is to use a backslash (\) before the double quotation mark.

```
print "</BODY></HTML>\n";
```

5 Writing EAI servers in PHP

PHP is a programming language that was specifically designed to develop web sites. It allows developers to write normal HTML, and put code for server execution between **<?php** and **?>** characters.

Important: Before testing the PHP EAI server, remember to update the triggers in the WebSEAL configuration file, as explained in "The [eai-trigger-urls] stanza" on page 4.

5.1 The form, eai-form.php

The **eai-form.php** file contains the logon form. It is similar to the Perl version, but considerably shorter.

```
<HTML>
<HEAD><TITLE>Log-on Form</TITLE></HEAD>
<BODY>
<H2>Log-on Form</H2>
<FORM ACTION="eai-useform.php" METHOD=POST>
```

Most of the file is standard HTML. Only the parts that need to be coded are interpreted on the server, resulting in better performance.

```
<?php
```

This line starts the PHP code.

```
$users = array("sam", "joe", "jack");
```

This line creates an array called **$users**, and initializes it to a list of three users. Notice that the syntax is similar to Perl, but not identical. If you are used to Perl syntax, make sure to review the PHP documentation for the differences.

```
foreach ($users as $user) {
```

This is a **foreach** loop, similar to the same construct in Perl. The variable **$user** gets each of the values in the **$users** array.

```
    print "<INPUT TYPE=RADIO NAME=uid VALUE=\"$user\">";
    print "$user</INPUT>";
```

```
}
```

This part is nearly identical to the way it would be written in Perl.

```
?>
```

This line ends the PHP code.

```
</BR>
Password: <INPUT TYPE=PASSWORD NAME=pwd \>
</BR></BR>
<BUTTON TYPE=SUBMIT>Log on</BUTTON>
</FORM></BODY></HTML>
```

5.2 The log on page, eai-useform.php

The eai-useform.php file processes the form data to either log the user on, or reject the log on. It is also similar to the Perl version.

```
<?php
```

This line starts the PHP code. The PHP code has to appear in the beginning of the file here because after the HTML starts it is impossible to add header fields, such as the fields used for EAI log on.

```
$user = $_POST[“uid”];
$password = $_POST[“pwd”];
```

$_POST is an associative array that includes all the form fields. In contrast to Perl, PHP handles the input processing automatically for the programmer. These lines read the user name and password and assign them to variables.

```
if ($password == “object00”) {
      header(“am-eai-user-id: $user”);
      header(“am-eai-redir-url: /ihs”);
}
```

These lines check whether the password is **object00**. If so, the **header()** function adds the appropriate lines to the HTTP header.

```
?>

<HTML>
<HEAD><TITLE>Authentication Failure</TITLE></HEAD>
<BODY>
<H2>Login Error</H2>
If you are really
<?php print $user ?>,
```

This is another PHP code fragment. Notice that the variables from one PHP code fragment are available in all the other fragments in the same file.

```
<A HREF=”eai-form.php”>try again</A>.</BR>

</BODY></HTML>
```


6 Sending extended attributes

EAI can send WebSEAL ***extended attributes***, which are attributes that are added to the credential stored in WebSEAL. These extended attributes can then affect authorization decisions made by authorization rules.

6.1 Sending extended attributes from EAI to WebSEAL

EAI servers send extended attributes to WebSEAL in the HTTP header. By default, the list of HTTP header fields to use as extended attributes is provided in the HTTP header field **am-eai-xattr**. The attributes and their values are also provided in the HTTP header. For example, this portion of an HTTP header specifies two attributes, **clearance** and **rank**, along with their values

```
am-eai-xattr: clearance, rank
clearance: secret
rank: captain
```

6.2 Using the extended attributes in an authorization rule

For extended attributes to affect authorization decisions, a rule must exist that uses them. Those rules are called ***authorization rules***, and they are written in XSL. You can learn how to write authorization rules at the following URL:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc_6.1/am61_admin232.htm.

To create a rule that requires the **clearance** attribute to be **secret** or **top**, create a text file with the following content:

```
<xsl:if test='clearance=secret'>!TRUE!</xsl:if>
<xsl:if test='clearance=top'>!TRUE!</xsl:if>
```

Then, run the following pdadmin commands to create the rule and attach it:

```
authzrule create secret -rulefile <name of rule file>
authzrule attach <object name> secret
```

7 Step-up authentication

WebSEAL allows for web resources to require different authentication levels. This technique is useful, for example, if a particular application is very sensitive and requires two-factor authentication. An EAI server can report the level at which it authenticated a user.

7.1 WebSEAL configuration file modifications

To enable step-up authentication, edit the WebSEAL configuration file. Open the **[authentication-levels]** stanza. The first level, level zero, is normally **unauthenticated**. After that, set as many levels as you need for **password**. Then, you can use redirection from the login form to the EAI server.

For example, if a system uses three authentication levels, the **[authentication-levels]** stanza includes the following lines:

```
level=unauthenticated
level=password
level=password
level=password
```

Important: After you are done editing the WebSEAL configuration file, remember to restart the WebSEAL instance.

7.2 Resource configuration

To require a higher level of authentication for a particular resource, create a protected object policy (POP) that requires it, and then apply the POP to the object. To set the authentication level of a resource to 2, run the following pdadmin commands:

```
pop create authlvl2
pop modify authlvl2 set ipauth anyothernw 2
pop attach <object name> authlvl2
```


7.3 Redirection

Step-up authentication uses a logon form that differs from regular authentication. You can modify the step-up logon form to point at the EAI server as well. For the default instance, the file to modify is usually **/opt/pdweb/www-default/lib/html/C/stepuplogin.html**. Modify it as follows:

```
<SCRIPT LANGUAGE=JavaScript>
window.location.replace(
      “https://<webseal computer>/iealogin/<login form>”);
</SCRIPT>
```

7.4 Sending the authentication level from EAI to WebSEAL

To send the authentication level from an EAI server to WebSEAL, use the **am-eai-auth-level** HTTP header. For example, the following line in the HTTP header sets the authentication level to two:

```
am-eai-auth-level: 2
```


Conclusion

Summary

You should now be able to write EAI servers to use exotic authentication mechanisms, send additional attributes to WebSEAL, and combine step-up authentication with an EAI server.

Resources

The following resources are available for further information:

- To learn more about the Perl programming language, see the tutorials at the following website:

  http://perldoc.perl.org/index-tutorials.html

- To download Perl modules to interface with various systems, for example LDAP, go to the Comprehensive Perl Archive Network website:

  http://www.cpan.org

- Authorization rules are explained in the product documentation:

  http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame.doc_6.1/am61_admin232.htm .

- The IBM developerWorks® website has an article that explains how to write an EAI server in Java:

  http://www.ibm.com/developerworks/tivoli/library/t-tamebeaicar/index.html

- To learn PHP, view the PHP documentation:

  http://www.php.net/manual/en

