WebSphere LTPA-based authentication

Topics covered in this tutorial:

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

Introduction to WebSphere LTPA-based authentication

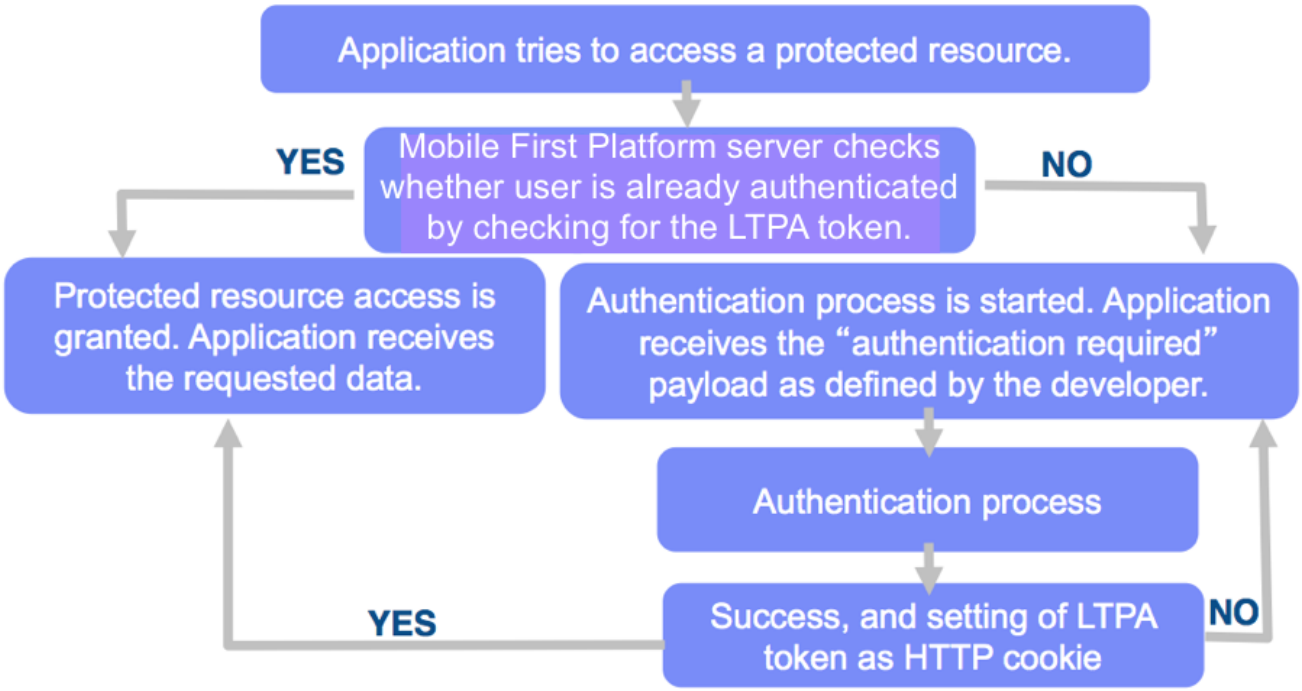
WebSphere Application Server uses a secure token in a Lightweight Third-Party Authentication (LTPA) cookie to verify authenticated users. WebSphere Application Server also uses this mechanism to trust users across a secure WebSphere Application Server domain.

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

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

Understanding the server-side authentication options

The diagram below illustrates the WebSphere LTPA based authentication process.



Option 1

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 Platform 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 username and password on initial login, 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

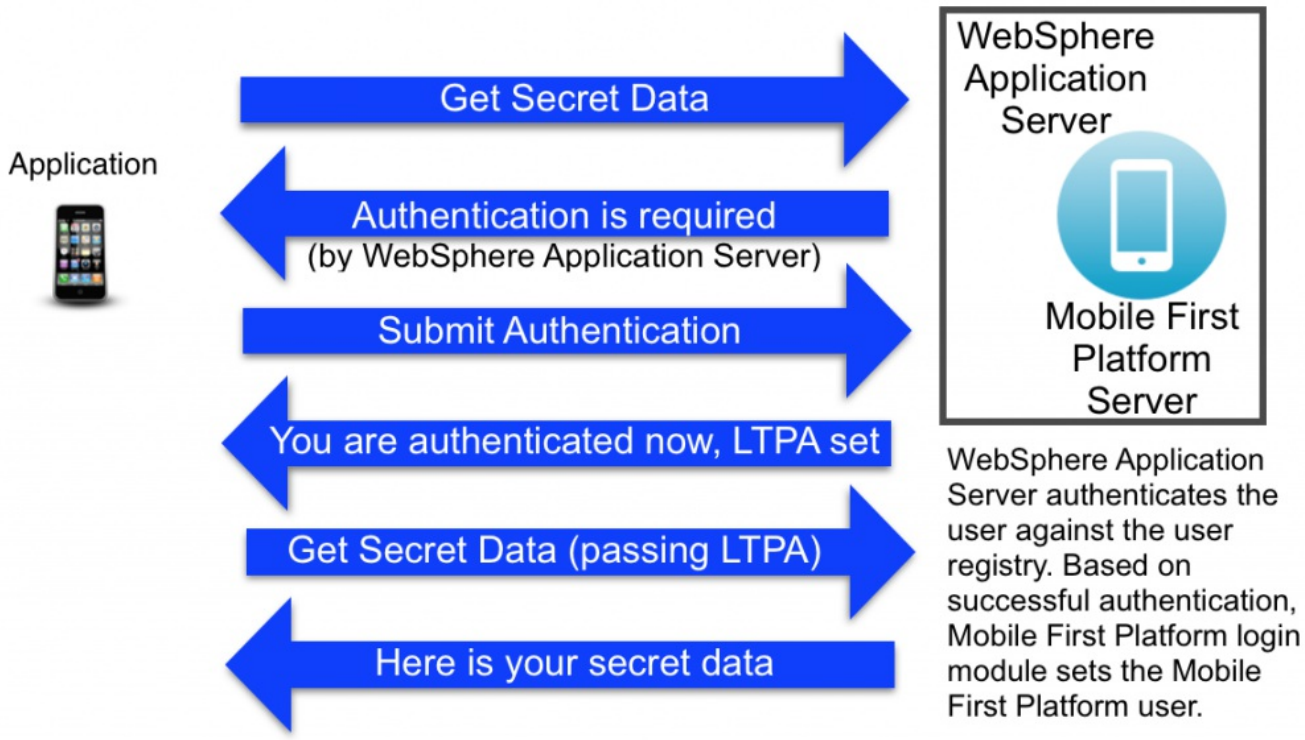
This option is for the MobileFirst Platform security configuration to handle user authentication at the MobileFirst Platform server level, by using the security configuration of the underlying WebSphere Application Server instances.

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

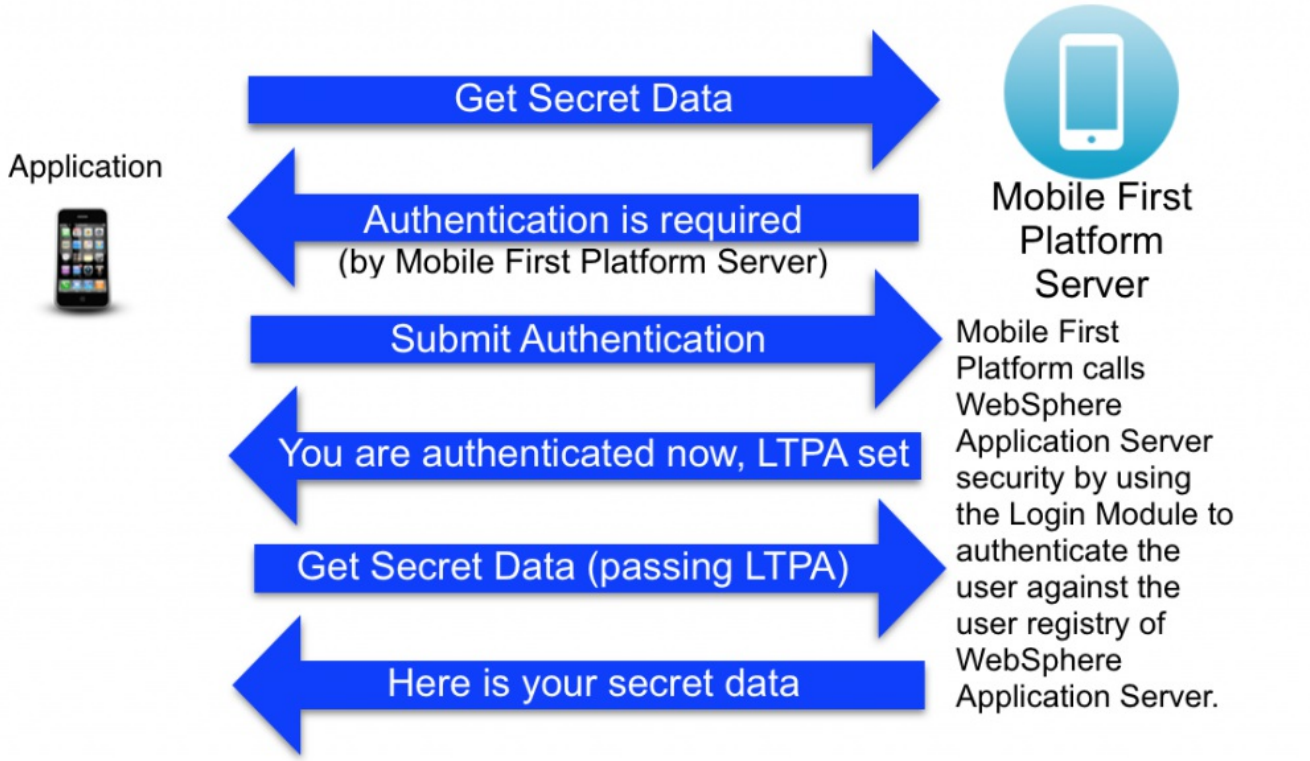
Option 1: Authentication is enforced by WebSphere Application Server

Option 2: MobileFirst Platform Server enforces the authentication by 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™ Enterprise Edition (Java EE) container by using SSO products from other software vendors.	This option uses the traditional WebSphere Application Server authentication and trust model without the impact of modifying the MobileFirst Platform project WAR file. The container enforces all security. Therefore, it can reuse existing investments in securing the Java™ 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 Platform runtime without being hindered 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 Platform 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 then Option 2 is best.

Configuring MobileFirst Platform Server for LTPA authentication

Configurations for WebSphere Application Server

Step 1: Enable WebSphere Application Server security (development environment is already set for Option 1)

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

For option 1:
Enable administrative sectuirty
Enable application security

For option 2:
Enable administrative security

Step 2: Configuring `authenticationConfig.xml`

Find the `authenticationConfig.xml` file in for your application, located in: `{WAS_HOME}/profiles/{your_profile}/installedApps/{your_node}/{MFP_EAR}/{MFP_WAR}/WEB-INF/classes/conf` (if you are in the development environment, this is located in `{MFP_Porject_Path}/server/conf`)

Uncomment the realm:

```
<!-- 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>
```

Optionally, you can include the parameters `cookie-domain`, `cookie-name`, and `httponly-cookie`. For more information, see the section about the LTPA authenticator in the product documentation.

In the same `authenticationConfig.xml`, uncomment the login module:

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

Note: The login module might already be uncommented.

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

- Add webSecurityTest if you plan to develop for web environments
- Add mobileSecurityTest if you plan to develop for mobile environments

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

Step 3: Creating login.html and loginError.html

Create a file that is named `login.html` or use the `login.html` that MFP provides you in the `{MFP_Project_Path}/server` directory. Save the login.html to the root of your WAR file: `{WAS_HOME}/profiles/{your profile}/installedApps/{your node}/{MobileFirst Platform WAR}`, if you are working in the development server this is the WAR file located in `{MFP_Project_Path}/bin`. Unzip that WAR and copy paste the login.html, rezip the project and change the .zip to .war.

If you decide to create your own `login.html` make sure the form contents are set like the following form:

```
<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>
```

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 Platform WAR}`. The `loginError.html` page is used when login fails.

Set its contents as follows:

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

Step 4: Configuring web.xml - This step is optional for option 2, but mandatory for option 1.

Locate the `web.xml` file: `{WAS_HOME}/profiles/{your profile}/installedApps/{your node}/{MobileFirst Platform EAR}/{MobileFirst Platform WAR}/WEB-INF/web.xml`

Inside the root tag, add the tags as shown in the code sample below. These tags pass to WebSphere Application Server the configuration that the WAR file expects.
Note: Tailor this code block to fit your needs. This is just one example of setting the configuration.

```
<security-constraint id="SecurityConstraint_1">
  <web-resource-collection id="WebResourceCollection_1">
    <web-resource-name>Servlet</web-resource-name>
    <description>Protection area for 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>Servlet Security:+:All Authenticated users for Servlet.</description
>
    <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 Platform Console

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

- Uncommment the element to enable protection of static resources:

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

- Add a element to your existing security test:

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

Creating client-side authentication components

Using an existing MobileFirst Platform application from one of the Authentication modules.
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:

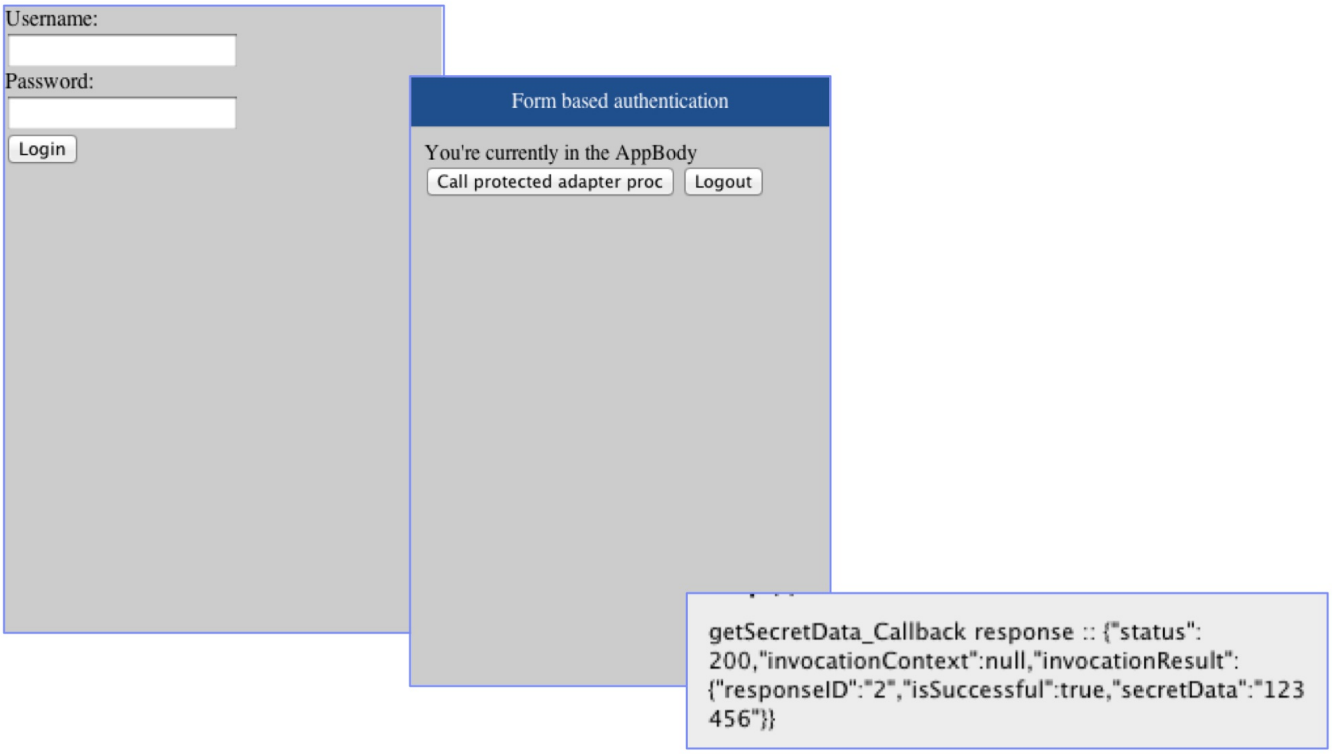
```
var sampleAppRealmChallengeHandler = WL.Client.createChallengeHandler("WASLTPARealm");<br /
>
```

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

```
<common securityTest="wasWebSecurity"/>
<android version="1.0" securityTest="WAS-securityTest">
  <pushSender key= "keyTest" senderId="senderIdTest"/>
</android>
<iphone bundleId="{IPhone_bundle_Id}" applicationId="{IPhone_application_Id}" version="1.0" security
Test="WAS-securityTest">
  <worklightSettings include="false"/>
  <security>
    <encryptWebResources enabled="false"/>
    <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, m
p3"/>
  </security>
</iphone>
```

Deploy and test the application by using Option 2. The authentication requires a valid user name and password from the underlying user registry that the WebSphere Application Server is configured against. When the authentication is successful, the MobileFirst Platform 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"}}