OAuth-based security model

Overview

The OAuth 2.0 (http://oauth.net/) protocol is based on the acquisition of an access token, which encapsulates the authorization that is granted to the client. In that context, IBM MobileFirst Platform Server serves as an authorization server and is able to generate such tokens. The client can then use these tokens to access resources on a resource server, which can be either MobileFirst Server itself or an external server. The resource server checks the validity of the token to make sure that the client can be granted access to the requested resource. This separation between resource server and authorization server in the new OAuth-based model allows you to enforce MobileFirst security on resources that are running outside MobileFirst Server.

To support backward compatibility, the classic (pre-V7.0) MobileFirst security model (../classic-security-model/) is still used in the flows that are based on the existing MobileFirst APIs (for example, `invokeProcedure` in Java). The new client APIs trigger flows that conform to the OAuth-based security model. IBM MobileFirst Platform Foundation V7.0 provides seamless integration between the two security models. The platform allows you to mix classic and new APIs in the same application, while keeping a consistent security context on the server side.

This tutorial covers the following topics:

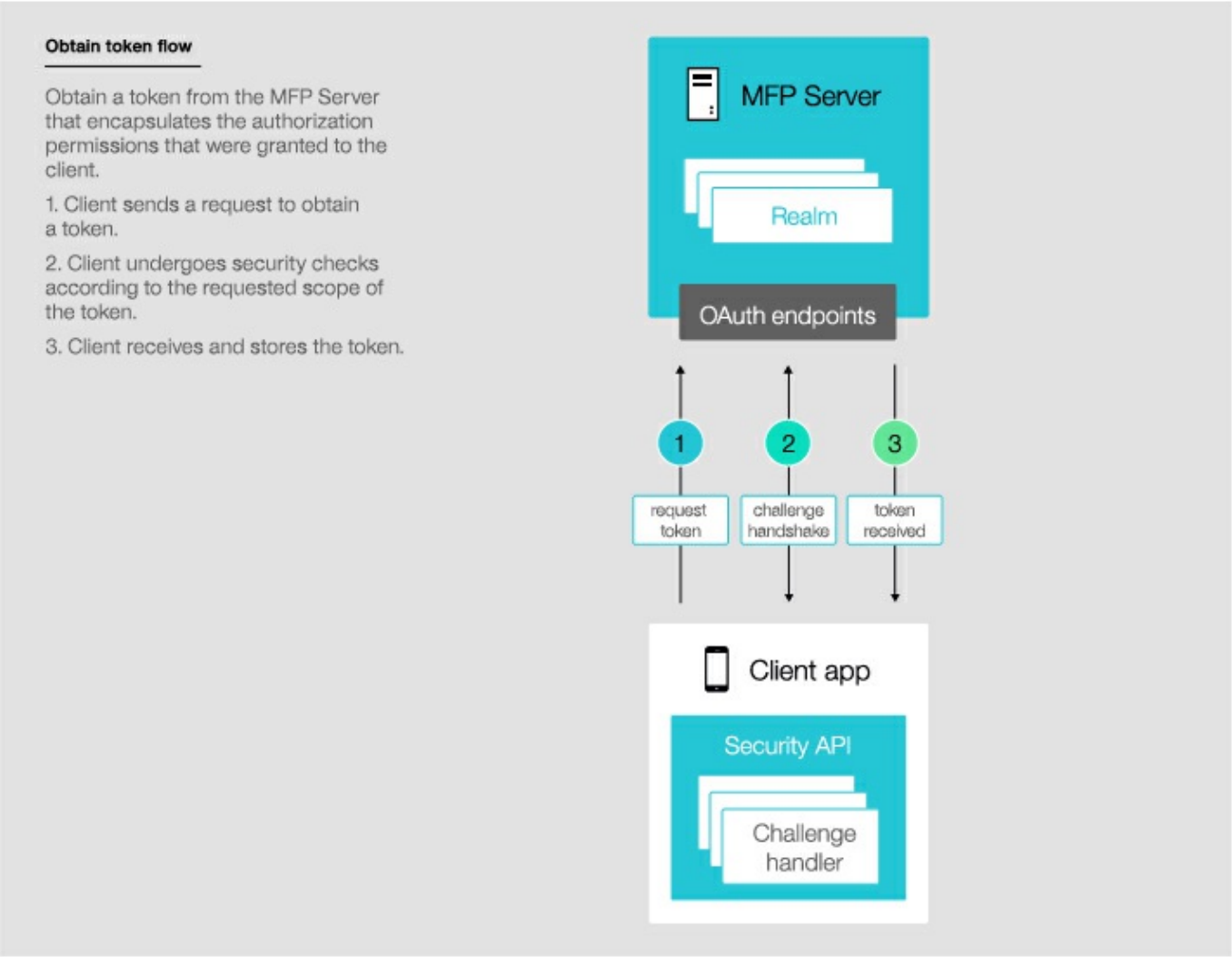
- Authorization flow
- Authorization entities
- Defining realms, authenticators, and login modules
- Protecting adapters
- Protecting external resources
- Protecting applications and static resources

Authorization flow

The new MobileFirst end-to-end authorization flow has two phases: the client acquires the token and then uses it to access a protected resource.

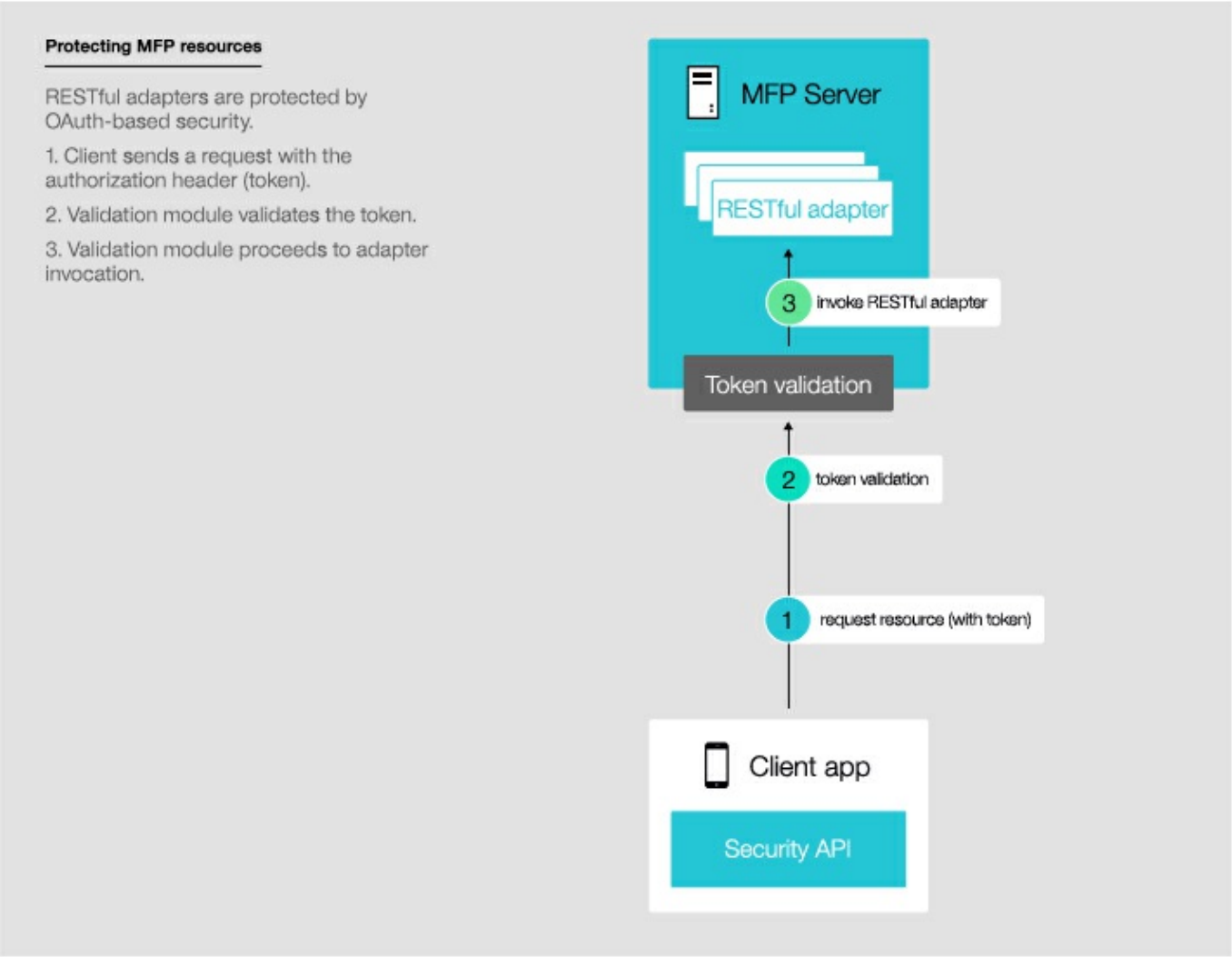
Acquiring a token

In this phase, the client undergoes security checks in order to receive an access token. These security checks use authorization entities, which are described in the next section.



Using a token to access a protected resource

It is possible to enforce MobileFirst security both on resources that run on MobileFirst Server, as shown in this diagram, and on resources that run on any external resource server as explained in tutorial Using MobileFirst Server to authenticate external resources (../../../authentication-security/using-mobilefirst-server-authenticate-external-resources/).



Authorization entities

You can protect MobileFirst resources such as adapter procedures from unauthorized access by specifying a **scope** that contains one or more *authentication realms*.

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

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

The same realm can be used to protect several resources.

Each 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.

MobileFirst 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 infrastructure 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 authenticators by using Java code, as explained in tutorial Custom Authenticator and Login Module (../../../authentication-security/custom-authenticator-login-module/).

Login Module

A login module is a server-side entity that is responsible for verifying the user credentials provided by the authenticator.

The credentials validation can be done, for example, 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 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, , as explained in tutorial Custom Authenticator and Login Module (../../../authentication-security/custom-authenticator-login-module/).

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?

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.

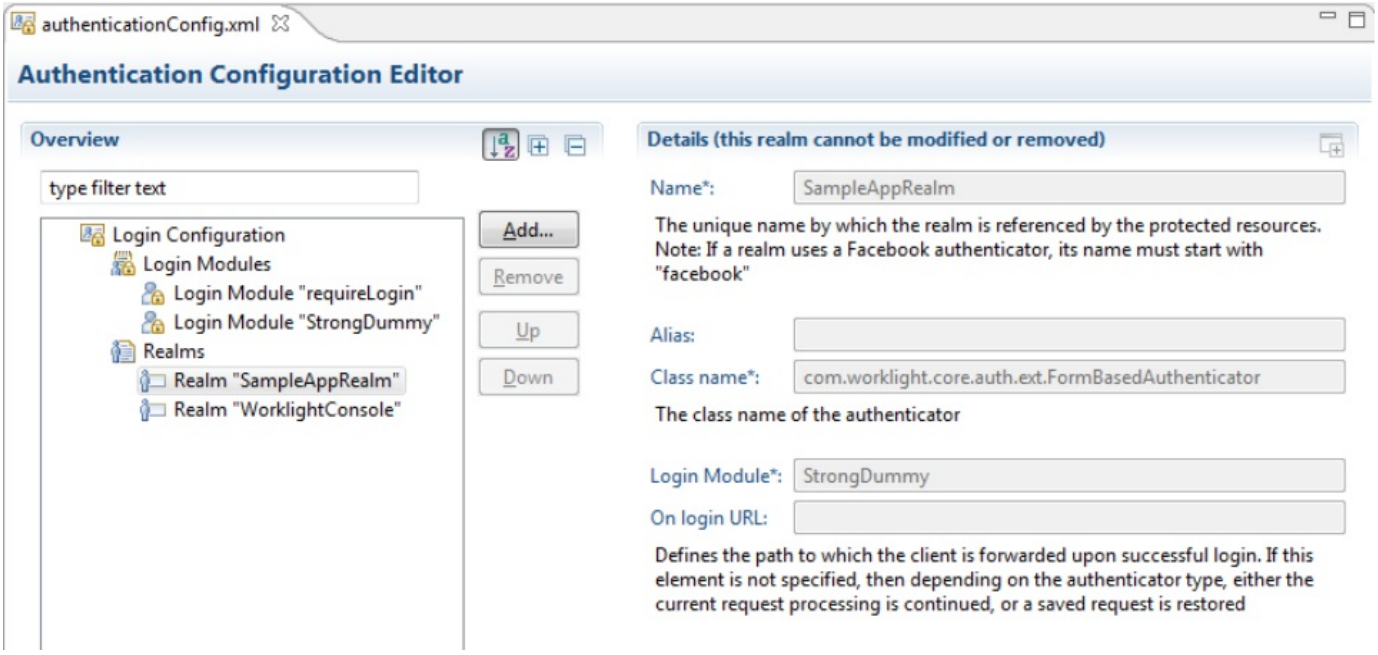
Scope

To protect a resource, you must define a scope.
A scope is a list of realm names - the realms that the user must authenticate against to get access to the protected resource.

Defining realms, authenticators, and login modules

You configure authentication settings in the MobileFirst project in the `server\conf\authenticationConfig.xml` file.

You can modify the settings by using the Authentication Configuration Editor.



Each realm takes a name, a `loginModule` specification, an authenticator implementation, and optional parameters.

```
<realms><br />
   <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 takes a name, a login module implementation, and optional parameters.

```
 <br />
<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>
```

User Identity

Although user authorization can include authorization with several realms, only one of these realms is defined as the *user identity realm*, and that realm determines the user identity. In an OAuth-based flow, the

user identity realm is set according to the `userIdentityRealms` definition in the **application descriptor**. This is the user identity that will appear in the Devices tab of the console.

Note: In the classic (pre-V7.0) flows, the user identity realm is selected according to the definition in the security test. In OAuth-based flows, information about the user identity, which is set by the `userIdentityRealms` attribute, is part of the data contained in the ID token.

```
<userIdentityRealms>SampleAppRealm</userIdentityRealms>
```

Protecting adapters

Java adapters

MobileFirst Java adapters come ready with OAuth capabilities. See Java Adapters (../../../server-side-development/java-adapter/).

```
    @DELETE
    @Path("/{userId}")
    @OAuthSecurity(scope="adminRealm")
    //This will serve: DELETE /users/{userId}
    public void deleteUser(@PathParam("userId") String userId)
{
        ...
    }
```

In this example, the `deleteUser` procedure uses the annotation `@OAuthSecurity(scope="adminRealm")`, which means that it is protected by a scope containing the realm `adminRealm`.

A scope can be made of several realms, space-separated: `@OAuthSecurity(scope="realm1 realm2 realm3")`.

If you do not specify the `@OAuthSecurity` annotation, the procedure is protected by the MobileFirst default security scope. That means that only a registered mobile app that is deployed on the same MobileFirst Server instance as the adapter can access this resource. Any security test protecting the application also applies here.

If you want to disable MobileFirst default security, you can use: `@OAuthSecurity(enabled=false)`.

You can use the `@OAuthSecurity` annotation also at the resource class level, to define a scope for the entire Java class.

JavaScript adapters

JavaScript adapters such as HTTP adapters, SQL adapters, etc, need to be protected by a *security test*. Security tests are covered in the Classic security model (../classic-security-model/) tutorial.

Remember that adapters protected via the previous "classic" model can still be accessed via REST APIs and respond to OAuth authentication requests.

Protecting external resources

With IBM MobileFirst Platform Foundation V7.0, you can enforce MobileFirst security on resources that run

outside MobileFirst Server. To this end, you can use the built-in `Node.js` and Java validation modules, or implement your own custom validation module in the technology of your choice, by using the online validation endpoint.

See Using MobileFirst Server to authenticate external resources (../../../authentication-security/using-mobilefirst-server-authenticate-external-resources/).

Protecting applications and static resources

Additionally, entire applications are protected by a security test. Security tests are covered in the Classic security model (../classic-security-model/) tutorial.