Authorization concepts

Overview

The MobileFirst Platform Foundation authentication framework uses the OAuth 2.0 (http://oauth.net/) protocol. The OAuth 2 protocol is based on the acquisition of an access token that encapsulates the granted permissions to the client.

In that context, IBM MobileFirst Platform Server serves as an **authorization server** and is able to **generate access tokens**. The client can then use these tokens to access resources on a resource server, which can be either the 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. The separation between resource server and authorization server allows to enforce security on resources that are running outside MobileFirst Server.

Jump to:

- Authorization flow
- Authorization entities
- Protecting resources
- Tutorials to follow next

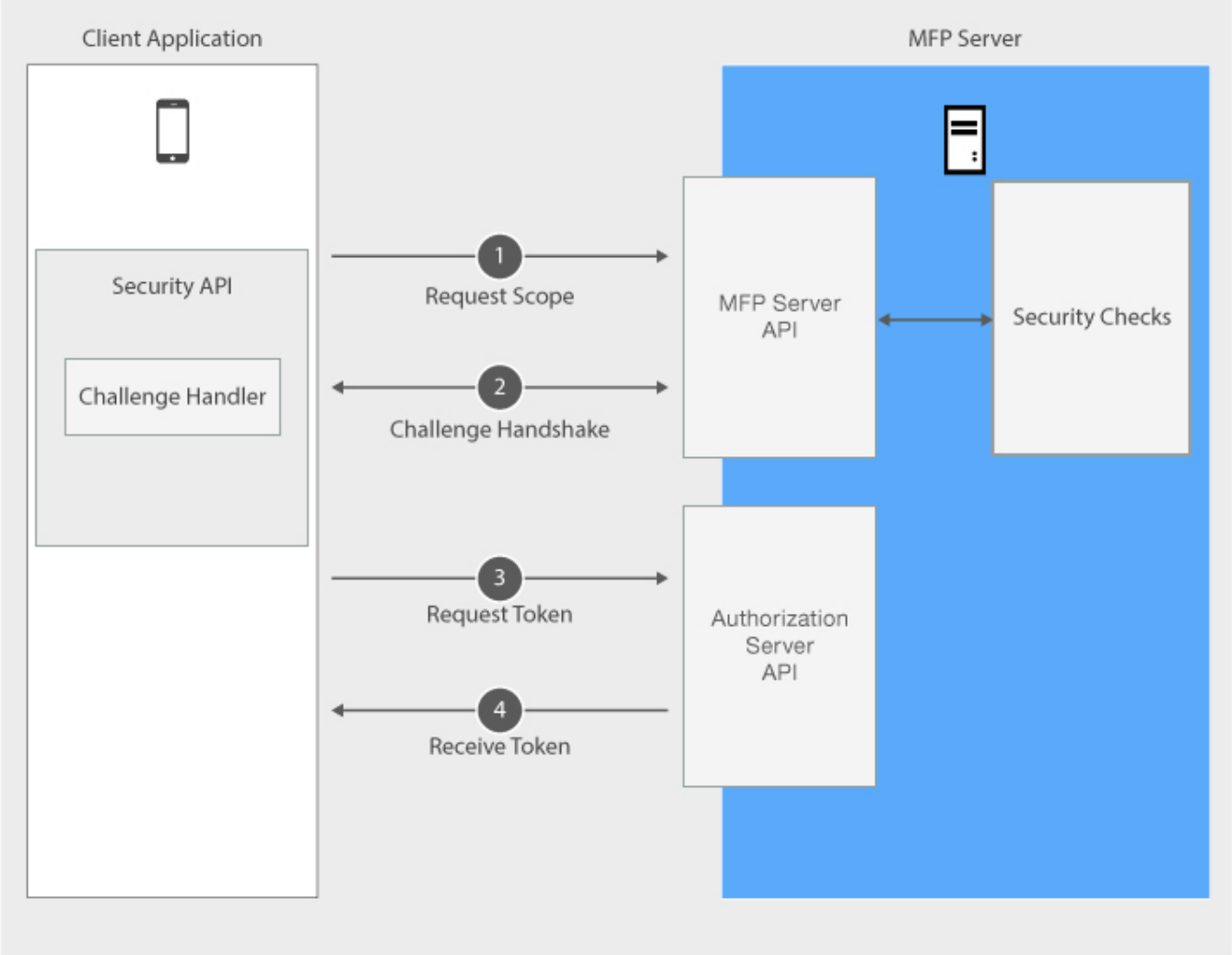
Authorization flow

The authorization flow has two phases:

1. The client acquires a token.
2. The client uses the token to access a protected resource.

Obtaining a token

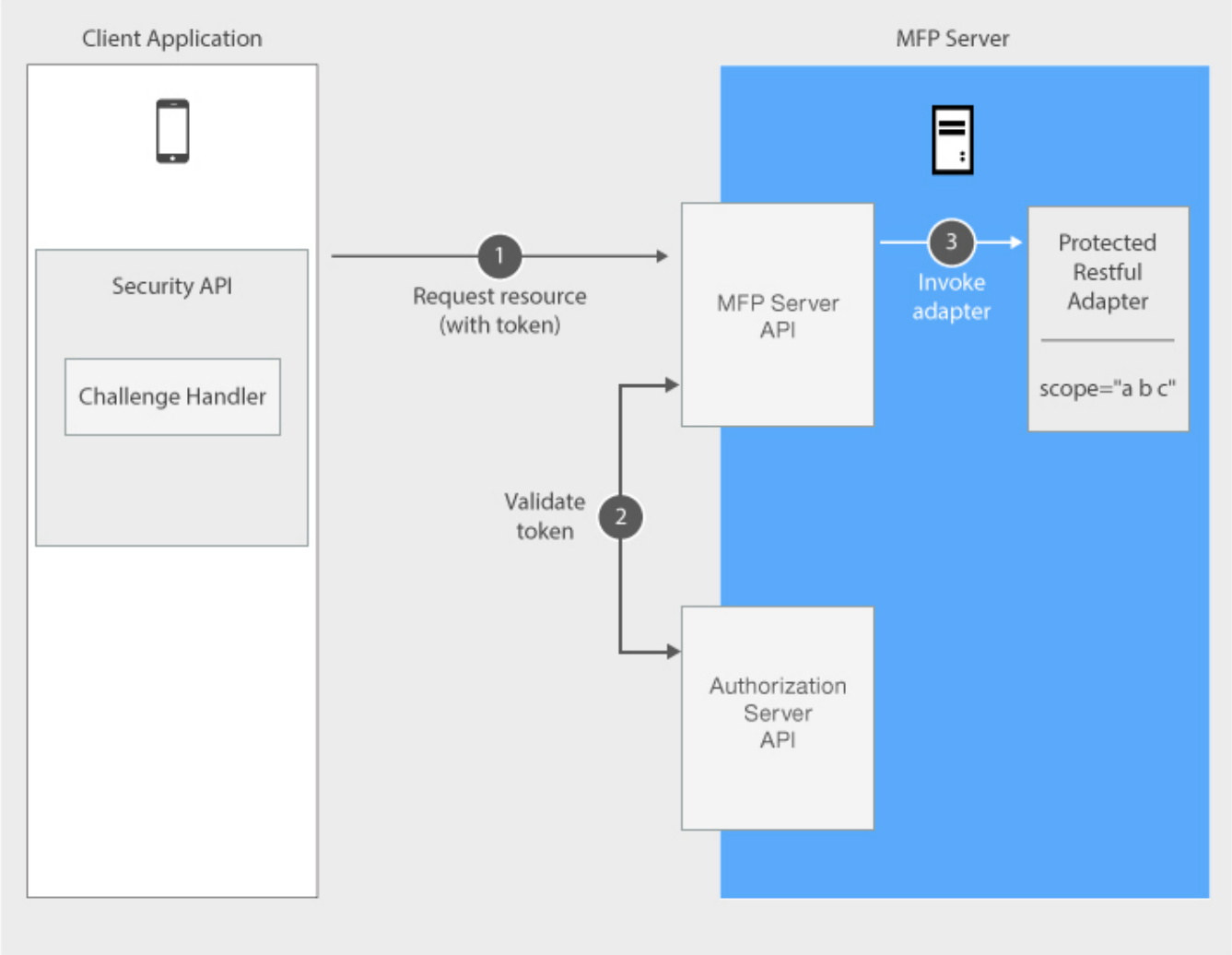
In this phase, the client undergoes **security checks** in order to receive an access token.
These security checks use **authorization entities**, which are described in the next section.



1. Client application sends a request to use a protected resource.
2. Client application undergoes security checks according to the requested scope.
3. Client application requests a token from the Authorization Server.
4. Client application receives the token.

Using a token to access a protected resource

It is possible to enforce 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 (../../using-mobilefirst-server-authenticate-external-resources/).



1. Client application sends a request with the received token.
2. Validation module validates the token.
3. MobileFirst Server proceeds to adapter invocation.

Authorization entities

Several authorization entities are available as part of the MobileFirst authentication framework:

Scope

You can protect resources such as adapters from unauthorized access by specifying a **scope**.
A scope is a space-separated list of zero or more **scope elements**.

Scope Element

A scope element is a keyword that indicates which **security checks** are being used to protect the resource.

Scope Mapping

By default, the scope elements you write in your scope are matched to a security check with the same name.

Optionally, at the application level, you can map a scope element to a different security check.
You can also map it to a list of zero or more security checks. This can be useful if you want to protect a resource differently depending on which application is trying to access it.

Security Check

A security check is an object responsible for obtaining credentials from a client and validate them. Security checks are instantiated by Adapters.

The security check defines the process to be used to authenticate users. It is often associated with a **SecurityCheckConfiguration** that defines properties to be used by the security check.
The same security check can also be used to protect several resources.

On the client-side, the application logic needs to implement a **challenge handler** to handle challenges sent by the security check.

Built-in Security Checks

Several predefined security checks available:

- Application Authenticity (../application-authenticity/)
- Direct Update (../../using-the-mfpf-sdk/direct-update)
- LTPA

> Learn more about security checks in the Creating a Security Check (../creating-a-security-check/) tutorial.

Protecting resources

Your resources can be protected by one of several ways:

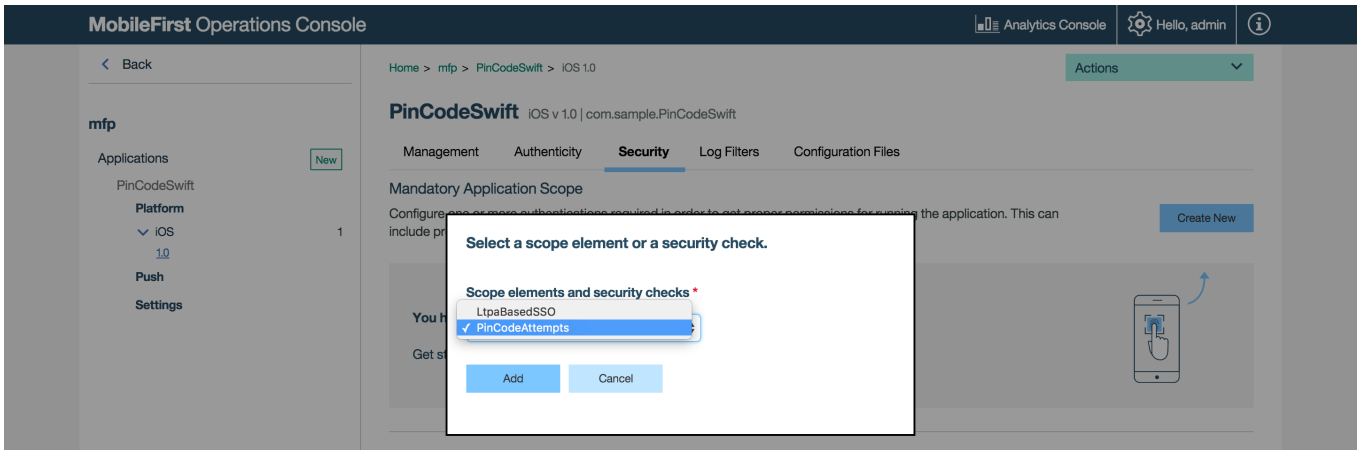
Default scope

By default, all resources are protected by a default scope that restricts access to registered mobile applications only.

Mandatory application scope

At the application level, you can define a scope that will apply to all the resources used by this application.

You can define the mandatory application scope from the MobileFirst Operations Console:



Resource-level

Java adapters

You can specify the scope of a resource method by using the `@OAuthSecurity` annotation.

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

In the above example, the `deleteUser` method uses the annotation `@OAuthSecurity(scope="deletePrivilege")`, which means that it is protected by a scope containing the scope element `deletePrivilege`.

A scope can be made of several scope elements, space-separated: `@OAuthSecurity(scope="element1 element2 element3")`.

If you do not specify the `@OAuthSecurity` annotation, the method is protected by the MobileFirst default security scope.

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

JavaScript adapters

You can protect a JavaScript adapter procedure by assigning a scope to the procedure definition in the adapter's XML file:

```
<procedure name="deleteUser" scope="deletePrivilege">
```

A scope can be made of several scope elements, space-separated:

```
<procedure name="deleteUser" scope="element1 element2 element3">
```

If you do not specify any scope - the procedure will be protected by the MobileFirst default security scope.

Disabled protection

Access to a resource with **disabled protection** is allowed to every client. No security or application management features are enforced on access to such resource.

Java adapters

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

JavaScript adapters

If you want to disable protection, you can use `secured="false"`.
This will also disable the default scope:

```
<procedure name="deleteUser" secured="false">
```

Tutorials to follow next

Continue reading about authentication in the following tutorials:

- Creating a security check (../creating-a-security-check)
- Implementing the CredentialsValidationSecurityCheck (../credentials-validation)

- Implementing the UserAuthenticationSecurityCheck (../user-authentication)