

# HARDENING APEX FOR MAXIMUM SECURITY

*Randy Cunningham, SageLogix, Inc.*

## **INTRODUCTION**

### **APPLICATION EXPRESS: BACKGROUND**

Oracle Application Express, also known by its short name of Apex, is a web-based software development and run-time environment for Oracle databases. Since its introduction in 2000, it has also been known by the names Flows, Oracle Platform, Project Marvel and HTML DB.

Apex is easy to learn, deploys quickly and requires essentially no endpoint support because it runs in a web browser. As a result, the software has gained a popular following for application prototyping and for rapid application development.

However, there is no reason to limit Apex only to speedy deployments; it is sufficiently rich and robust to allow its use for full-scale production applications. For example, the Ask Tom website (<http://asktom.oracle.com>) is built on Application Express as was Oracle's MetaLink support site, until recently.

Beginning with Oracle Database 11g, Apex is included in the base product deliverable, so it is readily available to a wide audience.

### **SECURITY IN APEX**

Out of the box, Apex includes a rudimentary user-and-role based access control framework, including administrative access for performing such super-user tasks as creating users, assigning roles and setting overall security policies. In addition, there is developer level access and end-user level access, each of which is adjustable based on installation needs and policies.

The immediately available authentication schemes include the Apex user framework, database authentication through the DAD, or authentication using a database account. Regrettably, authentication can be disabled entirely. In addition to the supplied authentication schemes, Apex shared components and PL/SQL libraries permit Apex session authentication using LDAP, including OID and Windows Active Directory, so enterprise roles, responsibilities and groups are readily accessible to the Apex application. This approach inevitably requires the development of site-specific, custom code.

By default, Apex logs all session authentication attempts into a database table.

### **ORGANIZATIONAL SECURITY REQUIREMENTS**

Apex is not inherently insecure. Similarly to the Oracle database itself, Apex can be compromised to a point where it offers essentially no security at all, or it can be augmented and configured to a degree that enables Apex applications to comply adequately with stringent organizational requirements and statutory requirements including HIPAA and Sarbanes-Oxley.

Also, Apex is just one component in an overall security landscape. Attempting to implement a security policy or to comply with security requirements only by adjusting Apex is doomed to fail; the database, the operating system and the network must also support the overall organizational security plan.

### **UNDERSTANDING THE THREATS**

It is a natural tendency when approaching topics concerning information technology security to focus on the model of an adolescent hacker in some faraway land, an individual with a malicious streak intent on stealing information, blackmailing their target, or vandalizing a web site. While this threat is undoubtedly real, security also encompasses these scenarios:

- Unauthorized data incursions by the organization's own employees, customers, vendors and other authorized parties;
- Massive corruption or loss of data through operator or procedural error;
- Accidental disclosure of data to unauthorized parties or to the public;
- Social engineering, where data are destroyed, changed or disclosed by authorized individuals who have been duped.

## **START WITH SECURITY IN MIND**

All too often, security is an afterthought; it is something that is done after all of the user acceptance testing is complete, and often with an attitude of appeasing the organization's internal audit group! In other cases, security is only considered after there has been a security breach, either within your own organization, or within another organization who receives adverse publicity for their apparent indifference to information security.

Some of the potential requirements that should be addressed as the application is architected include these:

What is the value, sensitivity and liability associated with the information managed by the application?

What consequences would ensue if the information were leaked or disclosed to unauthorized parties or to the public?

Who normally sees, and who is prevented from seeing, the information?

What legal requirements affect the disclosure and alteration of the information in the application?

What organizational security standards and policies are already in place for this type of application or information?

- a) Do data need to be encrypted in transit?
- b) Do data need to be encrypted at rest?
- c) Are password strength and expiration policies applicable?

What is the scope of access, i.e., is the information on the internet, available company-wide, or accessed by a select group?

What is the intended population who will be accessing the information? (The larger the population, the greater are the security challenges presented.)

Once requirements have been ascertained, more detailed and more technically focused issues should be addressed.

In many cases, the network infrastructure will require the most scrutiny; it is the likeliest point of entry for unauthorized access to the application. Issues including firewall configuration, SSL encryption, VPN access and other factors will require decisions.

The security architecture of the underlying database will also require consideration... Are business rules enforced at the app server, or at the database? Do applications other than the Apex application manipulate or access the underlying data? Who has access to the tables (and other objects) directly from the database, and not just from the Apex application? Does the database comply with security best practices, including password policies, access to data controlled through roles, adequate audit trails and appropriate backup/restore policies?

Also, the architecture of the operating system must be reviewed. What users and groups have access to various infrastructure components supporting the database and the application server? Is access granted according to the principle of least privilege? Is the server hardware in a secure location, with adequate environmental, physical access, recovery and failover provisions?

Finally, security architecture is not complete with consideration given to ongoing security maintenance... this includes provisions for patching various components with vendor-supplied security patches, reviews of audit logs and periodic review of access grants at all levels within the overall architecture.

## **APEX ARCHITECTURE**

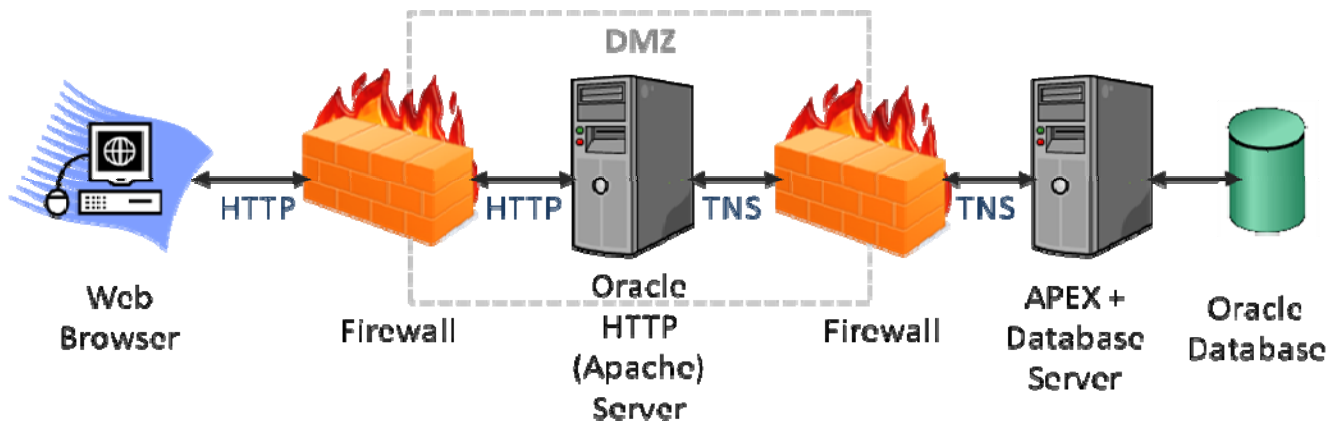
The Oracle Application Express Architecture requires a web server to proxy requests between a web browser and the Oracle Application Express Engine, which resides in an Oracle database. There are three distinct architectures that can be deployed for Apex:

- Using Oracle HTTP application server with mod\_plsql
- Using XDB HTTP protocol server with the embedded PL/SQL gateway (EPG)
- Using the Oracle Application Express Listener (new), with
  - Oracle WebLogic
  - Apache Tomcat
  - OC4J

Each of these has various strengths and weaknesses from the standpoint of performance, security and ease of administration. They are illustrated and compared below:

### CLASSIC HTTP SERVER CONFIGURATION

The original architecture for Apex, and still a very serviceable model, is the one depicted below:



In this architecture, a browser session communicates to an Oracle HTTP server, possibly through a firewall and also possibly over the HTTPS protocol, to an application server over a predefined port number. The application server, using a plug-in called `mod_plsql`, dispatches PL/SQL API calls to an Oracle database hosting Application Express. The PL/SQL procedures within Apex then negotiate data retrieval and modification on the database itself and return content (HTML) to the application server which in turn serves it to the web browser.

#### Advantages:

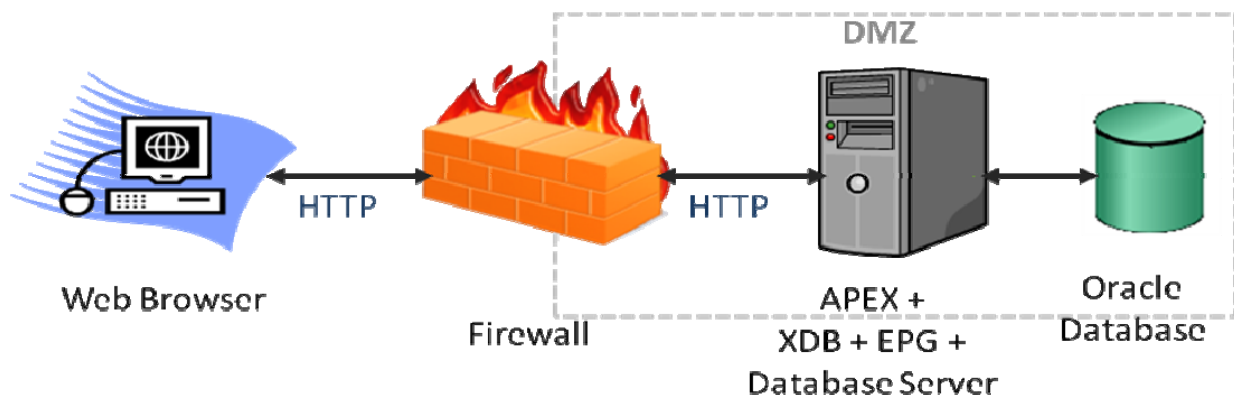
1. Oldest and most proven architecture for Apex.
2. Utilizes Apache, which is well-understood by non-Oracle practitioners.
3. Readily amenable to load-balancing and fail-over technologies.
4. Provides “separation of duties” among components, improving inherent security.

#### Disadvantages:

1. Involves a large number of moving parts.
2. Has potential vulnerability to performance issues.

### EMBEDDED PL/SQL GATEWAY

More recently, Oracle has made available the embedded PL/SQL gateway (EPG), which uses the XDB HTTP protocol server to broker network communications with Apex and the database. This is illustrated below:



This architectural model, which requires no coding changes and imposes no user-visible changes to the application, directs HTTP traffic directly to the database server. There, within the Oracle instance, an HTTP server handles the communication and seamlessly passes it to Apex.

*Advantages:*

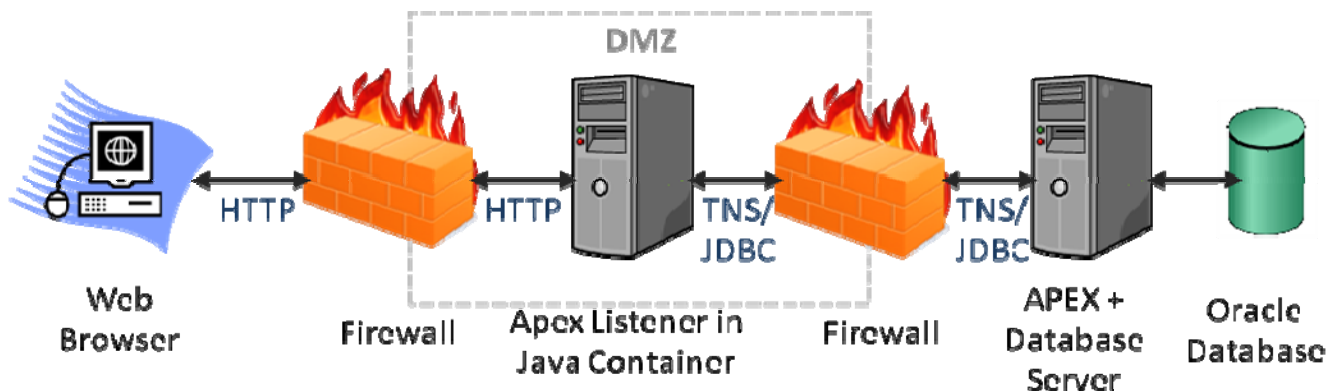
1. Minimizes the number of hardware and software components required to implement Apex.
2. Is easy to install and configure.
3. Minimizes network exposure, potentially improving security.
4. Performs well, due to minimal software and network layering.

*Disadvantages:*

1. Integration of networking and database functions requires enabling endpoint access to the database server, potentially compromising security
2. Functional integration could require that the database server be in the DMZ, compromising security.

### ORACLE APPLICATION EXPRESS LISTENER

Oracle's most recent proxy solution for connecting the web browser to Apex is the Oracle Application Express Listener. A choice of containers – Apache Tomcat, Oracle WebLogic and OC4J – is usable with the Apex Listener. The architecture is depicted below:



Architecturally, this setup resembles the classic configuration, except that a much richer Apex component is present in the middle tier between the web browser and the Apex engine residing on the database. The Apex Listener is configurable from a web page and includes a rich set of configuration options including security options, database connectivity options and caching options.

*Advantages:*

1. Potentially provides greatest security for Application Express implementations.
2. Provides simple configuration (web page or properties file).
3. Provides numerous configuration options related to debugging, security and performance.
4. Provides a choice of web server technologies.
5. Isolated from the database tier.

*Disadvantages:*

1. Brand new, relatively unproven technology.
2. Involves many software components.

## **HARDENING APEX**

Four specific security vulnerabilities will be addressed in this section; the final section of the paper will enumerate some fundamental “best practices” for ensuring maximum security in deployed Apex applications. The four vulnerabilities are:

- SQL Injection
- Cross-site scripting
- URL tampering
- Eavesdropping

## **EAVESDROPPING**

Eavesdropping refers to any number of techniques used for unauthorized viewing of the traffic occurring in a legitimate session. This can be done a variety of different ways including trojan horse software (trojans) on the workstation being intercepted and network packet sniffers which may be operating on either endpoint, or at some juncture within the network infrastructure.

The best defense against eavesdropping of any kind is to establish a secure connection between the web browser and the Apex application server. Imposing SSL or TLS on an HTTP connection makes it an HTTPS connection. All of the Apex brokering schemes support HTTPS and Apex can even be configured to *require* that all of its connections be secure connections. In addition, there may be requirements from audit teams and other organizational security personnel to encrypt data in transit.

The technologies for encrypting network traffic are ubiquitous, proven and governed by standards, ensuring seamless interoperability with a diversity of operating systems and web browsers.

The steps and required tools for enabling SSL/TLS encryption differ depending upon the web server technology in use. The high-level steps are enumerated here, and references to detailed implementation instructions follow:

Ensure that the required implementation tools are in place. For example, for Oracle HTTP Server and for XMLDB EPG connections, Oracle Wallet is a required tool. For non-Oracle implementations, the keytool utility supplied with Java JDK 1.3 and above is a tool of choice. If you are implementing the mod\_ssl module on Apache HTTP Server, it also requires that the OpenSSL Library (<http://openssl.org/>) be installed and operational.

Utilize the just-referenced utilities to generate a certificate request. Generally, the information requested will include the CN, or name of the server on which the certificate will be installed as well as the company name, organizational unit, city, state/province and country. The generation of a certificate request will result in a display output, or a file, with contents resembling those below. This is a PEM, or Privacy Enhanced Mail, encoded certificate request:

### **Example of a PEM-encoded certificate request (snakeoil.crt)**

```
-----BEGIN CERTIFICATE-----
MIIC7jCCAlEgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBqTELMAkGA1UEBhMCWFkx
FTATBgNVBAGTDFNuYWt1IERlc2VydeETMBEGA1UEBxMKU25ha2UgVG93bjEXMBUG
A1UEChMOU25ha2UgT21sLCBMdGQxHjAcBgNVBAsTFUN1cnRpZmljYXR1IEF1dGhv
cm10eTEVMBMGA1UEAxMMU25ha2UgT21sIENBMR4wHAYJKoZIhvcNAQkBFg9jYUBz
bmFrZW9pbC5kb20wHhcNOTgxMDIxMDg1ODM2WhcNOTkxMDIxMDg1ODM2WjCBpzEL
MAkGA1UEBhMCWFkxFTATBgNVBAGTDFNuYWt1IERlc2VydeETMBEGA1UEBxMKU25h
a2UgVG93bjEXMBUGA1UEChMOU25ha2UgT21sLCBMdGQxHjAcBgNVBAsTDld1YnNl
cnZlciBUZWFtMRkwFwYDVQDExB3d3cuc25ha2VvaWwuZG9tMR8wHQYJKoZIhvcN
AQkBFhB3d3dAc25ha2VvaWwuZG9tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDH9Ge/s2zcH+da+rPTx/DPRp3xGjHZ4GG6pCmvADIEtBtKBFACz64n+Dy7Np8b
vKR+yy5DQGiijsH1D/j8H1GE+q4TZ8OFk7BNBFazHxFbYI40KMiCxdKzdiflyfaa
lWoANFlaz1SdbxeGVHoT0K+gT5w3UxwZKv2DLbCTzLZyPwIDAQABoyYwJDAPBgNV
HRMECDAGAQH/AgEAMBEGCWCsSAGG+EIBAQQEAWIAQDANBgkqhkiG9w0BAQQFAAOB
gQAZUIHAL4D09oE6Lv2k56Gp380BDuILvLg1v1KL8mQR+KFjghCrtpqaztZqcDt
2q2QoyulCgSzHbEGmi0EsdkPfg6mp0penssIFePYNI+/8u9HT4LuKMJX15hxBam7
dUHziCxBVC1lnHyYGjDuAMhe3961Yan8bCld1/L4NMGBCQ==
-----END CERTIFICATE-----
```

The foregoing PEM certificate request should be submitted to a certificate authority (also called CA for short). Ordinarily, a certificate request is initiated from the CA's web site. You use a PASTE operation to transfer the PEM text – including the first and last lines – into a designated window on the CA's web site. A nominal fee may be charged, depending upon the expiration of the certificate; 90-day trial certificates are sometimes provided at no charge. The CA may request additional information that enables them to authenticate your identity and authority to request a certificate; often, this will take the form of an e-mail verification to the registered technical contact for the domain name being registered, to which the recipient must respond before the certificate will be issued.

Some organizations may have a specific preferred CA that must be used; in other cases, the organization has a policy of issuing “self-signed” certificates, typically administered by the organization's network administration unit. Either way, it is essential that the organization's web browsers recognize the certificate authority, or ensuing connections may be rejected or may display warning dialogues prior to connecting. Web browsers are delivered with specific certificate authorities that they recognize; an organization might alter or augment this list. The list of recognized certificate authorities can be viewed from the options feature of web browsers<sup>1</sup>:

*For Firefox:* Tools => Options => Advanced => Encryption => View Certificates

*For Internet Explorer:* Tools => Internet Options => Content => Certificates

*For Safari:*  => Advanced => (Proxies) Change Settings => Content => Certificates

*For Chrome:*  => Options => Under the Hood => Manage Certificates

The CA will issue the certificate, sometimes via e-mail, but usually with an e-mail hyperlink to their web-site. The certificate is a binary encoded block resembling the PEM above. Copy it – including the BEGIN CERTIFICATE and END CERTIFICATE lines – from the web-site and paste it into the indicated area (or put it in a file) so that it can be imported into the server's certificate store, using Oracle Wallet or *keytool* or another certificate management tool. The certificate should be imported by the same tool that originated the certificate request in Step 2. The certificate management tool will validate the certificate prior to importing it.

A root certificate may be required as well; the CA's web site can provide more information on this topic, including details of installation. To see the built-in root certificates supplied with Oracle Wallet, navigate to Wallet => Trusted Certificates.

Configuration changes may be required to the (web) application server configuration to allow SSL/TLS connections and to specify which port number will be used for listening for that type of connection. Note that both insecure and secure connections can be accepted concurrently, unless specific steps are taken to disable the insecure port. Following the configuration changes, it is normally necessary to bounce the application server.

Once all of the installation and configuration steps are complete, the connection can be verified using any web browser. Note, too, that any bookmarks or icons that launch the URL will have to be changed to incorporate HTTPS (instead of HTTP) and the new port number in the URL for Apex. For example, if the previous URL was this:

<http://oregano.sagelogix.com:5555/apex/apex>

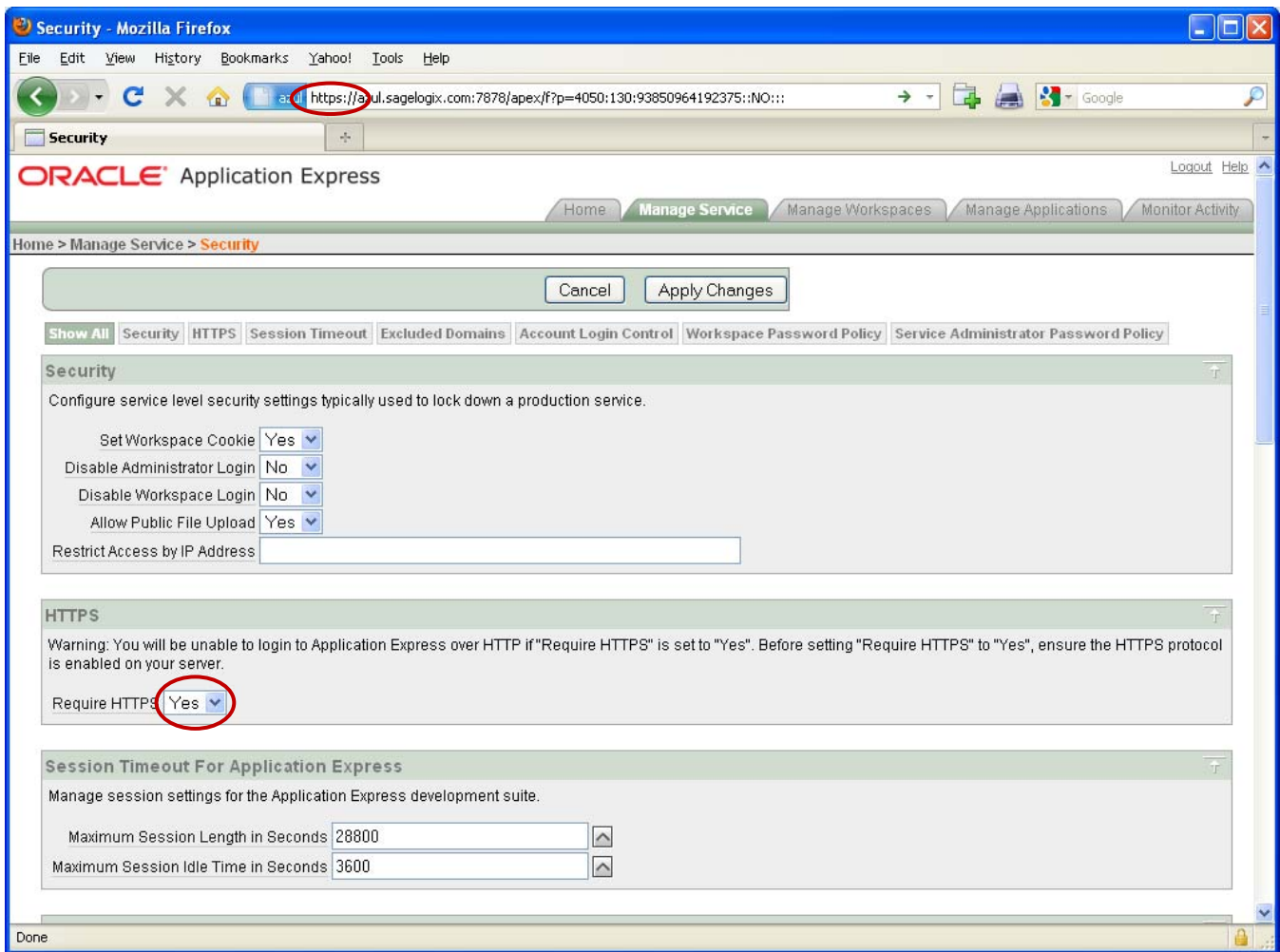
... it would now be changed to something resembling this (port numbers could vary):

<https://oregano.sagelogix.com:7878/apex/apex>

In order to ensure that only secured (encrypted) connections are utilized by Apex, it is strongly recommended that Apex be configured to accept only HTTPS connections<sup>2</sup>. To do this, log into Apex as administrator and navigate from the home page to: Manage Service => Security. Under the block labeled HTTPS, alter the “Require HTTPS” setting to “Yes” and press “Apply Changes”. This is how the page should appear:

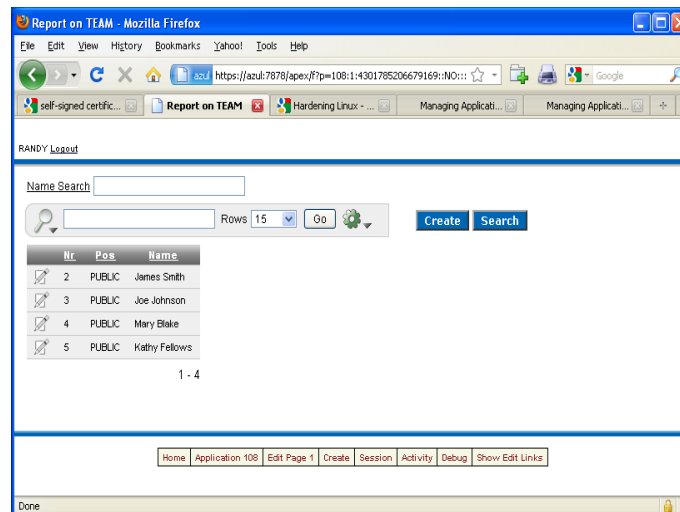
<sup>1</sup> Navigation is slightly different on Mac OS X.

<sup>2</sup> When this option is set, the login page appears but attempts to authenticate into the insecure (http://...) page simply do not work. No error is issued as of Apex 3.2.1.00.12. If this option is set on a database where https is not installed, not working, or where it ceases to work because the certificate is invalid, the option can be changed using SQL on the table WWV\_FLOW\_PLATFORM\_PREFS.

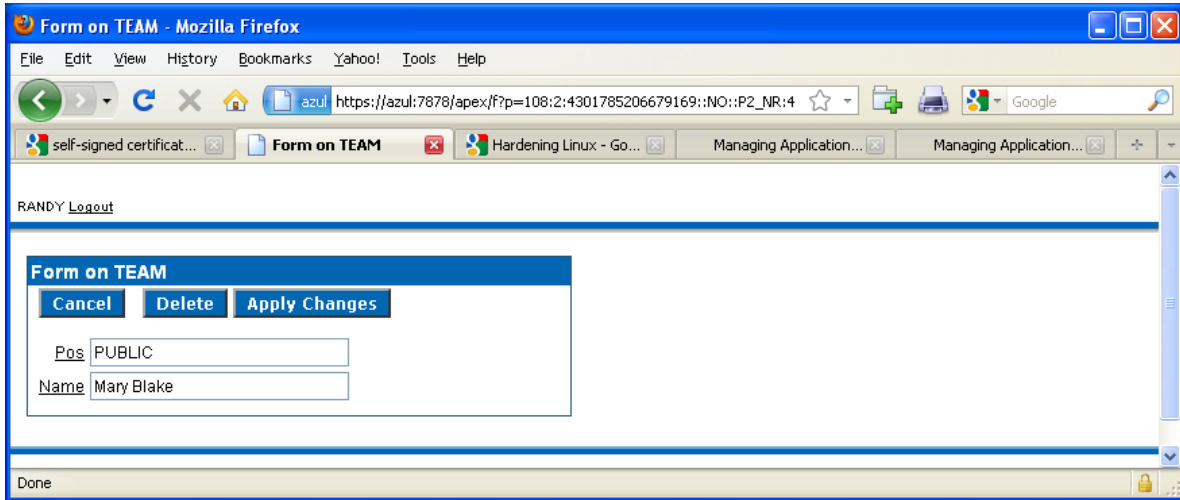


## URL TAMPERING

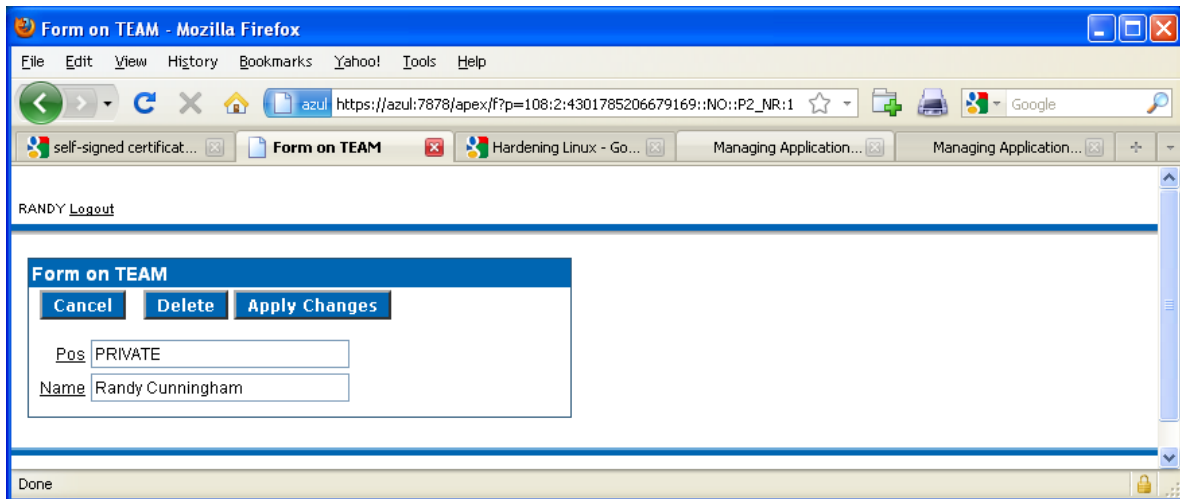
URL tampering refers to any practice where the URL is modified within the navigation pane in order to gain access to pages, privileges or information that is not available using ordinary navigation options within the Apex application. Consider the following example. In this summary screen, I can see 4 persons:



The curious might notice that the sequence numbers (second column) start with 2, so the question might arise, “Are there others?” To find out, I click on the edit icon for any of the rows; in the following example, I click on #4, Mary Blake:



At the end of the URL, the argument “P2\_NR:4” is plainly visible. If I change the 4 to a 1 in this portion of the URL, and resubmit the URL, then this is what I can see:

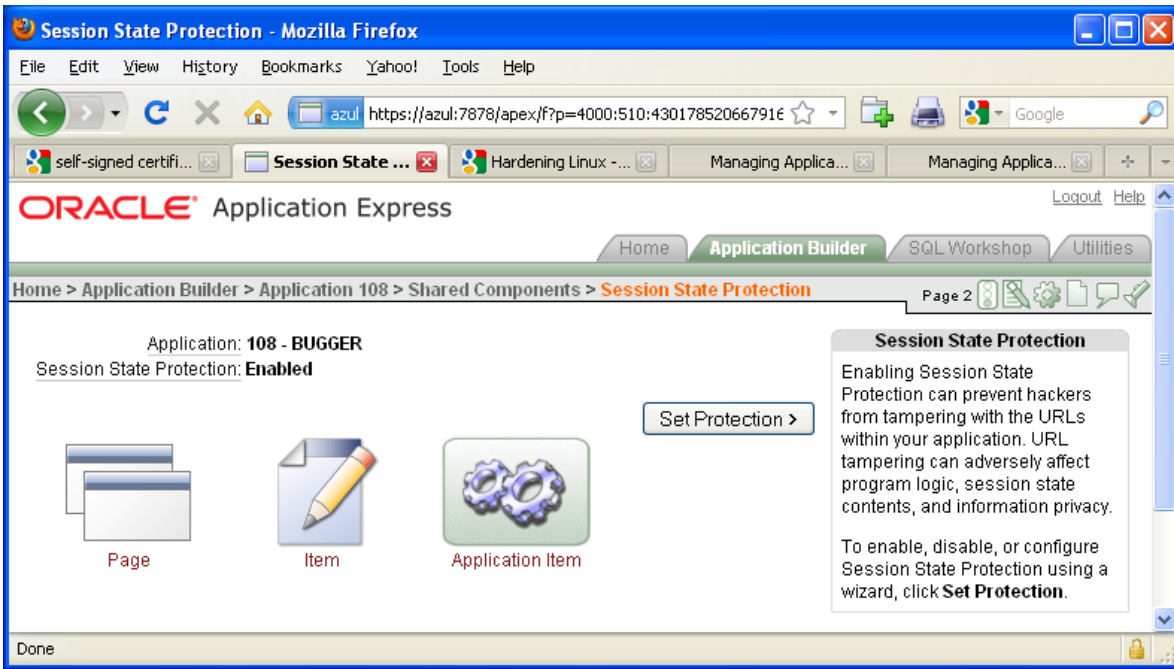


Noticing that this record is not displayed on the summary list on the preceding page, and noticing that the Pos designation for this member is “PRIVATE,” it is circumstantially evident that this record was not intended for prying eyes! Not only that, but this record can be *changed* with the page above, just by altering the enabled fields and pressing “Apply Changes”.

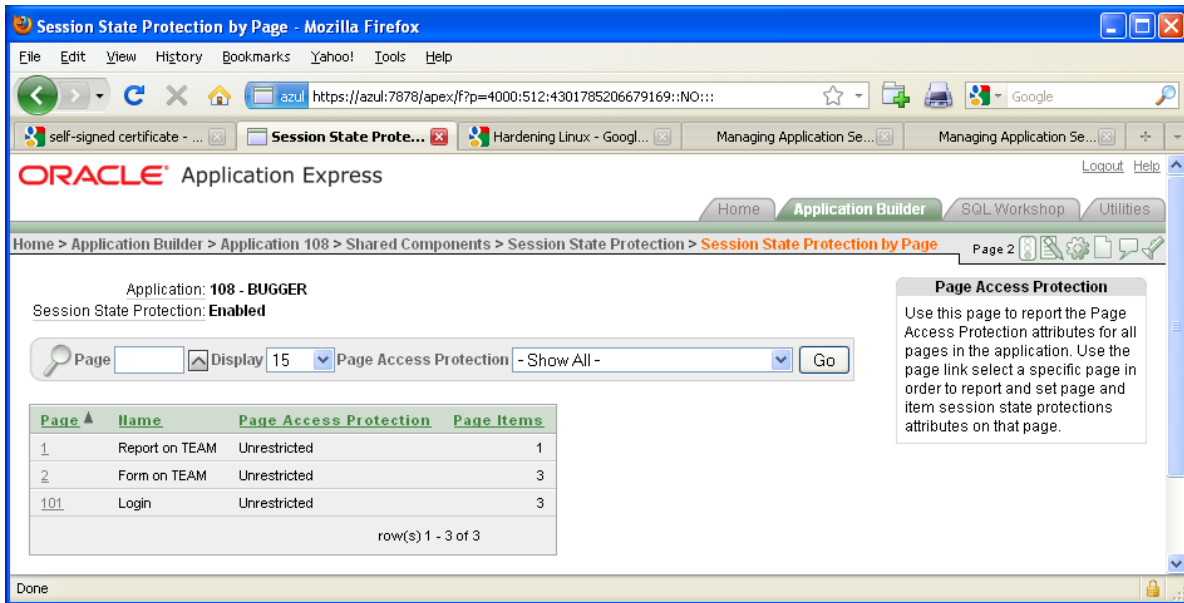
This example is highly simplified. Notice that the application page number is also there and could be changed in a similar way. For a more complex Apex application, where much more information is sent between pages using URLs or POSTDATA, the opportunities for mischief increase exponentially.

### *SOLUTION*

To prevent this from happening, Apex provides Session State Protection. From the application builder’s front page, click on “Shared Components” then, under the Security heading, click on “Session State Protection”. Here is an example of the Session State Protection Page:

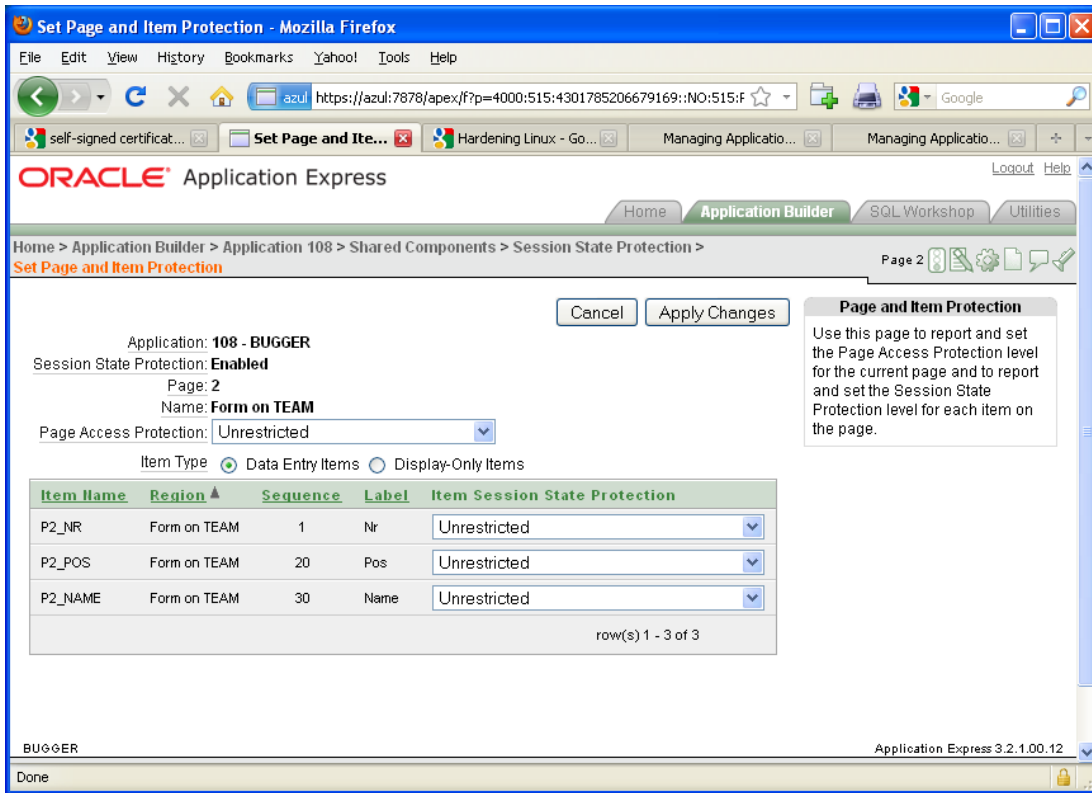


From this page, one can enable, disable or configure Session State Protection with a wizard, or the protection can be applied at various granularity levels including individual pages or even individual items. Clicking on the “Page” icon displays the 3 pages in this application and their Page Access Protection status:



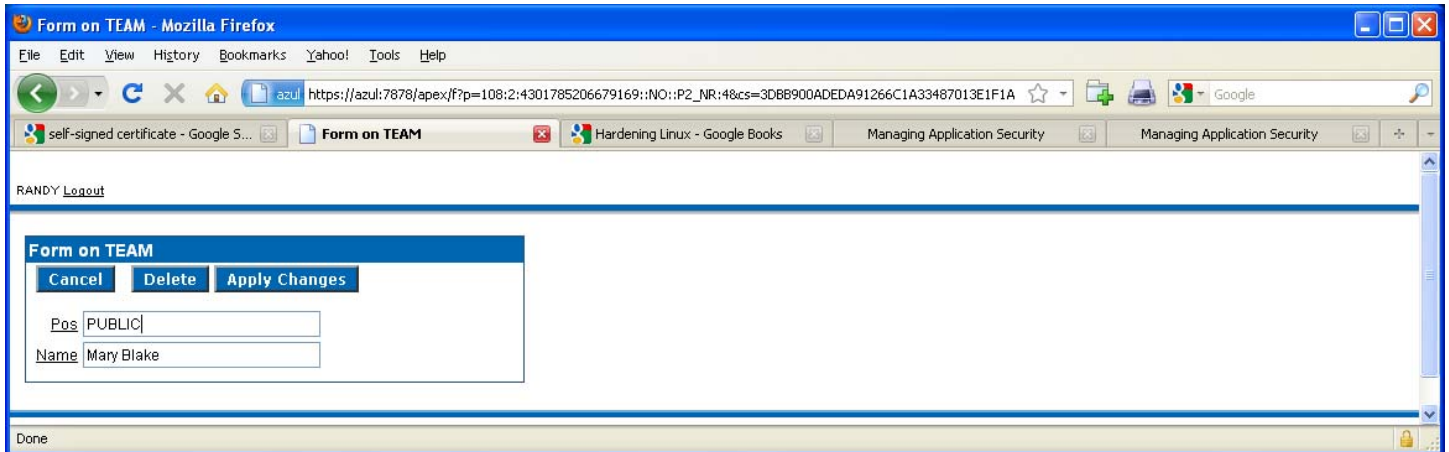
Because it is the form, page 2, which has an identified security vulnerability to be cured, a click on the “2” under Page displays the Page and Item Protection form shown on the following page.

Once again, notice that there is a granularity of choice; the entire page can be protected (Page Access Protection), or individual items can be protected (Item Session State Protection).



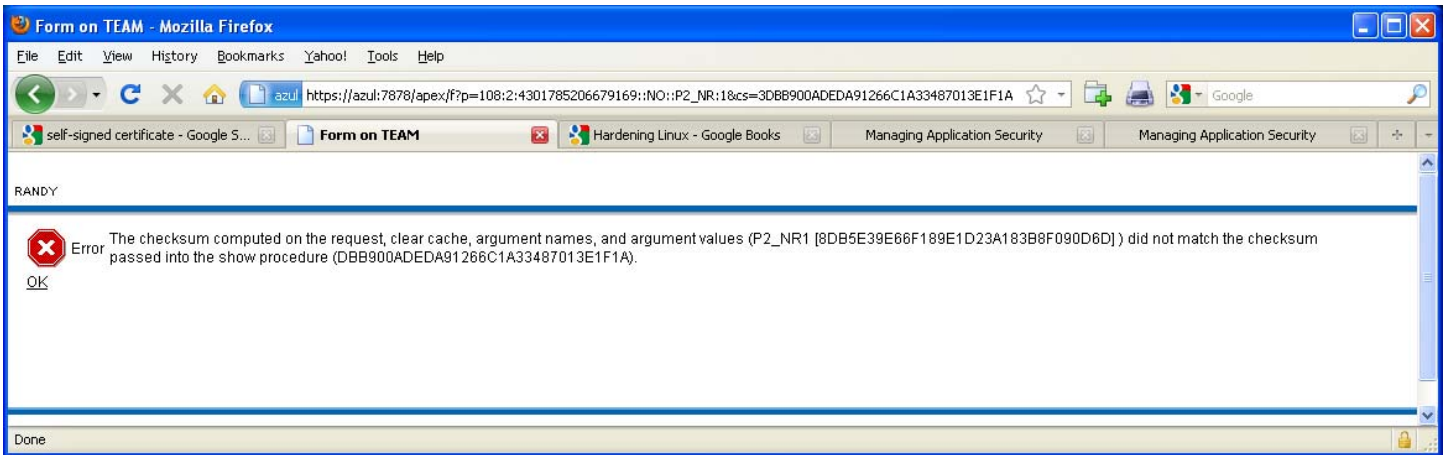
For this particular vulnerability, Page Access Protection is changed from “Unrestricted” in the pull-down list to the selection “Arguments Must Have Checksum”. (The other choices are “No Arguments Allowed” and “No URL Access”.) The changes are saved by clicking on “Apply Changes”.

Now, the identical navigation is done to the summary page and Mary Blake (#4) is selected. The content presented is the same as before, but notice the URL in the navigation pane:



There is now new material, cs=3DBB900A91266C1A33487013E1F1A, in the URL. This is the page checksum.<sup>3</sup> The argument, P2\_NR:4 is still present and if it is changed to be P2\_NR:1 as was done previously, this is the result of the page submission:

<sup>3</sup> This checksum can be tampered as well, but it is likely to be several centuries before the correct checksum is accidentally discovered, even with the help of automation.



Observe, too, that Page Access Protection will safeguard against situations where the operator unintentionally issues keystrokes into the URL and receives perplexing, inconsistent results due to the spurious characters.

### CROSS-SITE SCRIPTING

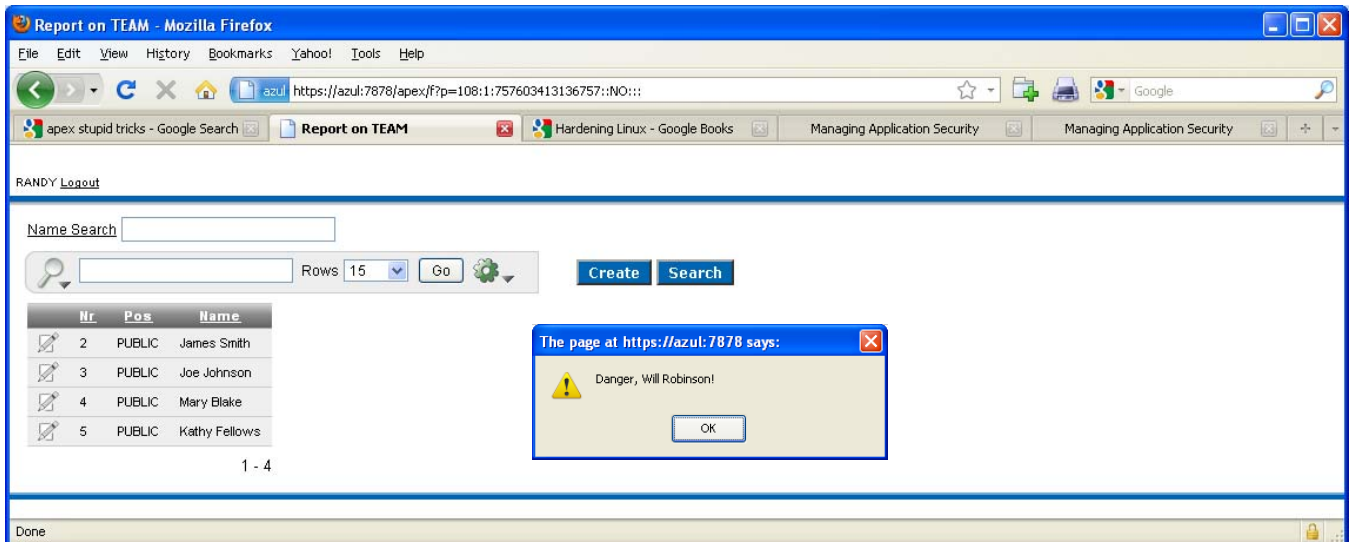
Cross-site scripting (also referred to as XSS) is a security breach that takes advantage of dynamically generated Web pages. In a XSS attack, a Web application is sent a script that activates when it is read by a user's browser. Once activated, such a script can steal data, including session credentials, and route the information to the attacker.

Note that cross-site scripting does not require elevated privileges; knowledge of javascript (for example) and update access to an Apex application or its supporting database is all that is required.

As a rather simple example, we could alter the name in the example application to include a script that presents a pop-up every time a specific row is called up from the database and displayed. This example, while benign, is also quite annoying. To do this, we can use SQL\*Plus, SQL\*Developer or even the Apex Form for this application to add the following text to the name in record key number 4:

```
<script>alert ('Danger, Will Robinson!') </script>
```

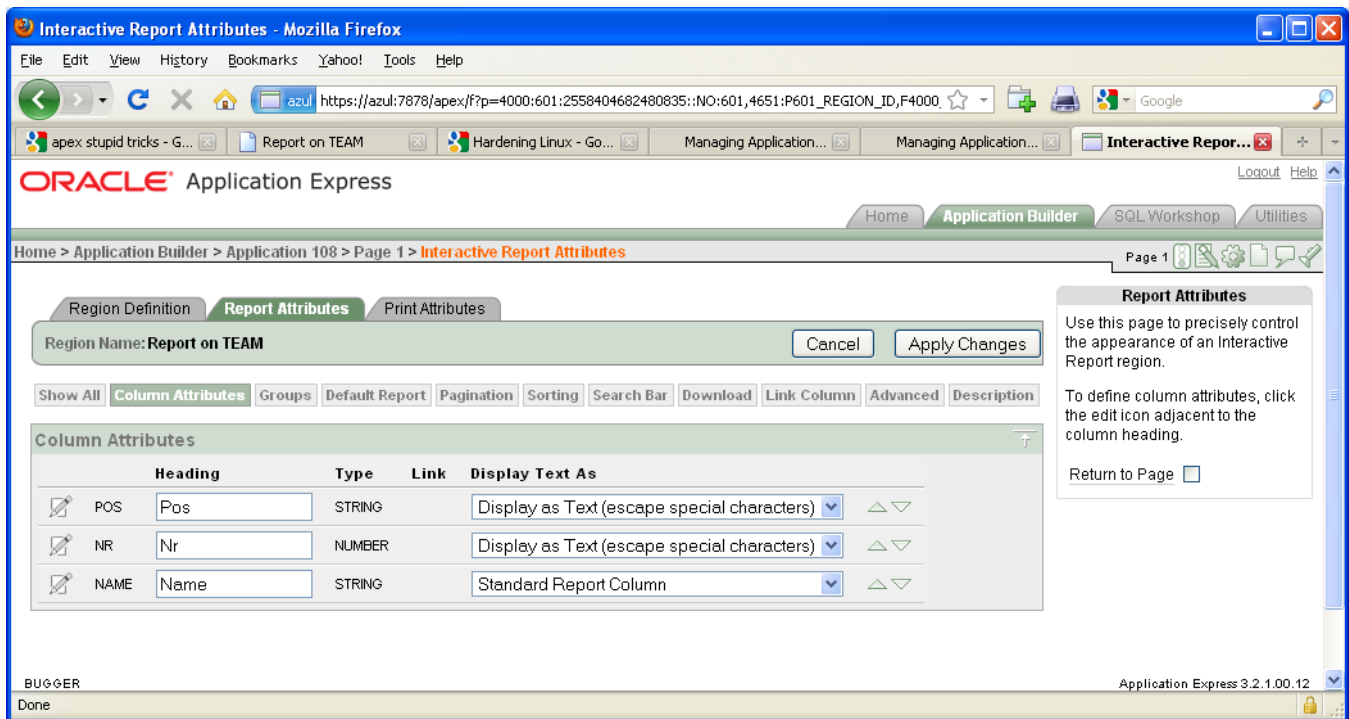
The result, when the report is activated, is as follows:



Notice that, aside from the pop-up, the application appears and functions as it normally does. A quick peek at the generated HTML reveals what is going on:

```
<tr ><td><a href="f?p=108:2:757603413136757::NO::P2_Nr:2&cs=302B2AA80BFD78D2F67A083D76F3235D7" ></a></td><td align="left">2</td><td align="left">PUBLIC</td><td
align="left">James Smith</td></tr>
<tr ><td><a href="f?p=108:2:757603413136757::NO::P2_Nr:3&cs=34AAD1D26A371C66F57BDF1E4FECA9B3" ></a></td><td align="left">3</td><td align="left">PUBLIC</td><td
align="left">Joe Johnson</td></tr>
<tr ><td><a href="f?p=108:2:757603413136757::NO::P2_Nr:4&cs=388A7A9C90A94D5DA8D9EDF7E6693B520" ></a></td><td align="left">4</td><td align="left">PUBLIC</td><td
align="left">Mary Blake<script>alert ('Danger, Will Robinson!')</script></td></tr>
<tr ><td><a href="f?p=108:2:757603413136757::NO::P2_Nr:5&cs=3B3C9B363F7C580A016A6E74A4D8AB030" ></a></td><td align="left">5</td><td align="left">PUBLIC</td><td
align="left">Kathy Fellows</td></tr>
```

To prevent this from happening, special characters should be escaped. A look at the report attributes in the application shows us that Name is defined as a Standard Report Column. This should be changed to Display as Text (escape special characters):



With escaped special characters, no pop-up is generated. A look at the generated HTML shows that we haven't lost any information, but the special characters have been escaped using HTML conventions:

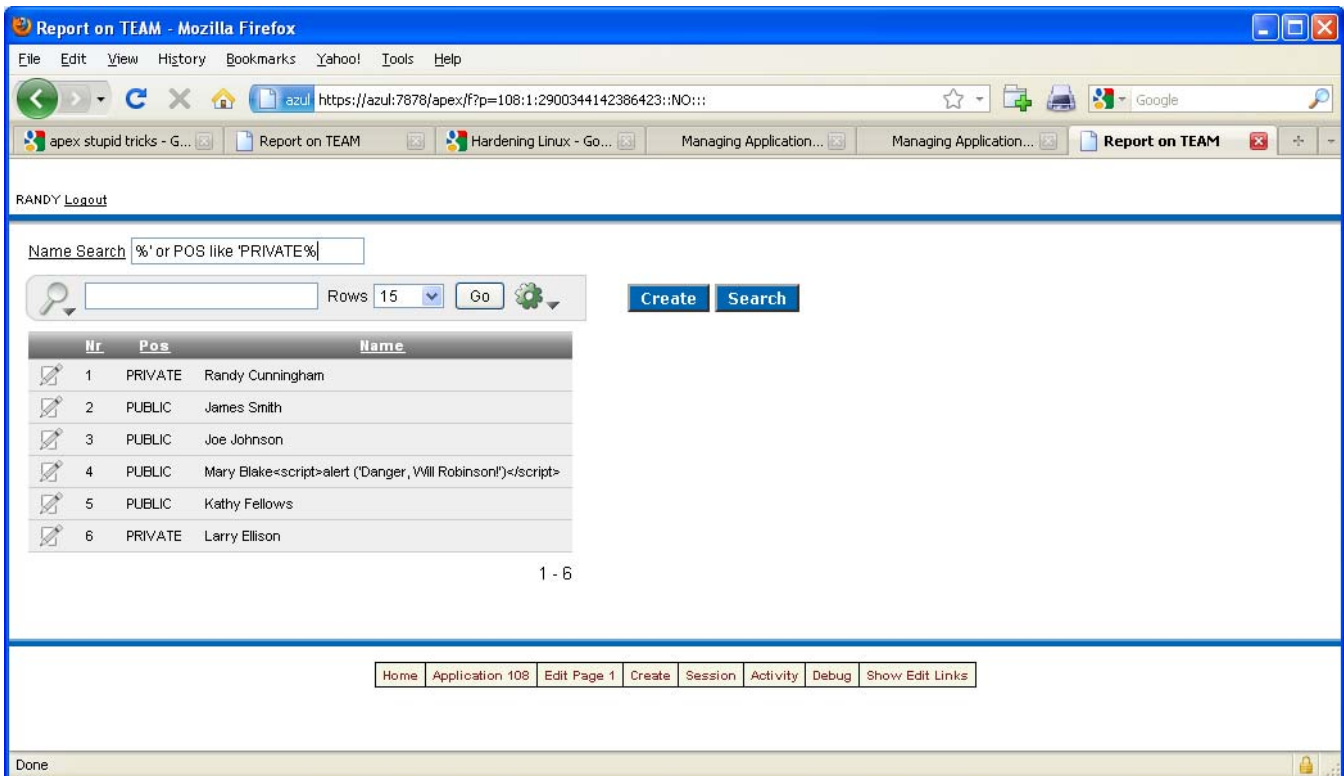
```
<tr ><td><a href="f?p=108:2:2900344142386423::NO::P2_Nr:2&cs=3F300B6FCE14426F52E0F54919368EBBE" ></a></td><td align="left">2</td><td
align="left">PUBLIC</td><td align="left">James Smith</td></tr>
<tr ><td><a href="f?p=108:2:2900344142386423::NO::P2_Nr:3&cs=34361D54B31474E31E449D654B1591AAE" ></a></td><td align="left">3</td><td
align="left">PUBLIC</td><td align="left">Joe Johnson</td></tr>
<tr ><td><a href="f?p=108:2:2900344142386423::NO::P2_Nr:4&cs=39FF32DBAAE397DBB029E174E59FD0552" ></a></td><td align="left">4</td><td
align="left">PUBLIC</td><td align="left">Mary Blake<script>alert ('Danger, Will
Robinson!')</script></td></tr>
<tr ><td><a href="f?p=108:2:2900344142386423::NO::P2_Nr:5&cs=357EA86E682E55692E3FA172FD1F4364A" ></a></td><td align="left">5</td><td align="left">PUBLIC</td><td align="left">Kathy Fellows</td></tr>
```

Observe that the cross-site scripting attempt has not been removed; instead, it will now appear on the displayed element, as evidenced in the following section on SQL injection.

## SQL INJECTION

SQL injection is a security breach that augments the SQL in the application with additional SQL language constructs, often using an input field that might be concatenated into a SQL predicate, for example. Similarly to cross-site scripting, SQL injection requires no special privileges or access to the Apex application or its code.

Using the same application as before, note the Name Search field in this example:



This result is based on a prediction, which is accurate in this case, that the Name Search box plugs directly into a SQL WHERE predicate. So by coding additional SQL – or “injecting” it into the base query, the result set can be expanded in a way unintended by the developer. The actual query that is in the Apex Region Source for this report is:

```
select "NR",
"POS",
"NAME"
from "#OWNER#"."TEAM"
where "POS"='PUBLIC'
and "NAME" like '%&P1_X.%'
```

This results in the following SQL when the input above is supplied:

```
select "NR",
"POS",
"NAME"
```

```
from "#OWNER#". "TEAM"
where "POS"='PUBLIC'
and "NAME" like '%%' OR POS = 'PRIVATE'
```

Mere modification of the predicate is not the only possibility with SQL injection. Consider what would happen if this input were supplied to the Name Search box in this example:

```
XXX' UNION select CODE_NR, PASSWORD, USERNAME FROM SECRET_LIST - -
```

In this case, the entire query has been hijacked to provide an unauthorized inquiry into the table SECRET\_LIST.

The solution to prevent SQL injection<sup>4</sup> is to use bind variables. The query in the Region Source should read this way:

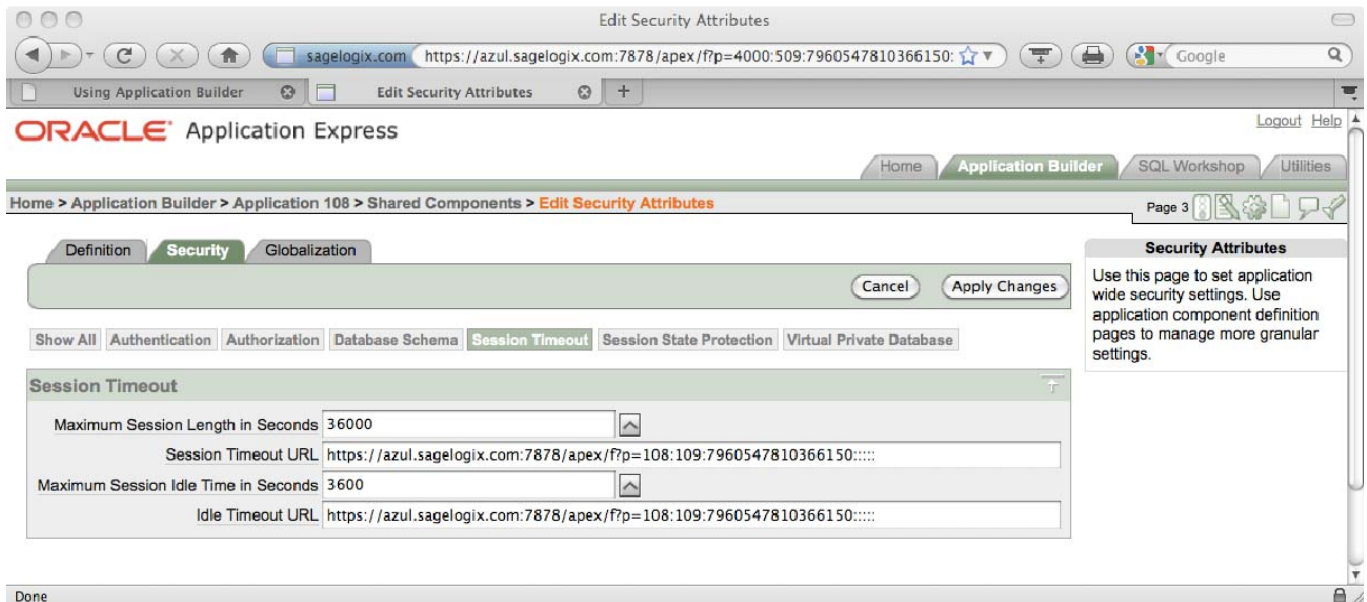
```
select "NR" ,
"POS" ,
"NAME"
from "#OWNER#". "TEAM"
where "POS"='PUBLIC'
and "NAME" like '%' || :P1_X || '%'
```

When this is implemented, attempts to use SQL injection will result in “No data found.”

### ABANDONED SESSIONS: SESSION TIMEOUT

Configuring Session Timeout attributes from within the application can reduce the application’s exposure. People often leave their computers unattended for extended periods of time and do not close applications when departing for the day. Therefore, an unauthorized individual can effortlessly assume the authorized person’s identity within the application. By setting the session and idle timeout appropriately, the exposure to these problems is mitigated.

To navigate to session timeout, from within the application builder for the selected application, choose Shared Components => Edit Security Attributes => Session Timeout.



Note that you can supply a URL for redirection of the session when a timeout occurs. This can provide more helpful information than the default behavior, which simply redirects to the Apex home page.

<sup>4</sup> This solution is applicable not only to Application Express, but to other development tools as well, such as Visual Basic or Java.

## OTHER TECHNIQUES TO HARDEN APEX

These more general steps can be used to minimize the likelihood of security breaches or compromises from within Apex:

- For production deployments, strive to use the Apex run-time environment instead of deploying the development environment. The run-time environment has no web-based administrative or builder components, making it a more hardened environment; administrative functions are performed with an API using SQL\*Plus.
- Decide to use the best authentication scheme available.
- Never accept user data as is. Where possible, use lists of values to restrict choices while enhancing the user experience. Match data types to what the database is expecting and deploy check constraints, referential integrity or triggers in the database to minimize the possibility of malicious or corrupt data.
- Perform sanity checks in Apex on input data and choose appropriate sizes and data types for input data areas.
- Use “safe” data types wherever possible within the Apex application.

## APEX ADMINISTRATIVE SECURITY PRACTICES

A treatise on Apex security would not be complete without examining Apex built-in security configuration features, generally available only to the Apex administrator account. The navigation to this page from the administrative login is Manage Service => Security. The features chosen and their options are subject to such considerations as the size of the user population, the extent of protection desired, authentication methods, organizational policies and – of course – the value and sensitivity of the data accessible to the Apex application.

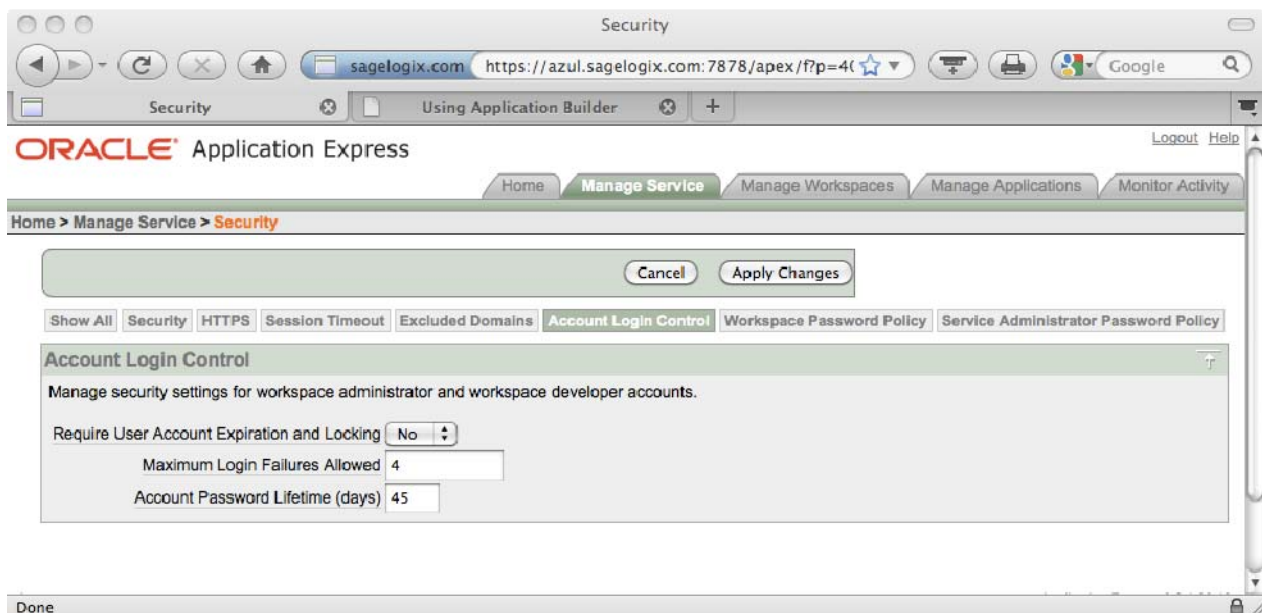
Note that these security features apply to the Apex development suite (Application Builder, SQL Work-bench, etc.) They are not applicable to production implementations using the Apex run-time environment.

### ACCOUNT LOGIN CONTROL

For applications that authenticate using the Apex workspace/user model, thoughtful configuration of account login control is strongly recommended. At a minimum, a maximum number of login failures should be specified, even if it is a fairly high number. This closes the opportunity for brute force penetration of legitimate accounts.

If passwords have expirations, this time limit should probably match the password lifetime policy in use in your organization.

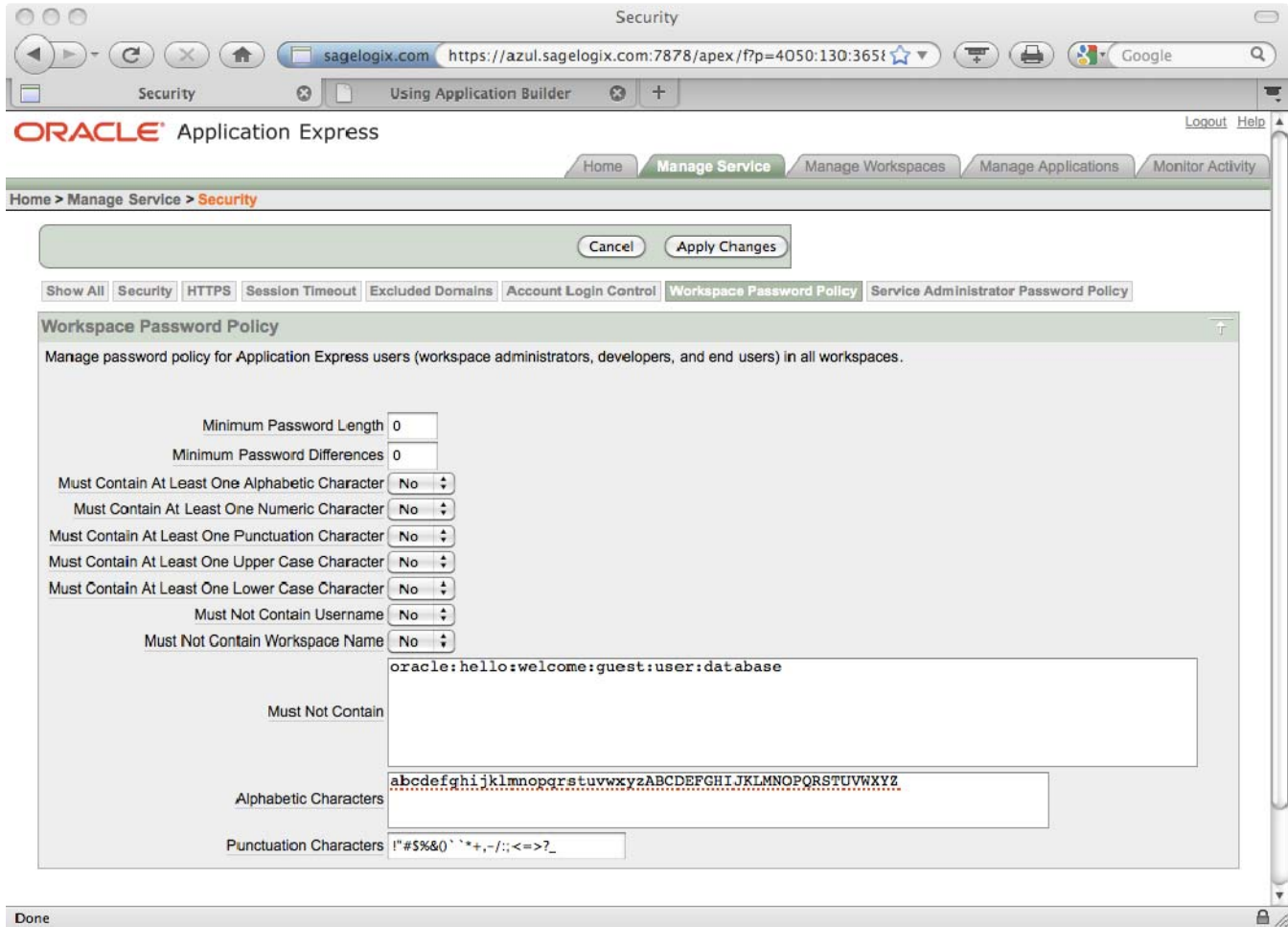
Following is a specimen of the account login control page within Apex:



*WORKSPACE PASSWORD POLICY*

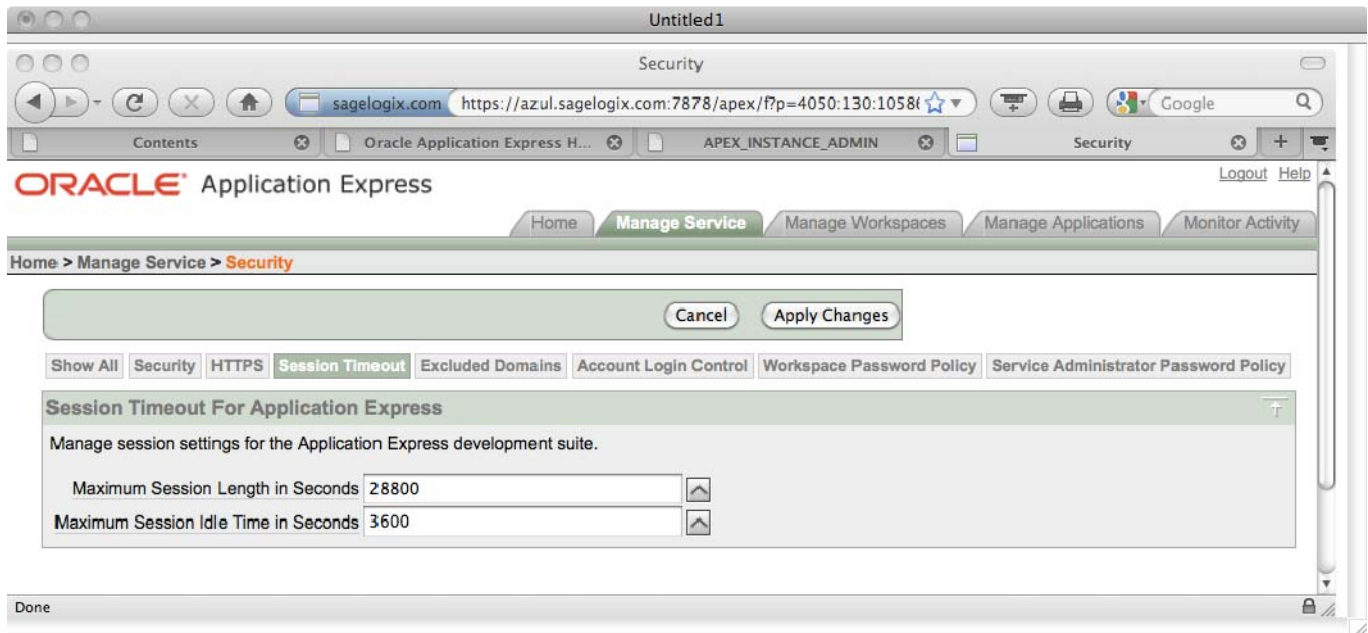
Closely related to the notion of account login control is establishment of a password policy, so that minimum standards of length and enforcement of nontrivial passwords is accomplished. Again, mirroring organizational policy or prevailing practice in your organization is likely to be the most satisfactory option, lacking any other guidance.

Following is a specimen of the Workspace Password Policy page. Most of the options should be self-explanatory.



## SESSION TIMEOUT

As noted above, it is a good practice to reduce exposure to abandoned computers with an open Web browser. This can be configured by the administrator for an entire Apex installation, regardless of the application, and protects the intellectual property that might otherwise be exposed. Navigation is via the administrator home page => Manage Service => Security => Session Timeout:



## HARDENING APEX ALONE IS NOT ENOUGH

Simply focusing on Apex, no matter how much is done to harden it, will be insufficient to ensure adequate security for the Apex application.

Here is a brief checklist of other areas that must be addressed if the overall Apex environment is to be truly secure.

### OPERATING SYSTEM

- Ensure that the operating system is patched to reasonably current levels, especially as regards any security patches.
- Follow the principle of least privilege when granting access to the operating system.
- Lock or remove any accounts that are not needed by the application, especially demo accounts.
- Enforce policies regarding account aging and provisioning, as well as password strength, length and duration.
- Limit the population with access to high-level administrative accounts such as root or administrator.

### ORACLE DATABASE

- Ensure that CPUs have been applied and that relevant security patches, especially for the Apex technology stack, are applied on a current basis.
- Follow the principle of least privilege when granting access or privileges to database user accounts.
- Enforce organizational policies concerning password strength, length and duration.
- Avoid, if possible, granting update privileges to tables within the purview of the Apex application.
- Use check constraints and referential integrity to ensure that only clean data goes into the database.

### *APACHE WEB SERVER*

- Remove preloaded modules and preinstalled content
- Don't publicize the names/versions of your software on error or other web pages:
  - ServerSignature OFF
  - ServerTokens PROD
- Keep patching up-to-date... this component could be the likeliest point of entry for unauthorized users.

### *NETWORK*

- Because of easy URL access, consider opening the Apex port only to the departments, groups or subnets that will be accessing the application.
- Think twice before implementing EPG on a wide scale, as this places the entire database outside of the security perimeter.

### **RESOURCES**

- [1] [Pete Finnegan Web Site](#) This is the most comprehensive independent source of information on Oracle Security. Contains many white papers and presentations, all on the subject of Oracle security (mostly, but not entirely, around the RDBMS).
- [2] [Caleb Sima, Securing Oracle Application Server](#) This paper focuses on security issues for the Oracle Application Server.
- [3] [Alexander Kornbrust, "Hardening Oracle Application Server 9i and 10g"](#) Another treatise on improving security within the Oracle Application Server environment.
- [4] [Oracle Apex Documentation Library](#) This is the entire reference set for Apex 3.2.
- [5] [Pete Lorenzen's Apex Web Site](#) This web site is an excellent source of information on Apex security.