How to configure the TAMeb 6.0 EAI to implement complex authentication requirements

Implementation of AGAF

Paul Ashley  
Trevor Norvill

October 16, 2006  
(First published September 21, 2006)

Tivoli® Access Manager for e-business (TAMeb) version 6.0 introduced the external authentication interface (EAI). This TAMeb interface provides another method for providing solutions for complex authentication requirements. This paper discusses how the EAI operates and through an example describes the details of how to configure TAMeb for an EAI application. The source code of the EAI application is available for download.

Introduction

This paper describes how TAMeb version 6.0 can be used to satisfy complex authentication requirements. A new feature of TAMeb 6.0, called the EAI, is designed to provide flexibility in authentication.

In this paper, an example EAI application is built and configured with TAMeb to satisfy the requirements of Australian Government e-Authentication Framework (AGAF). AGAF describes policies for determining authentication assurance levels for citizens that are accessing Federal Government department Web sites.

In addition to implementing AGAF, a solution based on TAMeb provides the following benefits:

1. TAMeb provides a security proxy-based environment, where all requests must be authorized by the TAMeb proxy (WebSEAL) before they can get to an application.
2. The environment can be easily extended to support Federated Identity Management capabilities (SAML, Liberty, WS-Federation, WS-Trust). The Tivoli Federated Identity Management (TFIM) product builds on TAMeb.
3. The TAMeb J2EE™ (WebSphere® and WebLogic) and .NET authorization plug-ins are available to provide role-based security for these environments.
4. The flexible TAMeb authorization policy capabilities can be employed.
5. TAMeb single sign-on capability can be leveraged to a range of applications.
The method described in this paper for building a solution to meet the AGAF requirements is through the use of the EAI capabilities introduced in TAMeb version 6.0. This new capability is used in conjunction with the TAMeb authentication levels, protected object policy, login re-direct and authentication step-up functionality. A detailed description of the TAMeb configuration is described in this paper.

What is AGAF?

A description of AGAF can be found on the Australian Government Information Management Office (AGIMO) Web site. One of the keys to AGAF is the use of an authentication assurance level (AAL) when determining whether access to a resource should be given. The AAL is calculated based on how the user was originally registered and how they authenticate online. This is shown in Figure 1. In this example, resources have AALs of 1 (low) to 4 (high).

Figure 1. AGAF AAL table

<table>
<thead>
<tr>
<th>Strength of registration of entity, identity, or attribute</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Low (2)</td>
<td>Moderate (3)</td>
<td>High (4)</td>
<td>High (4)</td>
<td></td>
</tr>
<tr>
<td>3 Low (2)</td>
<td>Moderate (3)</td>
<td>Moderate (3)</td>
<td>High (4)</td>
<td></td>
</tr>
<tr>
<td>2 Low (2)</td>
<td>Low (2)</td>
<td>Moderate (3)</td>
<td>Moderate (3)</td>
<td></td>
</tr>
<tr>
<td>1 Minimal (1)</td>
<td>Low (2)</td>
<td>Low (2)</td>
<td>Low (2)</td>
<td></td>
</tr>
</tbody>
</table>

Registration

When a person, who may be acting on behalf of a business, wants to access department resources such as Internet Web-based applications, they must first register with the department (or at one of the participating departments). The outcome of registration is that a person has a strength of registration level and also capabilities to later authenticate on-line. This is shown in Figure 2 with an LDAP-capable directory storing these values.

Figure 2. The registration process
Accessing Web-based applications

When the registered users subsequently access a web-based application of the department, the security system will determine if access should be granted. As shown in Figure 3, the steps are as follows:

1. The user attempts to access the Web-based application. The user will be asked to identify themselves, and optionally may be asked to authenticate. For example, all users might be given a user name and password and this step might require the user to authenticate with these credentials.
2. The security system consults with the authorization policy to determine the required AAL of the application.
3. The security system determines from the LDAP the user’s strength of registration and authentication capabilities. For example, a user might have been registered using their Passport for identity verification, and have additional authentication credentials such as token or biometric authentication capabilities.
4. The security system consults the AAL matrix to determine if the user can access the application. This means looking at strength-of-registration and authentication capabilities.
5. If the user has the necessary authentication capabilities, they will be challenged for the authentication credentials. For example, they might be asked to authenticate with a challenge-response token.

Figure 3. Security process implementing AGAF

Authentication alternatives with TAMeb

This section discusses authentication alternatives with TAMeb.

Using WebSEAL "out of the box"

The TAMeb web reverse proxy WebSEAL authenticates and authorizes user’s access to web-based applications. This is shown in Figure 4. In the example, WebSEAL "out of the box"
authentication capabilities are being used. This might be user name and password, certificate, or token authentication. In this case "out of the box" means no customisation of WebSEAL has been implemented.

1. The user attempts to access Application 2.
2. WebSEAL consults its security policy to determine if the user should authenticate. It first checks the access control list (ACL) to see if the user needs to authenticate, followed by an (optional) (Protected Object Policy (POP)). If authentication is required, WebSEAL prompts the user to authenticate.
3. The user provides the authentication information.
4. WebSEAL checks the authentication information. In this case WebSEAL is checking against the LDAP (user name and password).
5. If the user is successfully authenticated, and is also authorized to access the resource, WebSEAL forwards the original request to Application 2.

**Figure 4: Using TAMeb WebSEAL authentication**

**WebSEAL external authentication C API**

WebSEAL provides the ability to customise authentication. The WebSEAL external authentication C API (XAUTHN) capabilities are used when an authentication mechanism is required that is not provided by WebSEAL.

XAUTHN is a custom library written in the C language to interact with the user and implement some type of authentication. The XAUTHN library lives on the WebSEAL server, with WebSEAL handing off authentication to the XAUTHN library. At completion of authentication of the user, the
XAUTHN library identifies the authenticated user to WebSEAL or returns a failed result. It might optionally return extended attributes to the user.

WebSEAL then provides authorization in the usual way as shown in Figure 5.

**Figure 5: WebSEAL XAUTHN authentication**

<table>
<thead>
<tr>
<th>XAUTHN (authentication)</th>
<th>WebSEAL (authorization)</th>
</tr>
</thead>
</table>

**WebSEAL custom EAI**

In TAMeb v6.0, the EAI is introduced. The EAI provides a method for a custom authentication application to be written. As shown in Figure 6, instead of residing on the WebSEAL server, the EAI application is junctioned behind WebSEAL. At the end of authentication, similarly to a XAUTHN, the EAI application notifies WebSEAL of the identity of the authenticated user (and other attributes if required).

**Figure 6: WebSEAL with EAI authentication**

**The TAMeb EAI authentication for AGAF**

The key capabilities of TAMeb for this solution:

- **EAI**: This is the chosen solution for the AGAF requirements. The EAI application will need to take into account the required AAL of the resources being accessed, the user's registration level, what authentication capabilities are available to the user, the AAL table, and possibly other factors. The EAI interacts directly with the user (via a WebSEAL junction) to gather authentication data.

- **Authentication levels**: TAMeb allows authentication levels to be configured from 0 (unauthenticated) to any integer value. These will be mapped to the AGAF AAL by the EAI application.

- **POP**: This is used to indicate what authentication level (and hence AGAF AAL) is required to access a resource. This could be at the application level, or individual URLs.

- **WebSEAL step up authentication**: As the user moves to new applications (or different URLs), WebSEAL will examine the POP and trigger a new authentication if the resource's AAL is higher than the user's current value.
• **Local response redirect**: WebSEAL can be configured to redirect new login requests to a URL (in this case the EAI server). WebSEAL will indicate to the EAI application what AAL is required for this resource.

**Figure 7: TAMeb EAI authentication for AGAF**

---

**Figure 7** shows an example flow:

1. The user sends a request to WebSEAL to access Application 2 (junctioned behind WebSEAL).
2. WebSEAL consults its authorization policy and determines that the user requires AAL 4. As the user has only been accessing public resources up until that point and is unauthenticated (AAL 0) step up authentication is kicked off. The user's request is redirected to the EAI application with an indication to the EAI application that AAL 4 is required. WebSEAL caches the original URL request.
3. The EAI application consults the AAL matrix and the user's LDAP entries to determine how the user can authenticate to meet the AAL 4 requirement. (In the application, the user is always forced to authenticate with the user name and password to confirm the identity of the user before their LDAP information is queried).
4. The EAI application sends a response to the user indicating how the user can authenticate to access the application. There might only be one choice, or none (the user cannot reach the AAL based on their registration level and authentication capabilities) or there might be multiple choices.
5. The user enters their authentication data. This is returned to the EAI application that authenticates the user with the data. The EAI signals to WebSEAL that the authentication has passed or failed. If passed, the EAI application will return the user’s AAL to WebSEAL. The user’s AAL will be the same or higher than that required for the resource.  
6. WebSEAL will redirect the user's browser to the original resource.

The user can later try to access a higher AAL resource, and the process will be repeated.

**The sample AGAF application**

We have created an AGAF application that implements the AGAF requirements. An Enterprise Archive Repository file (ear) contains the source code and is immediately deployable to your Java™ application server. (The code was created in Rational® Application Developer for running on WebSphere® Application Server 6.0.0.2).

The AGAF application is made up of two components:

1. EAI authentication application (http://hostname/eai/AGAFDemoWeb/AGAFAppMainPage.html). This Web page has links to the applications at different AAL  
2. Administration application (http://hostname/eai/AGAFDemoWeb/AGAFIndex.jsp). This URL is accessed by the department administrator.

**EAI authentication application**

The EAI authentication application servlet that interacts with the requests coming from the user is shown. This is the highest level servlet code, and does not contain all of the Java classes that are called from it. The full source code is available in the ear file.

The EAI authentication application will prompt the user first to authenticate with a user name and password whenever a protected resource is accessed. This is so that the user is positively identified before accessing their registration level and authentication capabilities.

When a user accesses a resource at a higher AAL, the EAI application calculates which authentication capability the user has available (there might be more than one) and offers the user the choice to authenticate. This is shown in Figure 8.
Figure 8: Authentication selection

Authentication Selection Page

You are trying to access an application that requires a stronger form of authentication. Please select from the following list:

- biometric

Submit

The EAI authentication application itself does not perform authentication. This code would be inserted for each authentication capability.

Administration application

The main administration screen has the following choices:

1. LDAP configuration: For configuring the EAI application to work with the TAMeb user registry.
2. TAM configuration: For configuring the EAI application as a TAMeb application (so it can create Tivoli Access Manager users).
3. User registration: For registering a new user with a registration level and authentication capabilities.
4. Changing authentication capabilities: For defining the different authentication capabilities of the department and assigning them to an authentication level.
5. Change registration requirements level: For defining the different registration options.
6. Change authentication assurance levels: For defining the matrix of AAL for each registration level and authentication level.

The registration screen as shown in Figure 9 is used for creating a user, defining their registration level and giving the user different authentication capabilities:
How to configure the TAMeb 6.0 EAI to implement complex authentication requirements

**Figure 9: User registration screen**

![User registration screen](image)

The different authentication capabilities that are provided by the EAI application can also be defined as shown in Figure 10:
Figure 10: Defining authentication capabilities

AGAF Authentication Capabilities

- **Authentication Level 1**: User name/Password
- **Authentication Level 2**: Token
- **Authentication Level 3**: Smart card
- **Authentication Level 4**: Biometric

The different registration levels can be defined as shown in Figure 11:
The AAL table is configurable as shown in Figure 12, so that based on a registration and authentication capability, an AAL can be also be defined. For example, from the table, at registration level 2, authentication level 4, the AAL has been defined as 5.
Change Authentication Assurance Level (AAL)

Configuration of the AGAF application

This section describes the configuration of the AGAF Application.

AGAF demo configuration

To configure the AGAF demonstration application, you need the following environment:

1. Websphere Application Server 6.0.0.2
2. Tivoli Directory Server 6.0
3. Tivoli Access Manager 6.0 Runtime
4. Tivoli Access Manager 6.0 Policy Server
5. Tivoli Access Manager 6.0 Authorization Server
6. Tivoli Access Manager 6.0 Runtime for Java
7. Tivoli Access Manager 6.0 WebSEAL

Create protected applications

Six back-end applications are protected by WebSEAL. Create six applications in the Webseal/opt/pdweb/www-default/docs directory. Call them app1.html to app6.html. The content of the html files can be trivial but should identify the protected application.
On the computer that you are using to access the application, change the /etc/hosts file to add an entry for www.demo.com to point to the Webseal Server network interface. This URL is hard-coded in the application main page.

**WebSEAL configuration**
This section details how to configure the EAI application with WebSEAL.

Create a junction from the WebSEAL servers to the EAI application:

```
# pdadmin -a sec_master -p password
pdadmin> server task default-webseald-hostname create -t tcp-h eai-hostname -c iv_user /eai
```

Allow unauthenticated access to the junction. First create the unauthenticated ACL:

```
padmin> acl create unauth
padmin> acl modify unauth set group iv-admin TcmdbsvaBRrxl
padmin> acl modify unauth set group webseal-servers Tgmdbsrxl
padmin> acl modify unauth set user sec_master TcmdbsvaBRrxl
padmin> acl modify unauth set any-other Trx
padmin> acl modify unauth set unauthenticated Trx
```

Now attach the ACL to the EAI junction:

```
padmin> acl attach /WebSEAL/hostname-default/eai unauth
```

Configure EAI authentication library in WebSEAL. In this step and the following, the configuration file `webseald-default.conf` is being edited:

```
[authentication-mechanisms]
# EXTERNAL AUTHENTICATION INTERFACE
# ext-auth-interface = <ext-auth-interface-library>
ext-auth-interface = /opt/pdwebrte/lib/libeaiauthn.so
```

Enable the EAI for HTTP and HTTPS sessions:

```
[eai]
#-----------------------
# EXTERNAL AUTHENTICATION INTERFACE
#-----------------------
# Enable EAI authentication. No other EAI parameters will take effect
# if this is set to 'none'.
# One of <http, https, both, none>
eai-auth = both
```

Set up the trigger URLs. A trigger URL is the URL of a back-end resource that generates an EAI message. WebSEAL only accepts EAI messages from resources that are configured as trigger URLs. In the `webseal-default.conf` file:

```
[eai-trigger-urls]
trigger = /eai/AGAFDemoWeb/AGAFDemoEAI
```
Configuring Local Response Redirect in WebSEAL. This feature changes WebSEAL so that all locally generated pages (including the log-in page) are replaced with a re-direct to a specified URL. If this URL is part of the EAI Application, it can check whether the request is for the log-in page and, if so, initiate EAI Authentication.

First enable local response re-direct:

```
[acnt-mgt]
enable-local-response-redirect = yes
```

Next, provide the URL of the application that will handle the requests for local responses.

```
[local-response-redirect]
local-response-redirect-uri = /eai/AGAFDemoWeb/AGAFDemoEAI
```

Finally, configure the information that will be passed to the local response handler. This information is passed in the query string of the redirect so it can be easily read by the back-end application.

```
[local-response-macros]
macro = AUTHNLEVEL
macro = ERROR_CODE
```

Configure the authentication levels for step-up authentication. All levels of authentication will map to the EAI application:

```
[authentication-levels]
level = unauthenticated
level = ext-auth-interface
level = ext-auth-interface
level = ext-auth-interface
level = ext-auth-interface
level = ext-auth-interface
level = ext-auth-interface
level = ext-auth-interface
level = ext-auth-interface
level = ext-auth-interface
```

Restart webseal:

```
# pdweb restart
```

Configure POPs for each of the applications. In this example, there are six applications that you can attach POPs to. For each application level, x, do the following:

```
# pdadmin -a sec_master -p password
pdadmin> pop create authlevelx
pdadmin> pop modify authlevelx set ipauth anyothernw x
pdadmin> pop attach /WebSEAL/hostname-default/appx.html authlevelx
```
Tivoli Access Manager Java Runtime configuration

The AGAF demonstration application creates Tivoli Access Manager (TAM) users through the TAM Java Administration API. To do this, the TAM Java runtime must be configured. Configure the TAM Java runtime against the Websphere Java Runtime Environment.

Configure the AGAF application into the secure domain

Configure the AGAF application into the TAM secure domain by using the java svrSslCfg command:

```
java com.tivoli.pd.jcfg.SvrSslCfg -action config
   -admin_id sec_master -admin_pwd <sec_master password>
   -appsvr_id <application Server Name> -appsvr_pwd <application server password>
   -host <host> -mode remote -port <port> -policysvr <policy server name>:7135:1
   -authzsvr <authorization server name>:7136:1
   -cfg_file <configuration file> -key_file <key file>
   -domain mydomain
   -cfg_action create
```

Deploying the application

To deploy the AGAF EAI application to WebSphere Application Server 6.0.0.2 use the Websphere administration console and do the following:

1. Expand the Applications tab.
2. Click on Install new applications.
3. Specify the path to the AGAF demo EAI application on the local file system.
4. Accept the default options for the rest of the deployment.
5. Save the configuration.

The application is not started automatically after deployment. Start the application through the Enterprise Application link.

Using the AGAF administration application

After the AGAF application is installed, it needs to be configured. The configuration index page is located at:

http://hostname/eai/AGAFDemoWeb/AGAFIndex.jsp

From this page, configure the following:

1. LDAP configuration. Enter the following LDAP information:
   1. The LDAP provider name in the form ldap://<ldap hostname>:<port>
   2. The LDAP Suffix: eg c=us
   3. The bind DN
   4. The bind password

   2. TAM configuration
1. The TAM configuration file configured during the SvrSslCfg. This will be in the format file://<file>
2. The TAM administration User (sec_master)
3. The TAM administrator password
4. d. The registry suffix.

3. User registration: Register users that will authenticate to the EAI. Provide the following information for each user configured:
   1. User name
   2. First name
   3. Last name
   4. Registration type
   5. Specify an authentication mechanism for each required authentication level

4. Change authentication capabilities: Change the authentication capabilities for each level as required

5. Change registration requirements Level: For each registration level, select a registration type

6. Change authentication assurance Levels: Define authentication assurance levels for the authentication assurance level matrix.

Accessing the AGAF applications
The configuration of the AGAF demonstration application is now complete. You can now access protected resources. To do this, visit the following link:

http://hostname/eai/AGAFDemoWeb/AGAFAppMainPage.html

This is an index page that has links to six protected resources, app1.html through app6.html. You will be able to access each one depending on the registration strength you gave the user, the authentication capabilities you gave the user and the configuration of the authentication assurance level(AAL) Matrix. If your AAL is not high enough and you are able to step up to a higher AAL by using a different authentication mechanism, you will be prompted to choose a stronger authentication mechanism. Upon successful authentication with the required authentication mechanism, you will be allowed to access the protected resources.

Conclusions
The addition of the EAI to TAMeb provides another method for implementing arbitrarily complex authentication requirements. By following the steps outlined in this paper, and using the example EAI application, it is possible to quickly set up a simple EAI application for use with TAMeb.

Sample code listing
Listing 1. EAI Application Servlet for AGAF

AGAFDemo.java

```java
package com.tivoli.am.eb.AGAFDemo;

import java.io.IOException;
import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```
import javax.servlet.http.HttpServletRequest;
import java.net.URLEncoder;
import java.util.Enumeration;
import java.io.IOException;
/*
* AGAF Demonstration EAI Servlet
*/
public class AGAFDemoEAI extends HttpServlet implements Servlet {

    /*
    * Configuration object to store EAI state
    */
    private AGAFServerConfig config = null;

    /*
    * Form fields
    */
    public static final String AUTH_OP = "authType";
    public static final String AUTHN_LEVEL_FIELD = "AUTHNLEVEL";
    private String authOperation = null;

    public AGAFDemoEAI() {
        super();
    }

    /*
    * The first time the EAI is accessed via the Webseal Local Response
    * Redirect capability, the EAI redirects the user to the
    * username/password login page. By default, all users are allowed
    * to use username/password authentication. The EAI needs to establish
    * the user's identity via username/password authentication in
    * order to retrieve the authentication methods and their registration
    * type.
    */
    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException,
        IOException {
        // TODO Auto-generated method stub
        String url = "/AGAFUserPassLogin.jsp";
        Enumeration parameters = arg0.getParameterNames();
        String name = null;
        String value = null;
        url = url.trim() + "?";
        while (parameters.hasMoreElements()) {
            name = (String)parameters.nextElement();
            value = arg0.getParameter(name);
            url = url + name + "=" + URLEncoder.encode(value) + ";";
        }

        arg1.sendRedirect(arg1.encodeRedirectURL(url.substring(0,
        url.length())));
    }

    /*
    * Handle the authentication request depending on the contents of the
    * posted login form. Each authentication object completes the
    * authentication and determines if the user has the AAL required. If
    * the AAL is not high enough, the authentication object redirects to
    * the authentication selection page to determine what authentication
    * methods will allow the user to achieve the required AAL.
    */
    protected void doPost(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException,
        IOException {
        // TODO Auto-generated method stub
    }
}
authOperation = arg0.getParameter(AUTH_OP);
AGAFAuth auth = null;
try {
    if (authOperation.equals(config.getUserPassString())) {
        auth = new AGAFUserPassAuth(config, arg0, arg1);
    }
    if (authOperation.equals(config.getTokenString())) {
        auth = new AGAFToken(config, arg0, arg1);
    }
    if (authOperation.equals(config.getSmartCardString())) {
        auth = new AGAFSmartCardAuth(config, arg0, arg1);
    }
    if (authOperation.equals(config.getBiometricString())) {
        auth = new AGAFBiometricAuth(config, arg0, arg1);
    }
    if (authOperation.equals(config.getAuthSelectionString())) {
        auth = new AGAFAuthSelection(config, arg0, arg1);
    }
    if (auth == null) {
        auth = new AGAFUnauth(config, arg0, arg1);
    }
} catch (IllegalStateException e) {
    super.log("A fatal exception has occured: ", e);
    throw new ServletException(e);
}
if (auth != null) {
    try {
        auth.authenticate();
    } catch (IOException e) {
        super.log("A fatal exception has occured: ", e);
        throw new ServletException(e);
    }
}

/**
 * Initialize the Servlet*
 */
public void init() throws ServletException {
    // TODO Auto-generated method stub
    super.init();
    config = AGAFServerConfig.getInstance(this.getServletContext());
}
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example AGAF EAI Application</td>
<td>AGAFDemo.zip</td>
<td>76KB</td>
</tr>
</tbody>
</table>
Related topics

- Secure Web Based Applications with Tivoli Access Manager for e-business.
- Federate identities with Tivoli Federated Identity Manager.
- Get details on how to implement AGAF at the Australian Government e-Authentication Framework (AGAF) for Business Web site

© Copyright IBM Corporation 2006
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)