WebSphere LTPA-based authentication

This tutorial covers the following topics:

- Introduction to WebSphere LTPA-based authentication
- Understanding the server-side authentication options
- Configuring MobileFirst Server for LTPA authentication
    - Configurations for WebSphere Application Server
    - Additional steps for Option 1
    - Optional steps for protecting the MobileFirst Operations Console
- Creating client-side authentication components
- Examining the result

Introduction to WebSphere LTPA-based authentication

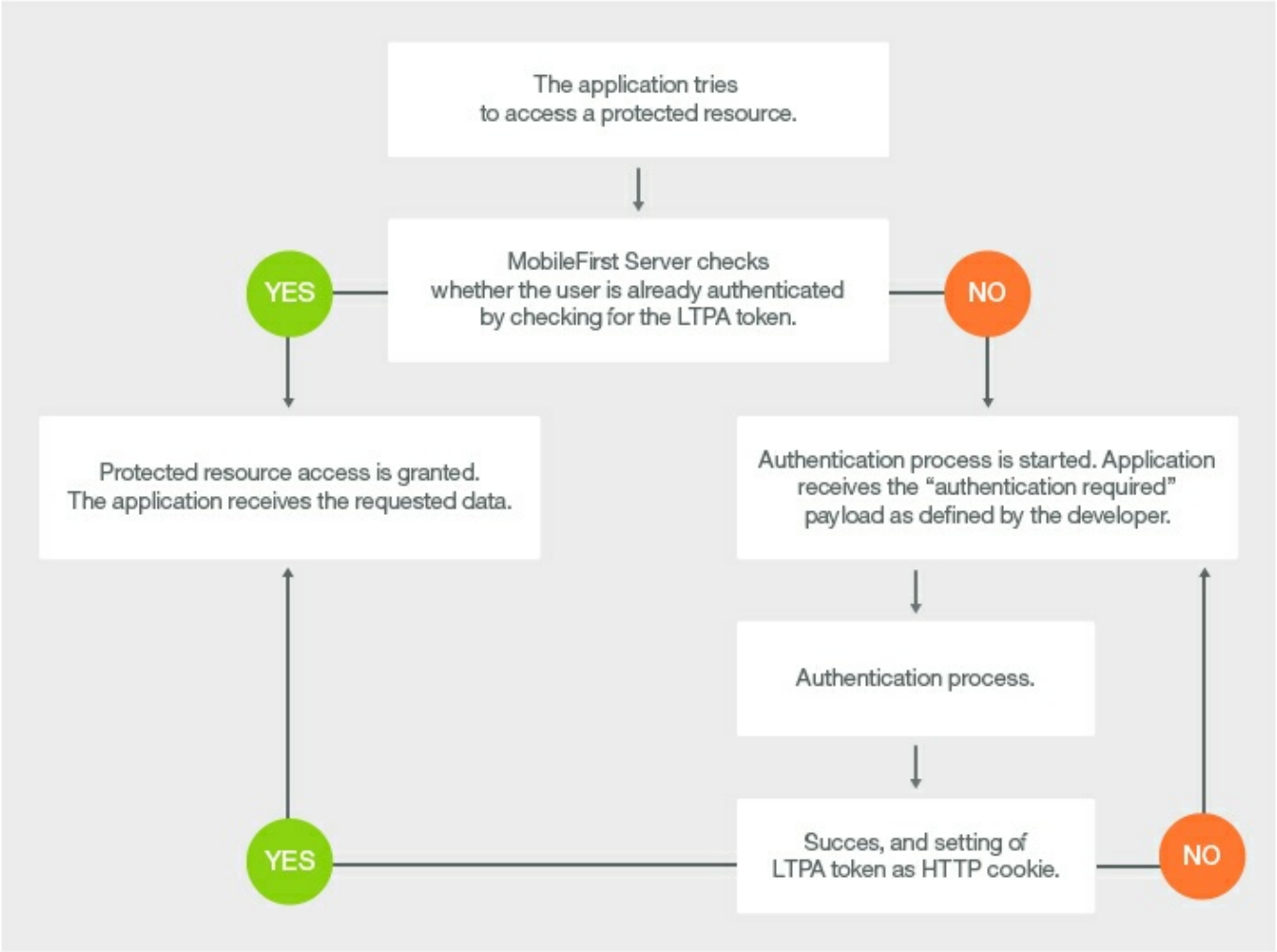
Websphere Application Server uses a secure token in a Lightweight Third-Party Authentication (LTPA) cookie to verify authenticated users. WebSphere Application Server also uses this mechanism to trust users across a secure WebSphere Application Server domain.

When you run IBM MobileFirst Platform Foundation on WebSphere Application Server, you can use `WebSphereFormBasedAuthenticator` and `WebSphereLoginModule` to authenticate to the MobileFirst app by using an LTPA token.

Two options, referred to as Option 1 and Option 2, are available to support WebSphere LTPA-based authentication for MobileFirst apps.

Understanding the server-side authentication options

This diagram illustrates the WebSphere LTPA-based authentication process.



Option 1: Authentication is enforced by WebSphere Application Server

If the enterprise policy requires WAR files to be protected on secured instances of WebSphere Application Server, you can use Option 1 to handle this situation.

Secure the web resources in the MobileFirst project WAR file by specifying the resource and user role.

- The Authenticator and Login Module that are defined as part of this configuration authenticate the user (based on the provided credentials) by using the underlying WebSphere Application Server security API. This mechanism means that if the user provides a user name and password on initial log-in, this data is used to authenticate the user against the underlying registry on which the WebSphere Application Server configuration is based. Otherwise, if a valid LTPA token is provided on subsequent access, then this LTPA credential is used.

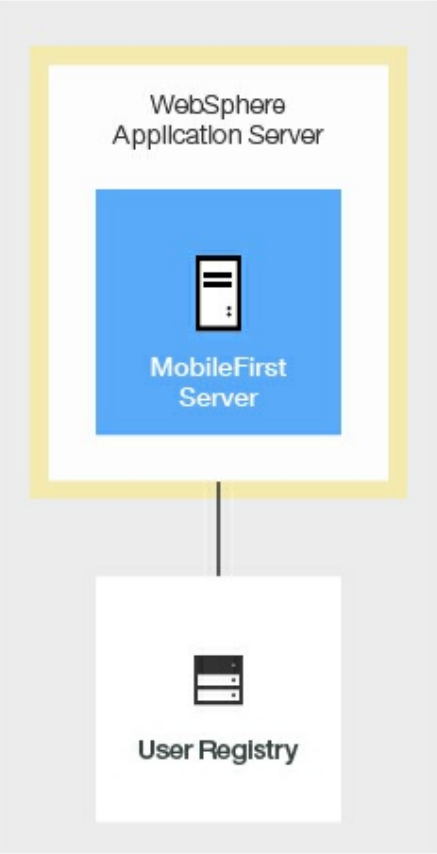
Option 2: MobileFirst Server enforces authentication by relying on the WebSphere Application Server configuration

With Option 2, the MobileFirst security configuration handles user authentication at the MobileFirst server level, by using the security configuration of the underlying WebSphere Application Server instances.

- The MobileFirst Platform project that is deployed as a WAR file on WebSphere Application Server is not secured. The `web.xml` file of the WAR file does not reference any security constraints that protect the web resources.
- The authenticator and login module that are defined as part of this configuration authenticate the user

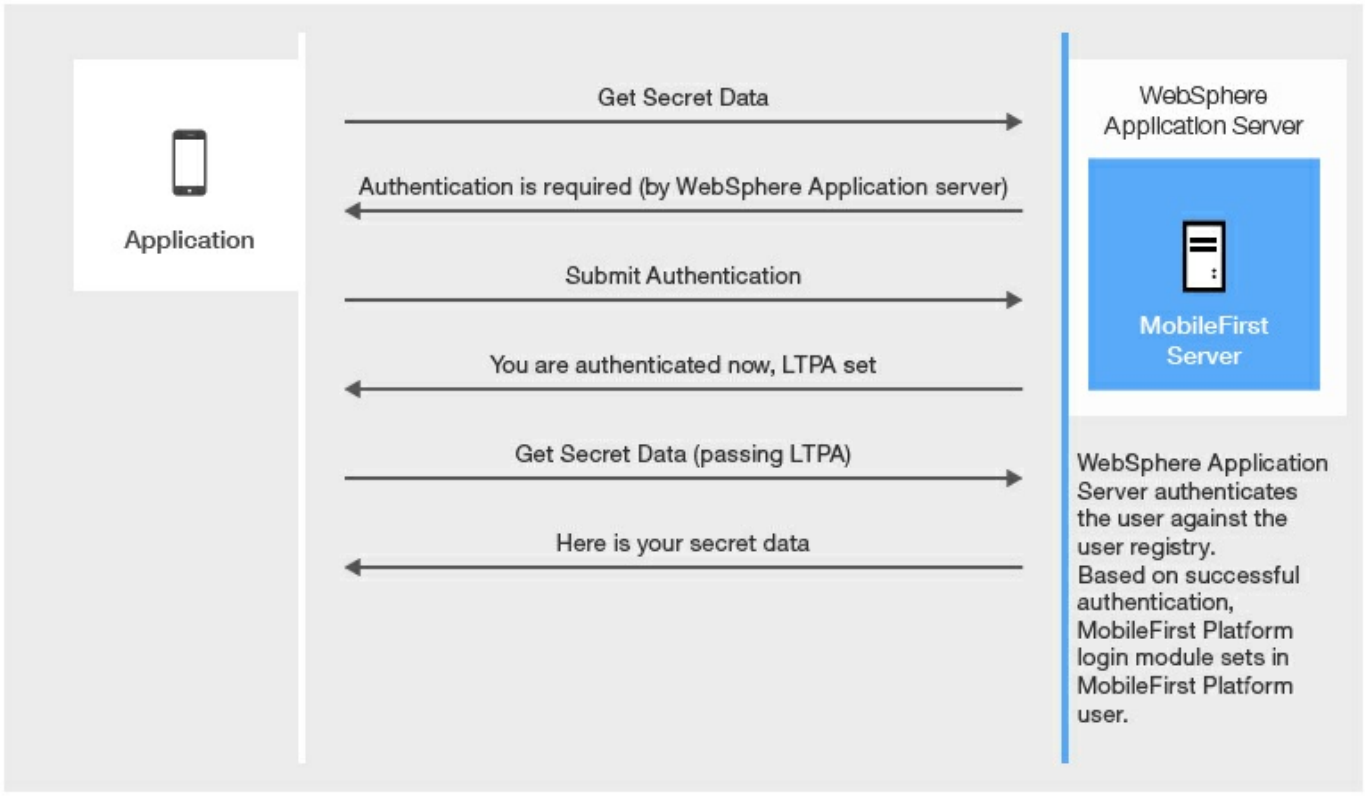
(based on the provided credentials) by using the underlying WebSphere Application Server security API. This mechanism means that if the user provides a user name and password on initial log-in, this data is used to authenticate the user against the registry on which the WebSphere Application Server security API is based. Otherwise, if a valid LTPA token is provided on subsequent access, this LTPA credential is used.

Option 1: Authentication is enforced by WebSphere Application Server.

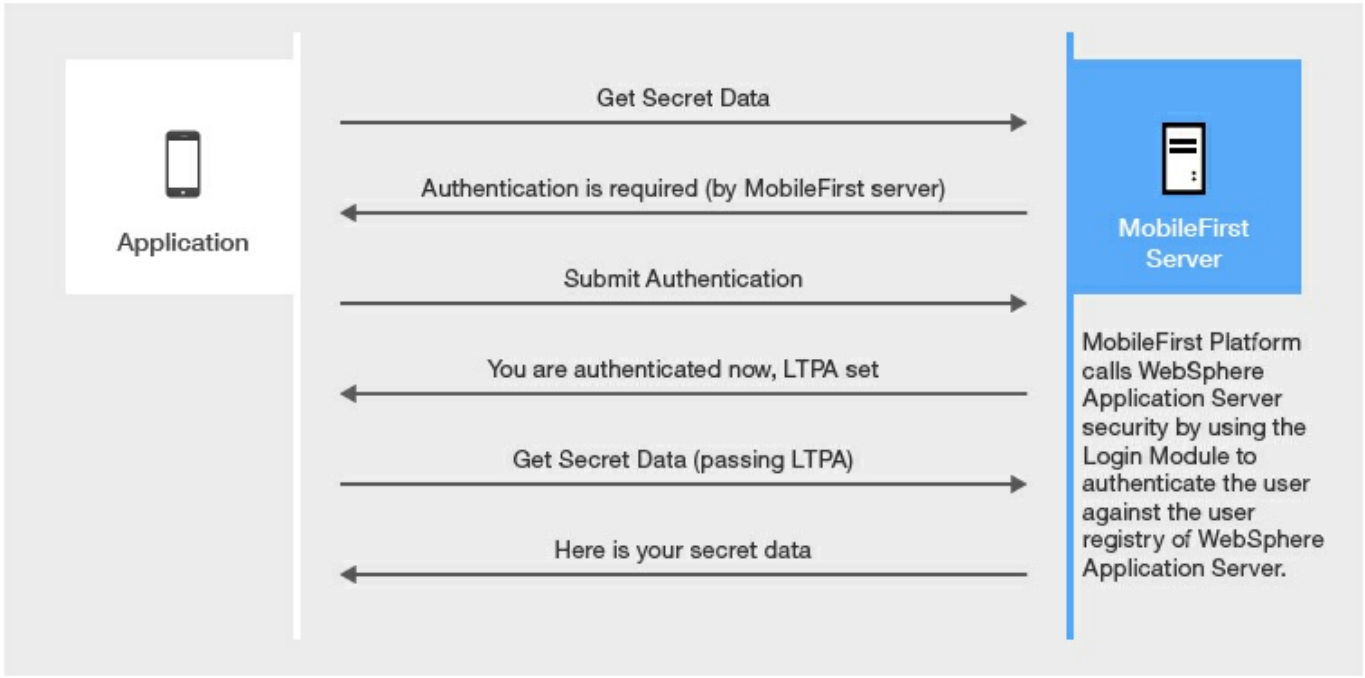


Option 2: Authentication is enforced by MobileFirst Server, relying on the WebSphere Application Server configuration.

Option 1:



Option 2:



	Option 1	Option 2
Benefits	This option uses the traditional WebSphere Application Server authentication and trust model. The container enforces all security. Therefore, it can reuse existing investments in securing the Java™	This option uses the traditional WebSphere Application Server authentication and trust model without the impact of modifying the MobileFirst project WAR file. The container enforces all security. Therefore, it can reuse existing investments in securing the Java™

	Enterprise Edition (Java EE) container by using SSO products from other software vendors.	Enterprise Edition (Java EE) container by SSO products from other software vendors. The layered authentication of device, application, application instance, and user works as intended. Flexibility is gained by configuring security settings that are specific to the MobileFirst runtime without being hindred by the underlying container security.
Usage	This option is suitable for scenarios where the devices can be trusted and access for rogue applications is restricted.	This option is suitable for scenarios where the devices or the apps on the devices cannot be trusted. The multistep authenticity checking that is built into the MobileFirst Server ensures denial of services to jail-broken devices, rogue applications, and unauthorized users.

Based on these benefits, if your business does not require Option1, Option 2 is best.

Configuring MobileFirst Server for LTPA authentication

Configurations for WebSphere Application Server

Step 1: Enable WebSphere Application server security

To compare the two options, you must first define the following settings on WebSphere Application Server:

For option 1:
Enable administrative security
Enable application security

For option 2:
Enable administrative security

Step 2: Configuring authenticationConfig.xml realm and authenticator

1. Find the `authenticationConfig.xml` file in this directory: `{WAS_HOME}/profiles/{your profile}/installedApps/{your node}/{MobileFirst Platform EAR}/{MobileFirst Platform WAR}/WEB-INF/classes/conf` and uncomment the realm under the `For websphere` comment to obtain the following text:

```
<!-- For websphere -->
<realm name="WASLTPARealm" loginModule="WASLTPAModule">
   <className>com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator</className
>
   <parameter name="login-page" value="/login.html"/>
   <parameter name="error-page" value="/loginError.html"/>
</realm>
```

2. Optionally, include the `cookie-domain`, `cookie-name`, and `httponly-cookie` parameters. For more information, see the topic about the LTPA authenticator in the product documentation.
3. Uncomment the login module under the `For websphere` comment:

(http://developer.ibm.com/mobilefirstplatform/wp-content/uploads/sites/32/2014/07/WASSecurityPic.png)

```
<!-- For websphere -->
<loginModule name="WASLTPAModule"><
   <className>com.worklight.core.auth.ext.WebSphereLoginModule</className
>
</loginModule>
```

Note: The login module might already be uncommented.

4. Add a security test to the `authenticationConfig.xml` as appropriate.

- Use `webSecurityTest` if you plan to develop for web environments.
- Use `mobileSecurityTest` if you plan to develop for mobile environments.

```
<!-- For websphere -->
<securityTest>
  <webSecurityTest name="wasWebSecurity">
    <testUser real="WASLTPARealm"/>
  </webSecurityTest>
  <mobileSecuirtyTest name="WAS-securityTest"
>
    <testAppAuthenticity/>
    <testDeviceId provisioningType="none"/>
    <testUser realm="WASLTPARealm"/>
  </mobileSecurityTest>
</securityTest>
```

Additional steps for Option 1

Step 1: Create the login.html file

1. Create a file that is named `login.html` and save it to the root of your WAR file: `{WAS_HOME}/profiles/{your profile}/installedApps/{your node}/{MobileFirst WAR}`
2. Set its contents as follows:

```
<html>
  <head></head>
  <body>
    <form action="j_security_check" method="post">
    Username: <input type="text" name="j_username" size="20"><br>
    Password: <input type="password" name="j_password" size="20"><br
>
    <input type="submit" value="Login">
    </form>
  </body>
</html>
```

Step 2: Create the loginError.html file

1. Create the `loginError.html` error page and place it in the root of your WAR file: `{WAS_HOME}/profiles/{your profile}/installedApps/{your node}/{MobileFirst WAR}`. The `loginError.html` page is used when log-in fails.
2. Set its contents as follows:

```
<html>
  <head></head
>
  <body>
    Login invalid.
  </body>
</html>
```

Step 3: Configuring the web.xml file

Important: This step is optional for option 2, but **mandatory for option 1**.

1. Locate the `web.xml` file: `{WAS_HOME}/profiles/{your profile}/installedApps/{your node}/{MobileFirst EAR}/{MobileFirst WAR}/WEB-INF/web.xml`
2. Inside the root tag, add the tags as shown in the code sample. These tags pass to WebSphere Application Server the configuration that the WAR file expects.

```
<security-constraint id="SecurityConstraint_1">
  <web-resource-collection id="WebResourceCollection_1">
    <web-resource-name>Snoop Servlet</web-resource-name>
    <description>Protection area for Snoop Servlet.</description>
    <url-pattern>/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint id="AuthConstraint_1">
    <description>Snoop Servlet Security:+:All Authenticated users for Snoop Servlet.</descripti
on>
    <role-name>Role 3</role-name>
  </auth-constraint>
  <user-data-constraint id="UserDataConstraint_1">
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint></p>
<p><security-role id="SecurityRole_1">
  <description>All Authenticated Users Role.</description>
  <role-name>Role 3</role-name>
</security-role></p>
<p><login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login.html</form-login-page>
    <form-error-page>/loginError.html</form-error-page>
  </form-login-config>
</login-config>
```

Optional steps for protecting the MobileFirst Operations Console

To protect the MobileFirst Operations Console with WebSphere Application Server authentication credentials, modify the `authenticationConfig.xml` file as follows:

1. Uncomment the element to enable protection of static resources:

```
<!-- Uncomment the next element to protect the worklight console and the first section in securityT
ests below. -->
<staticResources>
   <resource id="mobileFirstPlatformConsole" securityTest="MobileFirstPlatformConsole">
      <urlPatterns>/console*</urlPatterns>
   </resource>
</staticResources>
```

2. Add a element to your existing security test:

```
<securityTests>
   <customSecurityTest name="WorklightConsole">
      <test realm="WASLTPARealm" isInternalUserID="true"/
>
   </customSecurityTest>
</securityTests>
```

Creating client-side authentication components

The tutorial uses an existing MobileFirst application from one of the Authentication (../../authentication-security/) tutorials.
To implement security for an app, follow the same methods as for any other type of realm, and then configure the challenge handler to use your realm:

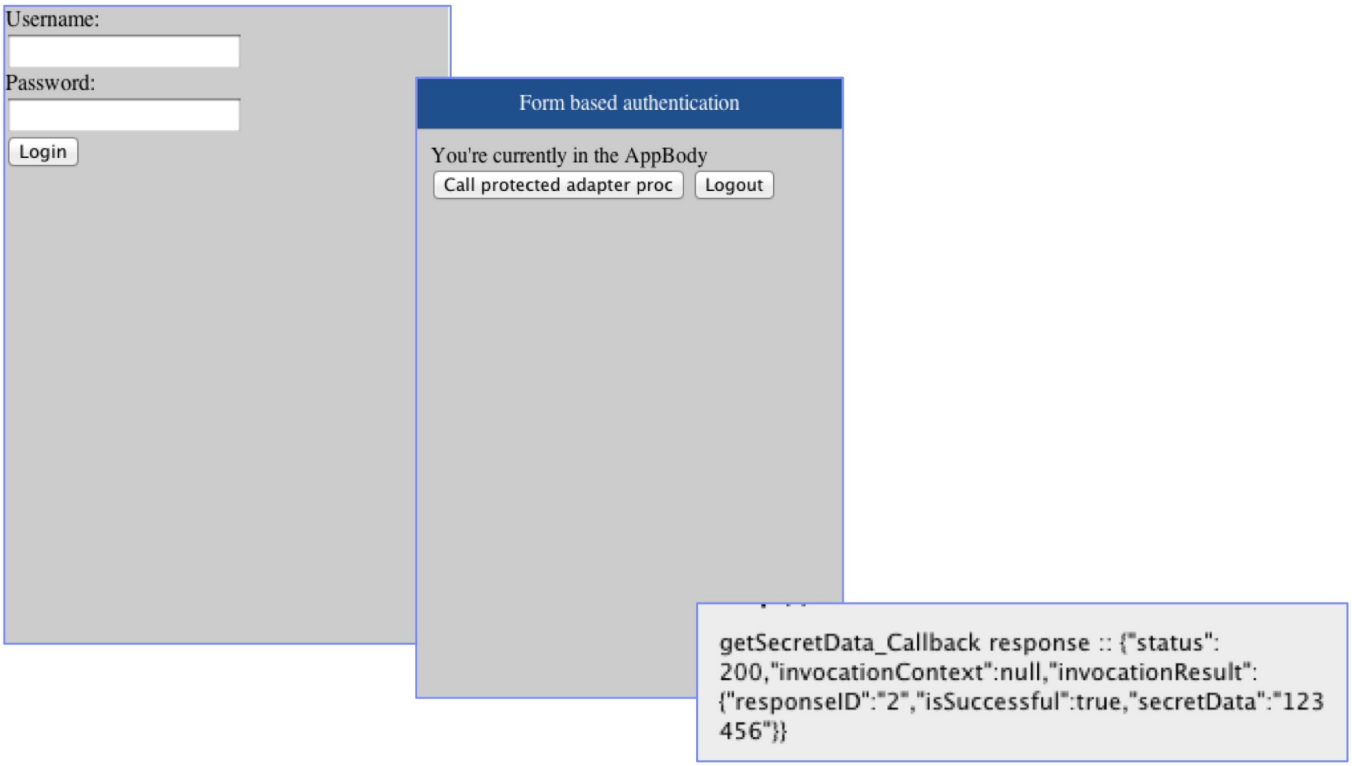
```
var sampleAppRealmChallengeHandler = WL.Client.createChallengeHandler("WASLTPARealm");<
```

1. In the `applicationDescriptor.xml` file, specify the security test that your app must use for the appropriate environments.
   For example:

   ```
   <common securityTest="WAS-securityTest"/>
   <android version="1.0" securityTest="WAS-securityTest">
      <pushSender key= "keyTest" senderId="senderIdTest"/
   >
   </android>
   ```

2. Deploy and test the application by using Option 2. The authentication process requires a valid user name and password from the underlying user registry against which the WebSphere Application Server is configured. When authentication is successful, the MobileFirst app is authenticated.

Examining the result


Username:
Password:
Login
Form based authentication
You're currently in the AppBody
Call protected adapter proc
Logout
getSecretData_Callback response :: {"status":
200,"invocationContext":null,"invocationResult":
{"responseID":"2","isSuccessful":true,"secretData":"123
456"}}