Classic security model

This tutorial gives an overview of IBM MobileFirst Platform Foundation authentication concepts in the following topics:

- Authentication concepts and entities
- Defining realms, authenticators, and login modules
- Defining security tests
- Protecting applications
- Protecting adapters
- Protecting static resources

Authentication concepts and entities

MobileFirst entities such as applications, adapter procedures, and static resources can be protected from unauthorized access.
Rules are defined by a **security test** that contains one or more **authentication realms**.

An **authentication realm** defines the process to be used to authenticate users.

Each authentication realm consists of an **Authenticator** and a **Login module**, which are server-side components.

The same authentication realm can be used to protect several resources.

Each authentication realm requires a **challenge handler** component on the client-side.

Authenticator

An authenticator is a server-side entity that is responsible for collecting the credentials from the client application.

An authenticator can collect any type of information that is accessible from an HTTP request object: cookies, headers, body, or any other properties.

IBM MobileFirst Platform Foundation Server comes with a set of predefined authenticators, including:

- A **form-based** authenticator that returns a challenge in the form of an HTML login form, making it useful for web environments and mobile applications.
- An **adapter-based** authenticator that uses the MobileFirst adapter procedure to collect and validate the credentials from the client application.
- A **header-based** authenticator that does not require interactive credentials collection, but checks the specific HTTP header instead.

In addition to predefined authenticators, you can create your own custom authenticator by using Java code.

Login module

A login module is a server-side entity that is responsible for verifying the user credentials and for creating a *user identity* object, which holds the user properties for the remainder of the session.

Credentials can be validated in one of the following ways:

- By using a web service.
- By looking up the user in a users table in a database.
- By using the WebSphere LTPA token.

It is possible to add custom user properties according to the enterprise needs.

A login module destroys the user identity object when the authenticated session terminates (logout or timeout).

A login module can be configured to automatically record login attempts for audit purposes.

In addition to predefined login modules, you can create your own custom login module by using the Java code.

Authentication realm

An authentication realm is a combination of one authenticator and one login module. Each authentication realm defines its authentication flow:

- What should happen after the authentication process is triggered?
- What form of challenge should be sent to the client application?
- Which credentials should be collected?
- How and when should credentials be collected?
- How should credentials be sent to the server?
- How should credentials be validated by the server?
- What will be the result of credentials validation?
- What will be the properties of the user identity object?

Each authentication realm that is defined in the server authentication configuration must have a corresponding challenge handler in the client application. IBM MobileFirst Platform Foundation provides several predefined authentication realms for security features, such as remote application disablement or application authenticity.

Security test

A security test is an ordered set of authentication realms that is used to protect a resource such as an adapter procedure, an application, or a static URL.

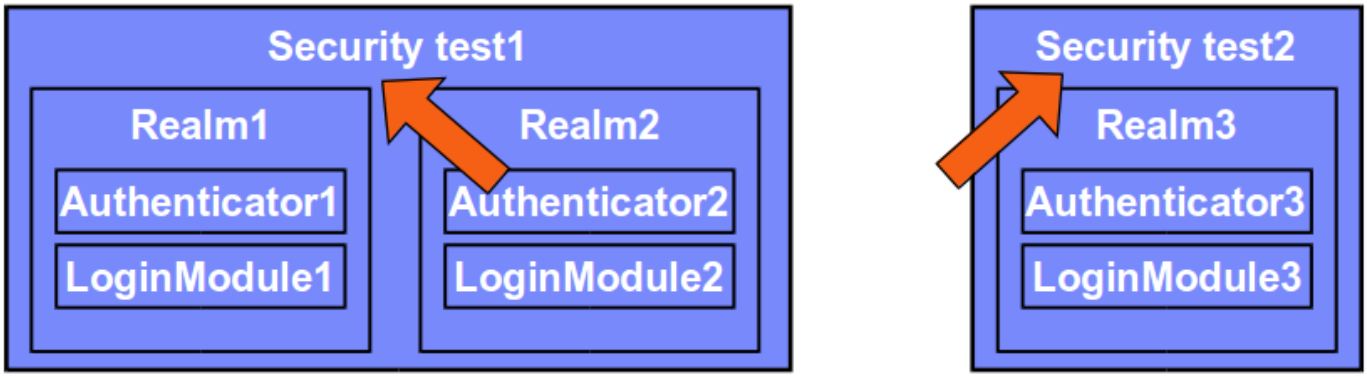
A security test defines the realms that the user must authenticate against to get access to the protected resource.

A developer can define the order in which authentication is processed, for example to request authentication in realm2 only after realm1 authentication succeeds.

The MobileFirst framework provides definitions of default security tests for mobile and web environments, and the ability to create custom security tests.

(http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_sample-security-config.png)

(http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_sample-security-config.png)
(http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_sample-security-config.png)

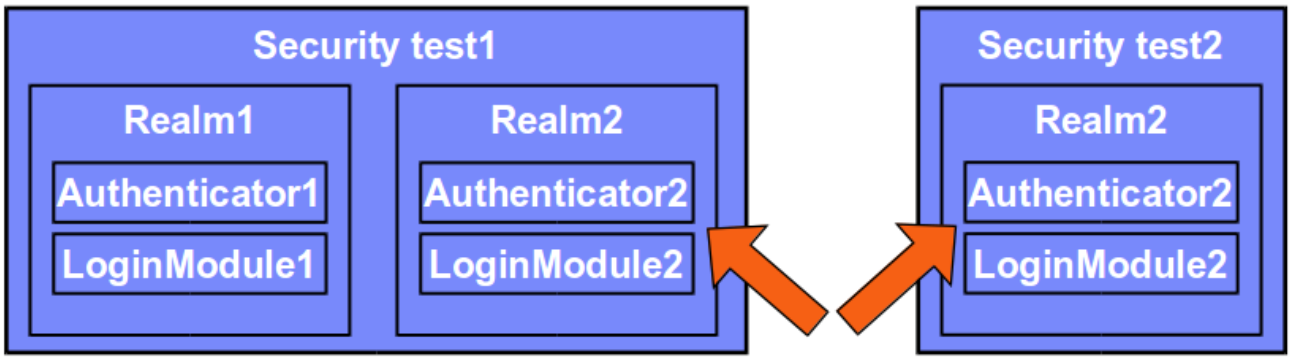


In the example above, a resource - for example, an application or adapter procedure - can be protected by either of two security tests.

- Using **Security test 1** means that the user must authenticate in both **Realm1** and **Realm2**, each one with its own set of rules.
- Using **Security test 2** means that the user must authenticate in **Realm3 only**.

  Each realm defines its own set of **Authenticator** and **Login module**, meaning that each realm has its own rules for collecting credentials and validating them.
  (http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_sample-security-config-reuse.png)

  (http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_sample-security-config-reuse.png)
  (http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_sample-security-config-reuse.png)

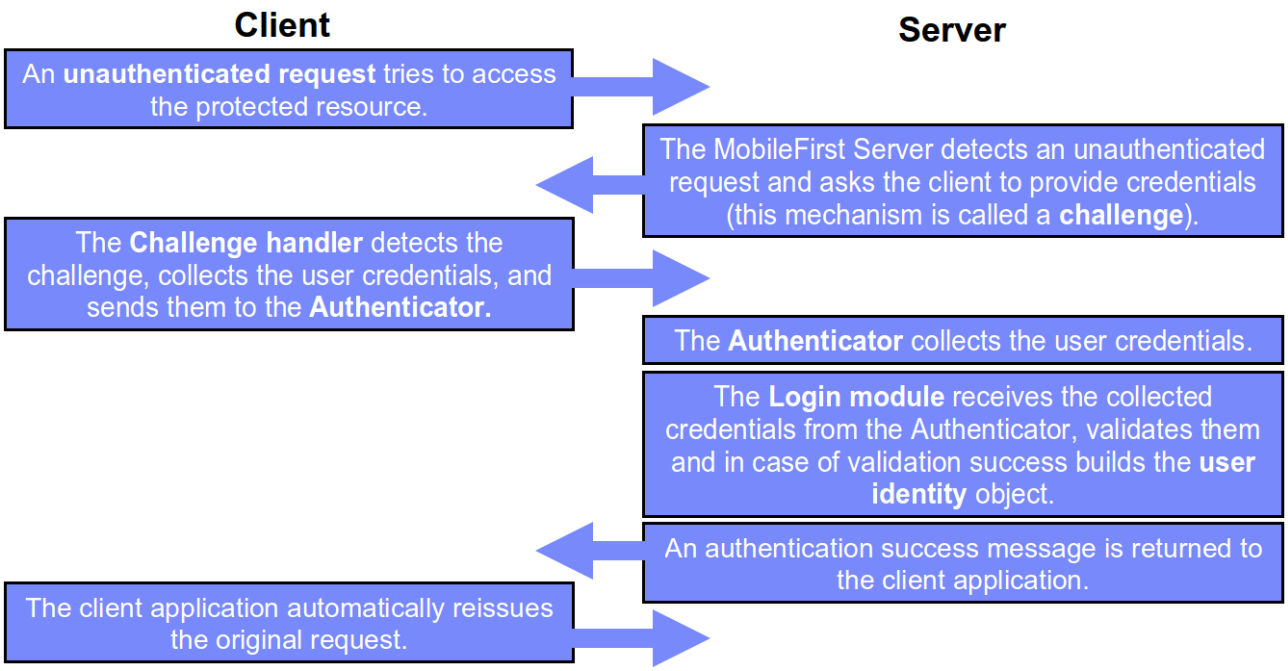


Realms, authenticators, and login modules can be reused.
On the updated configuration above, **Realm2** is reused.
Protecting a resource with **Security test1** means that the user must authenticate in both **Realm1 and Realm2**.
Protecting a resource with **Security test2** means the user must authenticate in **Realm2 only**.

When a request is made to the protected entity, IBM MobileFirst Platform Foundation checks whether the session is already authenticated. If it is not, a process is automatically triggered to verify the user's identity.
(http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_client-server-flow.png)

(http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_client-server-flow.png)

(http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_client-server-flow.png)



Challenge handler

A challenge handler is a client-side entity that controls the authentication process. It is a piece of code (JavaScript, Objective-C, Java, etc.) that you write in your client-side project to handle a specific challenge. It is used to detect the authentication challenges in the server responses and handle them.

Create a separate challenge handler instance for each realm that the application must authenticate in.

A challenge handler can be used to detect and handle both MobileFirst-related and external authentication challenges, like authentication proxies and gateways. After a challenge handler detects an authentication challenge that is returned from the server, it is responsible for collecting the required credentials and for sending them back to the server.

After the authentication flow completes, the challenge handler can send a notification back to the MobileFirst framework about the authentication success or failure.

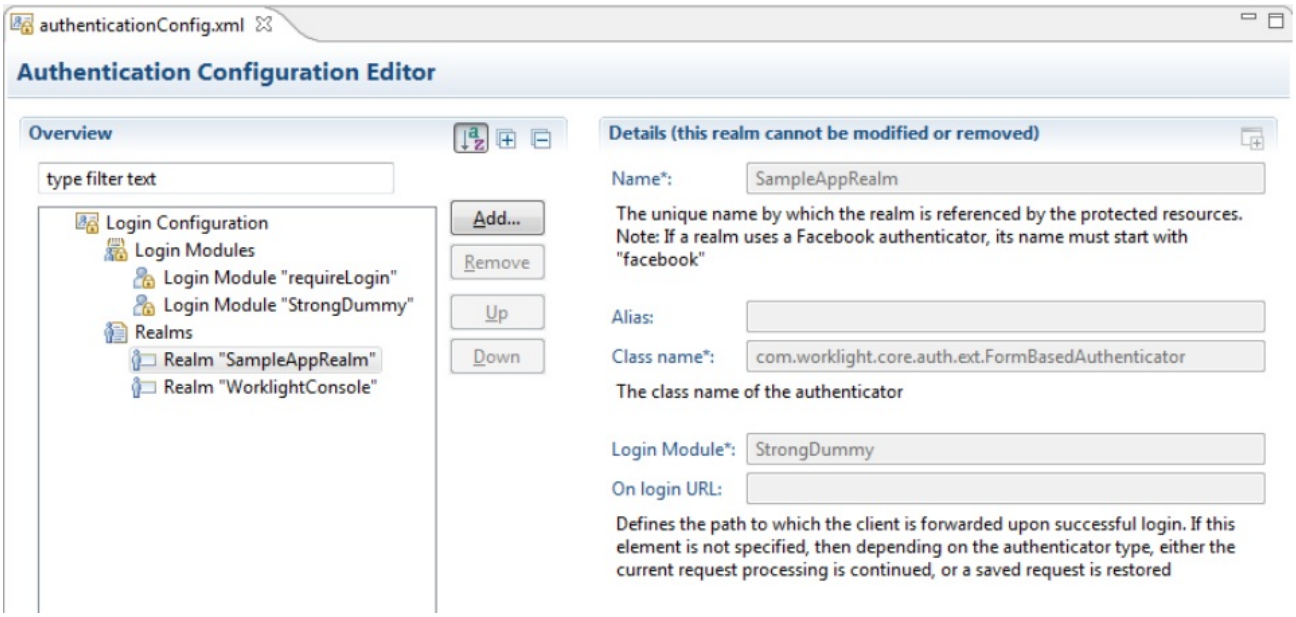
Though customizable, a challenge handler is created with a preset of methods that you can use to submit the credentials to the built-in user authentication types of the IBM MobileFirst Platform Foundation Server. In your challenge handler, do not add code that modifies the user interface when this modification is not related to the authentication flow.

Defining realms, authenticators, and login modules

Authentication settings are configured in the MobileFirst project in the `server\conf\authenticationConfig.xml` file.

You can modify them by using the Authentication Configuration Editor. (http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_authentication_config_editor.png)

(http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_authentication_config_editor.png)
(http://developer.ibm.com/worklight/wp-content/uploads/sites/32/2014/07/09_01_authentication_config_editor.png)



Each realm has a name, a **loginModule** specification, a `**className**` of an authenticator implementation and optional parameters.

```
<realms>
  <realm name="SampleAppRealm" loginModule="StrongDummy">
    <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className
>
  </realm>
  <realm name="SubscribeServlet" loginModule="rejectAll">
    <className>com.worklight.core.auth.ext.HeaderAuthenticator</className>
  </realm>
</realms>
```

Each login module has a name, a `**className**` of the implementation and optional parameters.

```
<loginModules>
  <loginModule name="StrongDummy">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className
>
  </loginModule>
  <loginModule name="requireLogin">
    <className>com.worklight.core.auth.ext.SingleIdentityLoginModule</className>
  </loginModule>
  <loginModule name="rejectAll">
    <className>com.worklight.core.auth.ext.RejectingLoginModule</className>
  </loginModule>
</loginModules>
```

Defining security tests

With IBM MobileFirst Platform Foundation, you can set up multiple realms for a security test.

As part of the security test setup, you must set which realms are considered a **user realm** and which are considered a **device realm**.

Any identity from a realm that is defined as a **user realm** is used as a *user identity* for features that require one, such as push notification or application usage reports.
An identity from a realm that is defined as a **device realm** is used as a *device identity* for features that require one, such as device provisioning, push notification, and SMS notification.

After you set up your authentication realms, you must define the security tests to be used to protect your applications, adapter procedures, and static resources.
Three types of security tests can be defined in the `authenticationConfig.xml` file:

- `webSecurityTest` – A test that enables default web security-related realms.
- `mobileSecurityTest` – A test that enables default mobile security-related realms.
- `customSecurityTest` – A custom security test. Does not contain any default realm.

webSecurityTest

Use the **webSecurityTest** to protect web applications.
By default, the **webSecurityTest** includes a protection against XSRF attacks. For more information about this protection, see the user documentation.
Each **webSecurityTest** must contain one **testUser** element with a realm definition.
This realm is considered a **user realm**.

```
<webSecurityTest name="SampleWebSecurityTest">
  <testUser realm="SampleRealm" />
</webSecurityTest>
```

mobileSecurityTest

Use the **mobileSecurityTest** to protect mobile applications.
By default, the **mobileSecurityTest** includes:

- A protection against XSRF attacks
- An application authenticity test. For more information, see the user documentation.
- An capability to disable mobile applications remotely from the MobileFirst Operations Console.

Each **mobileSecurityTest** must contain one **testUser** element with realm definition.
This realm is considered a **user realm**.

```
<mobileSecurityTest name="SampleMobileSecurityTest">
  <testUser realm="SampleRealm" />
</mobileSecurityTest>
```

customSecurityTest

Use the **customSecurityTest** to dictate your own security preferences.
Unlike the mobile and web security tests, the **customSecurityTest** does not include any predefined authentication realms. It includes only the tests that are defined by the developer.
Any number of tests can be defined within the **customSecurityTest**.
You can define which realm to be used as a user realm by adding the `isInternalUserId="true"` property.
You can define the order of realms that the user must authenticate in.

```
<customSecurityTest name="SampleCustomSecurityTest">
   <test realm="SampleRealm1" step="1"/>
   <test realm="SampleRealm2" step="2"/>
   <test realm="SampleRealm3" step="3" isInternalUserID="true"/
>
</webSecurityTest>
```

Protecting applications

Protecting an application means that authentication is required immediately when the application tries to connect to the MobileFirst Server instance.

A separate **securityTest** can be defined for each application environment in the `application-descriptor.xml` file.

```
<android version="1.0" securityTest="SampleMobileSecurityTest">
  ...
</android>
```

If no security test is defined for a specific environment, only a minimal set of default platform tests are carried out.

Protecting adapters

Protecting an adapter procedure means that authentication is required when this adapter procedure is called by a client application.
A separate **securityTest** can be defined for each adapter procedure in the adapter XML file.

```
<wl:adapter name="DummyAdapter"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:wl="http://www.worklight.com/integration"
   xmlns:http="http://www.worklight.com/integration/http"><
   <displayName>DummyAdapter</displayName>
   <description>DummyAdapter</description>
   <connectivity>
      <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
         <protocol>http</protocol>
         <domain>rss.cnn.com</domain>
         <port>80</port>
      </connectionPolicy>
      <loadConstraints maxConcurrentConnectionsPerNode="2" />
   </connectivity>
   <procedure name="getSecretData" securityTest="DummyAdapter-securityTest"/
>
</wl:adapter>
```

Protecting static resources

A static resource is a URL that is loaded from the MobileFirst Server instance, for example: the console or mobile web application.

Protecting a static resource means that MobileFirst Server requires authentication at any attempt to browse to the specified URL.
The static resources and their protection can be defined in the `authenticationConfig.xml` file.

```
<staticResources>
  <resource id="mobileFirstConsole" securityTest="SubscribeServlet"
>
    <urlPatterns>/console*</urlPatterns>
  </resource>
</staticResources>
```