Mobile Security Jump Start

Wayne Henshaw & Mike Jacobs
Progress OpenEdge
October 8, 2013
Agenda

- Architectural basics
  - REST service
  - Mobile client

- Making required choices
  - Authentication model
  - User sessions
  - AppServer SSO

- Diving into the code
  - REST service
  - Mobile client

- What to do when things go sideways
Some Assembly Required...

- OpenEdge (OE) Web applications provide the starting point for your application’s security

- New in Mobile & REST services you will configure and use these security layers
  
  - Web server (i.e. Tomcat 7+)
    1. Everything will use **SSL/TLS** for web application’s client to web server
    2. **Web server** [login] **session management**
    3. **Web server or Mobile/REST web application** user **authentication**
    4. Mobile/REST **web application** role-based **authorization** to HTTP resources
  
  - OpenEdge AppServer
    1. OpenEdge AppServer for **application** level **authorization**
Web Application Security Goals

- Your web application will be probed by hackers & bots within 60 seconds
- Design and build security into my web application from day 1
- Use strong perimeter security before accessing business servers
  - Use OWASP web application security guidelines (www.owasp.org)
- Use strong, peer reviewed, industry security technologies
- Push identity from perimeter security to back-end servers for application authorization
Anatomy of an OpenEdge Mobile/REST Web Application

- Standard Java web application architecture & functionality
- Spring Security replaces Java container authentication & authorization security
- Combinations of REST api & OpenEdge Mobile components
General REST Web Application Architecture

- Mobile device
- Web browser
- Mobile / REST Web Application
  - HTTP session manager
  - REST web service
  - AppServer
  - AppServer
  - AppServer

- Web.xml (Java web application descriptor)
- appSecurity-xxxx-yyyy.xml
- log4j.properties (Spring & REST Adapter logging)
- WEB-INF/classes/
- WEB-INF/adapter/log/
- WEB-INF/
Using an AppServer for Application Level Authorization

- The Spring Security’s **authentication** credentials are transformed into a sealed Client-Principal that is accessible via SESSION’s REQUEST-INFO object

  **Sealed ABL Client-Principal**
  *(SSO state)*

---

**Diagram Notes:**

- **Mobile device**
- **Web browser**
- **AppServer**
- **Mobile / REST Web Application**
- **REST web service**
- **HTTP session manager**

**Key Features:**

- **Plug-in Authentication**
- **Role-based Authorization**
- **CORS**
CORS - Cross Origin Resource Scripting

- Javascript engines **always** block resource access to a domain external from the page.
- CORS is a W3C group standard that allows Javascript to access Web application resources in a DNS domain different from the one the current HTTP page and JavaScript were loaded from.
  - CORS works by using HTTP headers that allow servers to grant/deny Javascript resource access to permitted client domains.
OpenEdge Mobile Client Architecture

Mobile Device or Web Browser

Mobile App

Session

JSDO catalog

JSDO

Java Container (http/https)

HTTP session manager

Mobile / REST Web Application

Spring Security

JSDO catalog

REST web service
OpenEdge Mobile Client JavaScript Session Object

progress.data.Session :

• Log in to mobile service, sending necessary credentials

• Get and store the JSDO catalog

• Add session and credential information to the requests that a JSDO sends to mobile service

• Log out
OpenEdge Mobile Client Architecture

Mobile Device or Web Browser

Mobile App

- Session1
  - Catalog1
    - JSDO1
- Session2
  - Catalog2
  - Catalog3
    - JSDO1

Mobile / REST Web Application

- catalog1
  - REST web service 1
- catalog2
  - REST web service 2
- catalog3
Agenda

- Architectural basics
  - REST service
  - Mobile client

- Making required choices
  - Authentication model
  - User sessions
  - AppServer SSO

- Diving into the code
  - REST service
  - Mobile client

- What to do when things go sideways
Major Security Technology Decisions

- **Required choices**
  - The web application’s user authentication model (note: NOT where user accounts exist)
  - The web application’s user login session model (provided by the Web Server)
  - The web application’s role-based authorization (application defined role names or production defined role names)

- **Secondary choices**
  - The web application’s CORS configuration (restrict client domain access)
  - The web application’s AppServer SSO (use Client-Principals to control access to application and OpenEdge)

*Other authentication models available - not certified*
Web Application Authentication Models

- **Anonymous** — The *no user authentication or login session [ default ]*  
  *(NOT recommended for production applications – used for test & debug)*

- **HTTP BASIC authentication** — *Client sends base64 encoded user name/password to web application in each http request*
  - HTTP header: Authorization
  - *No user login session*
  - *No user logout*

- **HTTP FORM authentication** — *The client logs into and out of the web application once per session*
  - HTTP form passed to REST web application for **login** & HTTP session management
  - HTTP cookie returned to client – client echoes cookie for each HTTP request
  - HTTP request used to **logout** from REST web application & delete HTTP session
Choose Your User Login Session Model

- Three different concepts of *session*
  - Mobile client (server connection)
  - Web server user [login] session
  - Application user session

- Only two web application *session* models:
  - *stateless* (BASIC default)
  - *stateful* (FORM default)

- Web servers control user sessions *(not OpenEdge or your AppServer application)*
- Web servers do not share user sessions across web applications
- Client & server web server user session models must ALWAYS agree
OpenEdge Web Application Security Templates

- appSecurity-anonymous.xml
  - Anonymous security - every internet/intranet user is allowed full access
- appSecurity-basic-local.xml
  - HTTP BASIC model using an unsecured user.properties text file
- appSecurity-form-local.xml
  - HTTP FORM model using an unsecured user.properties text file
- appSecurity-container.xml
  - Spring Security SSO from Java container’s authentication token
- 11.2.1+
  - appSecurity-basic-ldap.xml
  - appSecurity-form-ldap.xml
- 11.3+
  - appSecurity-basic-oerealm.xml
  - appSecurity-form-oerealam.xml
Agenda

- Architectural basics
  - REST service
  - Mobile client

- Making required choices
  - Authentication model
  - User sessions
  - AppServer SSO

- Diving into the code
  - REST service
  - Mobile client

- What to do when things go sideways
Example: Choosing the Spring Security Template

- You edit the `web.xml` file to set the security configuration
  - Default location
    - `C:\Progress\OpenEdge\rest\server\WEB-INF`
  - See `param-values` in the `<!--USER EDIT` section for `contextConfigLocation`

```xml
<!-- BEGIN:Spring security.definition -->
<!--
   - Location of the XML file that defines the root application context
   - Applied by ContextLoaderListener.
-->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    <!-- USER EDIT: Select which application security model to employ
    /WEB-INF/appSecurity-basic-local.xml
    /WEB-INF/appSecurity-anonymous.xml
    /WEB-INF/appSecurity-form-local.xml
    /WEB-INF/appSecurity-container.xml
    /WEB-INF/appSecurity-basic-ldap.xml
    /WEB-INF/appSecurity-form-ldap.xml
    /WEB-INF/appSecurity-container-ldap.xml
    -->
    /WEB-INF/appSecurity-basic-perealm.xml
  </param-value>
</context-param>
```
Example: Choosing the Session Management Model

Controls Tomcat session manager
{stateless | never | always | ifRequired}

HTTP Basic Real name shown to users
Example: User Account Authentication Control

Username and encoded password and authorities
**user.properties**

- NOT A SECURE SOURCE OF PRODUCTION USER ACCOUNTS
- Simple to maintain source of user roles and roles for testing
- Format: `<userid>=<password>,ROLE_<rolename>[,ROLE_...],{enable|disable}`
- Clear-text password
- All role names have “ROLE_” prefix (so Spring can distinguish between userids & roles)
- Must restart web application for edits to take affect

```
restuser=password,ROLE_PSCUser,enabled
restmgr=password,ROLE_PSCAdmin,ROLE_PSCOper,ROLE_PSCUser,enabled
restoper=password,ROLE_PSCOper,ROLE_PSCUser,disabled
```
Session API at its simplest

```javascript
pdSession = new progress.data.Session();

pdSession.authenticationModel = progress.data.Session.AUTH_TYPE_FORM;

var loginResult = pdSession.login(serviceURI [, uname, pw]);

pdSession.addCatalog(catalogURI);

( create and use JSDO(s) )

pdSession.logout();
```
Session API
Using the Session API: The Reality

+ 11 more …
Session Services in the Mobile App Builder
Session Services in the Mobile App Builder

WHAT THE MOBILE APP BUILDER + TEMPLATES DO FOR YOU

• UI Login and Logout buttons
• UI fields for user to enter credentials
• UI fields and Settings values mapped to the calls made to the server
• Event handlers for the Login and Logout buttons
• Error and Success handlers for the Login service

WHAT YOU DO FOR THE MOBILE APP BUILDER

• Define three settings
## Session Service Settings

<table>
<thead>
<tr>
<th>Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>authenticationModel</td>
<td>basic</td>
</tr>
<tr>
<td>authenticationResource</td>
<td>/static/home.html</td>
</tr>
<tr>
<td>catalogURIs</td>
<td><a href="http://MyMachine:8980/MyService/static/mobile/MyCatalog">http://MyMachine:8980/MyService/static/mobile/MyCatalog</a></td>
</tr>
<tr>
<td>serviceURI</td>
<td><a href="http://MyMachine:8980/MyService">http://MyMachine:8980/MyService</a></td>
</tr>
</tbody>
</table>
Session Service Settings

- **catalogURIs**: http://MyMachine:8980/MyService/static/mobile/MyCatalog
Session Service Settings

- **Name**: serviceURI
  - **Default value**: http://MyMachine:8980/MyService
var appconfig = {
    "serviceURI": "http://MyMachine:8980/MyService",
    "tableName": "Customer",
    "resourceName": "Customer",
    "tableRef": "ttCustomer",
    "listFields": "Name"
};
Example: OpenEdge CORS support

1. Identify and open the security configuration you applied to your REST application
2. In the security configuration file, appSecurity-XXX.xml, uncomment only the required properties and you must assign a value to those properties
AppServer Single Sign-On

- ClientPrincipal authentication token created from Spring authentication token
- ClientPrincipal passed with each request to Agent
- AppServer client request context information available via:
  - `session:current-request-info:GetClientPrincipal()`.
  - `session:current-request-info:procedureName`.
- ABL Client-Principal handle can be UNKNOWN is using Anonymous security model
- ABL Client-Principal SESSION-ID attribute can be zero (0)
  - BASIC authentication with default stateless session model
- Same Client-Principal validation using `domain-name` and `domain-access-code`
- Cannot use with OpenEdge AppServer before 11.2
Example: OpenEdge Client-Principal Single Sign-On

AppServer Single Sign-On

Security filter that turns a Spring token into an OpenEdge CLIENT-PRINCIPAL
Agenda

- Architectural basics
  - REST service
  - Mobile client
- Making required choices
  - Authentication model
  - User sessions
  - AppServer SSO
- Diving into the code
  - REST service
  - Mobile client
- What to do when things go sideways
Diagnostic Steps: Browser Development Tools

Use browser from Windows or OS-X

- Browser development tools:
  - Console
  - Network traffic
  - Debugger ("Sources")
  - etc.

Inspect Element from context menu

Developer Menu (different places on different browsers)

Shortcut keys (e.g., F12 on Chrome, Firefox)
Diagnostic Steps: Check the Traffic

- Check the HTTP(s) traffic between your application and the server
  
  - 200
  - 4xx
  - 5xx
  
  - No request being sent!
  
  - Server returns 200 but you're getting a NETWORK ERROR on client

- From browser developer tools Network tab

- Standalone HTTP monitor
HTTP Monitors: Fiddler 2 and Other Standalones

Fiddler Web Debugger

HTTP/1.1 401 Unauthorized
Server: Apache-Coyote/1.1
WWW-Authenticate: Basic realm="REST Application"
Content-Type: text/html
Content-Length: 48
Date: Mon, 16 Sep 2013 17:38:48 GMT

<html>
<body>
Unauthorized
</body>
</html>
Check Your Settings!

- Especially if you’re getting strange errors
  Ex: login failure, getting an internal server error on a GET of /static/home.html
Diagnostic Steps: Where Does It Fail?

Access the service directly from a browser address bar

- **Login?**
  
  http://hostname:port/<webApplicationName>

- **Is the catalog accessible?**
  
  http://hostname:port/<webApplicationName>/static/mobile/<catalogFileName>

- **Is the REST adapter available?**
  
  http://hostname:port/<webApplicationName>/rest

- **Can you get data?**
  
  http://hostname:port/<webApplicationName>/rest/<serviceName>/<resourceName>

  Ex: http://localhost:8980/MobileTestApplication/rest/MobileTestService/Customer
Diagnostic steps: What’s Happening on the Back End?

Internal Server Error (HTTP status 5xx), or simply no data

- Is the AppServer running?
- Is the Database running?
- Check logs
- Debug! (see Developer Tools)
Browser Debugger

Useful breakpoints

progress.session.js

- this.login = function
- this.addCatalog = function
- this._openRequest = function

(JSDO uses this to prepare requests)
Debugging Apps Running on Devices

- Try running it in emulator in browser
- Run an HTTP monitor on your computer and set it as a proxy on the device
- Remote debuggers
  - iOS: Web inspector from Safari on OS-X
  - Android: Android Debug Bridge (ADB) through USB connection to computer
  - Weinre (**web inspector remote**)
Logs

- `WRKDIR/` (development) or `CATALINA_BASE/logs` (production)
  - `catalina.<date>.log` or `catalina.out`
  - `localhost.<date>.log`
  - `localhost_access_log.<date>.txt`
- `…/webapps/<web service application>/WEB-INF/adapters/logs`
  - `<service-name>.log`
- `AppServer broker logs`
Debugging in the REST Adapter

- Edit WEB-INF/classes/log4j.properties
- Change ERROR to DEBUG for these packages:

  - Debug core Spring security process
  - Debug Spring Security authn/authz
  - Debug OpenEdge Client-Principal and CORS handling
Troubleshooting Document

- For more information see:

  http://communities.progress.com/pcom/people/mcmann?view=overview
Summary

- Authenticate and authorize at the perimeter
- Client and server have to agree on authentication & session model
- Change the code on both client & server before testing
- Integrate OE Realm after local authentication works
- Beware of CORS configurations during initial testing
- Integrate AppServer SSO after the other things are done