How to Make Your Oracle ApEx Application Secure

Peter Lorenzen, WM-data a LogicaCMG company

Security - What security?

The lesson learned in the Internet era is that nothing is secure. There are security flaws in all systems. Therefore security is something that has to be taken seriously. The subject is enormous so this article will be more of an overview of the threats and countermeasures than a thorough analysis. It will be in terms of what I believe an Oracle Application Express (APEX) developer in a small shop, without a fulltime security expert or DBA, should know. We will look at architecture, hardening of the components in the architecture and then proceed to specific threats.

The amount of time and money that is used on security should always be measured against the value of the data that is secured. The basis for this presentation is that we have an application that contains valuable and secret information, and that the application is accessed from the Internet.

"Security is an architecture, not an appliance" - Art Wittman

APEX consists of a database and a HTTP Web server. The HTTP server can be one of these flavors:

- An Oracle XML DB HTTP Server
- An Oracle HTTP Server from the database companion CD
- An Oracle HTTP Server from the Oracle Application Server 10g

The Oracle XML DB HTTP Server

This HTTP server is running in the database. It was introduced with Oracle XML DB in Oracle 9i R2 and uses the Embedded PL/SQL Gateway (EPG) to talk to the database. The special version of APEX used in the free Oracle database - Oracle

Express Edition, uses this HTTP server. Currently it is not supported for any other database versions, but it will be supported for release 11 of the database.

The Oracle HTTP Server (OHS) from the database companion CD

This OHS is based on an Apache 1.3.x HTTP server and uses the mod_plsql module to communicate with the database. Apache 1.3 is an old but stable version that will not be enhanced anymore, except with security patches.

The Oracle HTTP Server (OHS) from the Oracle Application Server 10g

This OHS is based on an Apache 1.3.x HTTP server and as far as I know it is identical to the OHS from the database companion CD.

Which HTTP Server to Use?

When version 11 of the database is released we will have 2 HTTP servers to choose from. From a security perspective, I would choose an OHS. The reason being it is a popular Web server of proven quality, while the XML DB HTTP server is an unproven creature.

On the Oracle Application Server companion CD is an OHS based on an Apache 2.0.x HTTP server. This release does not contain the mod_plsql module and can therefore not be used for APEX.

Choosing an Architecture

You should choose an architecture where as little as possible is exposed to the Internet. The minimum setup I would suggest is shown in figure 1.

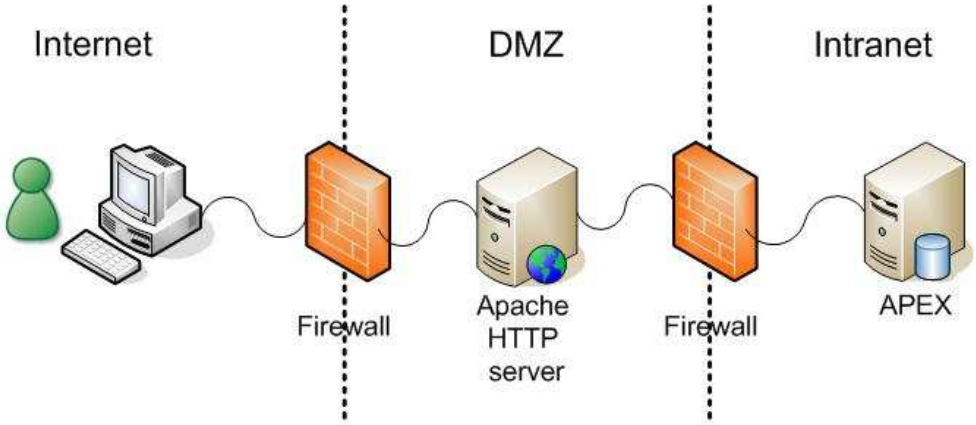

Figure 1. Minimum architecture

APEX is tucked away on the Intranet. The Apache HTTP server in the DMZ can be an OHS, but you have to pay a license fee to Oracle if you use it. You are not allowed to install an OHS on any server other than one that contains the Oracle Database or the Oracle Application Server, unless you pay for an extra license. A free alternative is to use a standard Apache 1.3/2.0 HTTP server directly from Apache. Do note that administering a standard Apache differs from an OHS, so if you want a familiar interface you need an OHS.

For added security many choose to encrypt the communication between the browser and the HTTP Web server. If encryption is not used an eavesdropper can intercept all information sent between the browser and the server, even passwords. The most popular way to encrypt is via Secure Sockets Layer (SSL). The encryption is transparent to APEX and therefore no configuration is needed in APEX. This architecture is shown in figure 2.

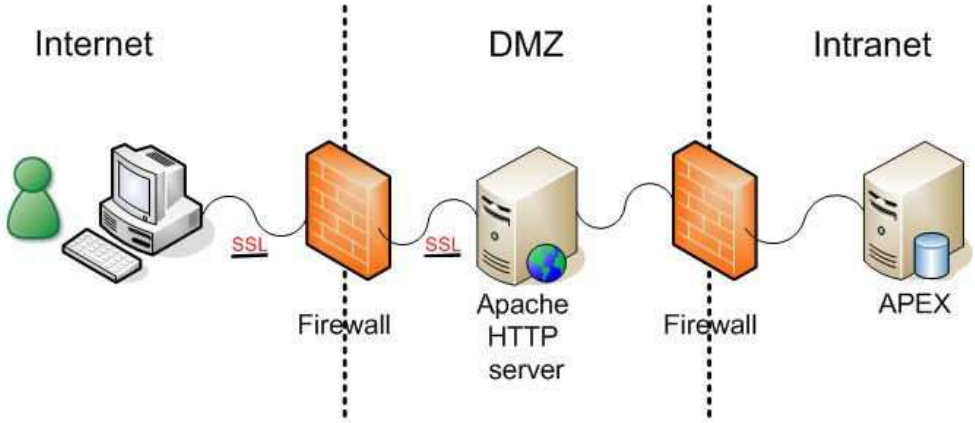

Figure 2. Using SSL encryption

For added security it is also possible to use SSL encryption between the Apache HTTP server in the DMZ and the HTTP server installed with APEX.

Since SSL is a de facto standard for securing Internet communication I will later explain a bit above how to set it up.

Hardening your Architecture

It is important to harden the different components in your architecture. This means patching all components and removing unnecessary functionality, users, privileges, etc.

Patch, Patch, Patch

One of the most important security measures is patching. Always keep your software updated with the latest patch. This includes the Operating system. Oracle releases a quarterly Critical Patch Update (CPU) that contains security patches for all implicated Oracle products. You can subscribe to Oracle Security Alerts at the Oracle Technology Network (OTN) site. Also remember to install the normal patch sets as they are released. You can get patch sets from Oracle Metalink, but only if you have an active support agreement. I know of no way to get notified when normal patch sets become available, so visit Metalink on a regular basis.

The Oracle XML DB HTTP is patched via the database patches. The OHS servers should be patched with patches from Oracle not from Apache. Apache servers installed directly from Apache should be patched via http://httpd.apache.org. Apache does not as such release patches, but releases a whole new version when it is needed.

Hardening the Database

Do *not* use the free Express Edition (XE) database since it can not be patched. Follow the principle of least privilege, so a user only has access to the resources required. Lock or remove unused users. Use sensible passwords, and do not use the same password for SYS and SYSTEM. There are lots of more advanced things you can do so check these:

- Oracle has a thorough up-to-date Database Security Checklist. See http://tinyurl.com/ytake2
- Oracle has created Project Lockdown which is a project that via 4 phases helps you to secure your database infrastructure. See http://tinyurl.com/24s4nf
- Hacking and Securing Oracle - A Guide To Oracle Security by Pete Finnigan. This is scary stuff so don't read it late at night. http://tinyurl.com/28jrt7

Hardening the Apache HTTP Web Server

Apache comes with a lot of pre-loaded modules that you probably won't need. For example mod_perl and mod_cgi. Remove these by deleting the corresponding LoadModule directives from the httpd.conf file. There are also pre-installed content and examples that should be removed, things like FastCGI examples. It is an advantage to give away as little information as possible about the software you run. If you call a page that does not exist, Apache will return a HTTP 404 error with information like this:

```
Oracle-Application-Server-10g/10.1.2.0.0 Oracle-HTTP-Server Server at apex.corp.net Port 7777
```

As default this is shown in the http header: **`Server: Oracle-Application-Server-10g/10.1.2.0.0 Oracle-HTTP-Server`**. To stop the publishing of this information these two directives in the httpd.conf file can be changed:

```
ServerSignature Off
ServerTokens Prod
```

For a comprehensive checklist of things to harden read these:

- Securing Oracle Application Server by Caleb Sima. See http://tinyurl.com/2ey89a
- Hardening Oracle Application Server 9i and 10g by Alexander Kornbrust. See http://tinyurl.com/2x5h3h

They both cover the Oracle Application Sever so they contain information that is not relevant if you use the OHS from the database Companion CD but they are still very informative.

Specific Threats

In this section, I will take a closer look at some of the major threats and what can be done to counter them.

Cross-Site Scripting (XSS)

XSS is a misleading label, but for historic reasons this is what it is called. It covers different types of attacks. My simple definition is that it covers situations where an attacker injects JavaScript in an application in order to steal data or corrupt the application. Here is a simple example in APEX:

1. Create or pick an existing table that contains a Varchar2 column
2. Create a Form on the table of type "Form on a Table with Report"
3. Run the Report and create a row with this data: **`Test<script>alert('Hello world');</script>`**
4. When you press Create and branch back to the Report the JavaScript is executed and you should see an alert message like the one in Figure 3.

Figure 3. XSS example

Following the example a user can write a malicious JavaScript that is executed when another user displays the Report page. The attacker can steal all the data in the browser including cookies and credentials. To prevent this, change Display as from "Standard Report Column" to "Display as text (escape special characters, does not save state)" in the Column Definition under the Report Attributes. Now the JavaScript is not executed. The reason is that special characters like <,>,&," are now escaped. If you look on the page source it looks like this:

```
Test&lt;script&gt;alert('Hello world');&lt;/script&gt;
```

Escaping is the weapon of choice when dealing with XSS threats. If you write your modules in PL/SQL you can use the htf.escape_sc function to escape special characters.

SQL Injection

An SQL Injection attack is a form of attack where the attacker inputs extra SQL in an application. Here is a simple but illustrative example. If you have an APEX Report based on a SQL Query, like this Select:

```
select ename, job, sal from emp where ename = '&P1_ENAME.'
```

P1_ENAME is a text item that the user can input. If the user inputs a valid name like KING one row is displayed, but if the user instead input this:

```
QWERTY' or 1=1--
```

All rows will be displayed, which might not be the intention. You can change the Select to use bind variables like this:

```
return 'select ename, job, sal from emp where ename = :P1_ENAME';
```

Now there won't be any problem. The reason is that the Oracle database uses the value of the bind variable exclusively and does not interpret its contents in any way. You should of cause always use bind variables since it is well known that it improves performance by allowing statement reuse and thereby prevents hard parses.

Generally SQL Injection should not be too much of a problem in APEX, but take care when using end-user input in your DML. In PL/SQL you can write dynamic SQL via the old DBMS_SQL package or the modern Native Dynamic SQL (NDS) e.g. Execute Immediate. If you use these you should be aware and always validate the input. Check for length, parentheses, comments (--, /* */) etc. or do a sanity check by validating the input against a table. If the user inputs a department name check that the department name exists.

Hardening APEX

APEX has a string of built in functions that can help make you application more secure.

Session State Protection (SSP)

In an APEX application it is by default possible to change parameters in the URL hereby gaining access to information you should not be able to access. This URL tampering can be prevented by using SSP. When SSP is enabled and configured for a page, a checksum is calculated from the URL parameters and set as a new parameter. When a page is called the checksum is recalculated and checked against the parameter value. If the checksum does not match, the page is not displayed and you receive an error message.

SSL is a very good security measure and should be used in most applications. When a checksum is calculated a random salt is used. You can change the salt via the clicking Expire Bookmarks on the Edit Security Attributes page.

Even though SSP helps to prevent URL tampering, there really should be other security measures on the pages or even better in the database to prevent unauthorized access. On the page you can prevent access to the whole page or objects on the page using authorization schemes. In the database you can for example use triggers, or Virtual Private Database (VPD) to prevent access.

Security Options in the Administration Services

In the APEX Administration Services, an administrator has a list of security options that should be configured on a production server before going live.

- Disable Administrator Login: This prevents any user from logging in to the Administration Services. This setting can be reversed by calling a couple of PL/SQL procedures. (See more in the User's Guide)
- Disable Workspace Login: Prevents any user from logging in to any workspace. When you need to install a new version of your application you have to undo the Disable Administrator Login as explained above and then undo Disable Workspace Login.
- Restrict Access by IP Address: Instead of using the 2 options above you can input a list of IP addresses from which it is OK to login as an administrator or into a workspace.
- Workspace Password Policy: It is possible to set a range of options for the password policy for Application Express users (workspace administrators, developers, and end users) in all workspaces. Things like checks for min./max. length and number of alphabetic and numeric characters. It is also possible to define a list of words that a password may not contain.

There are other options so be sure to check them out.

Miscellaneous

For the application running in production debugging should be disabled and Build Status should be Run Application Only. This can either be set manually in the Application definition or when you export the application for import on the production server. When you export also remember to set Export Comments to No. There is no reason to expose that kind of information.

When you use HTML controls like checkboxes, radio buttons or select lists you might think that you don't need to validate the input from these since the values are constrained. You cannot however count on this. A select list can easily be converted to a normal text input field. For example if you use the Mozilla Firefox browser with the Web Developer Extension installed, you have a menu option that can convert all select lists on the current page to text fields. So take care and validate all input. Use a database trigger or if possible a foreign key constraint or a check constraint.

Obfuscate the APEX_PUBLIC_USER Password

This function is not really a part of APEX but still interesting. When a user is logged into an APEX application all database operations are performed by a database session connected as APEX_PUBLIC_USER as defined in the DAD setup file (dads.conf). This is true for all authentication schemes. The APEX_PUBLIC_USER user only have the CREATE SESSION privilege. As default the password for APEX_PUBLIC_USER is stored in clear text in the PlsqlDatabasePassword parameter in the DAD. For added security the password can be obfuscated via the dadTool.pl utility. See:

```
ORACLE_HOME\Apache\modplsql\conf\dadTool.README
```

Here is an example of using the dadTool on Windows:

```
set ORACLE_HOME=D:\OracleDb10g
set PATH=%ORACLE_HOME%\Apache\modplsql\conf;%PATH%
set PATH=%ORACLE_HOME%\perl\5.6.1\bin\MSWin32-x86;%PATH%
set PATH=%ORACLE_HOME%\bin;%PATH%

cd %ORACLE_HOME%\Apache\modplsql\conf

perl dadTool.pl -o
```

If you use marvel.conf you need to temporarily rename the file to dads.conf.

Implementing SSL in a Oracle HTTP Server

There are several cryptographic techniques involved in SSL: Public-Private Key, Symmetric Key, and Digital Signature. For simplicity I will ignore most of this. The first time a browser sends a request for a page that is secured by SSL. The HTTP server responds by sending a certificate back to the browser. The browser checks if this certificate has been signed by a trusted authority listed in the browser. If it has an SSL connection is started. The trusted authorities are called a Certificate Authority (CA). An example of a CA is Verisign. You can use your own self signed certificates but it complicates matters.

Oracle uses a Wallet to store the keys, certificates etc. that is used for SSL on the OHS. A Wallet is just a directory on the server where the information is store encrypted. Oracle Wallet Manager is used to manage the Wallets. The first thing you need to do is to generate a Certificate Signing Request (CSR) in the Wallet Manager and send it to the CA. The CSR contains, among other things, information about the person/organization that is requesting the certificate signing. The CA receives the CSR, signs it and returns it. Then the CSR has to be imported into to the Wallet via the Wallet Manager. Now you are ready to use SSL. I have created a more thorough How-to on the APEX Wiki, see http://tinyurl.com/2zosrp.

Conclusion

This paper outlines a couple of ways to design your security architecture and lists a string of issues that I hope can inspire you to create your own checklist for hardening your architecture. Most of the things I have listed are some that can easily be implemented without big costs. Basic security hardening does not need to be expensive it is just at matter of knowing where the weaknesses are. Please read the database and Oracle Application Server documents that is mentioned. Especially Pete Finnigan's paper is an eye-opener and a must-read.

If you deploy on the Internet security cannot be ignored, but do not forget the threats inside the firewalls. Resent studies show that the vast majority of computer hacking is done by current and former employees.