Authorization concepts

Overview

The MobileFirst 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, the 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 entities

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

Security Check

A security check is an entity that is responsible for obtaining and validating client credentials.
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

Challenge Handler

When trying to access a protected resource, the client may be faced with a challenge. A challenge is a question, a security test, a prompt by the server to make sure you are allowed to access this resource. Most commonly, this challenge is a request for credentials, such as a username and password.

In the client code, this challenge must be handled by an object called a challenge handler. It is important to note that once a challenge is received, it cannot be ignored. You must answer it, or cancel it. Ignoring a challenge may lead to unexpected behavior.

> Learn more about security checks in the Creating a Security Check (../creating-a-security-check/) tutorial, and about challenge handlers in the Credentials Validation (../credentials-validation) tutorial.

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**, for example `element1 element2 element3`. The MobileFirst security framework requires an access token for any adapter resource even if the resource is not explicitly assigned a scope.

Scope Element

A scope element can be either:

- The name of a security check.
- An arbitrary keyword such as `access-restricted` or `deletePrivilege` which defines the level of security needed for this resource. This keyword will later be mapped to a security check.

Scope Mapping

By default, the **scope elements** you write in your **scope** are mapped to a **security check with the same name**.
For example, if you write a security check called `PinCodeAttempts`, you can use a scope element with the same name within your scope.

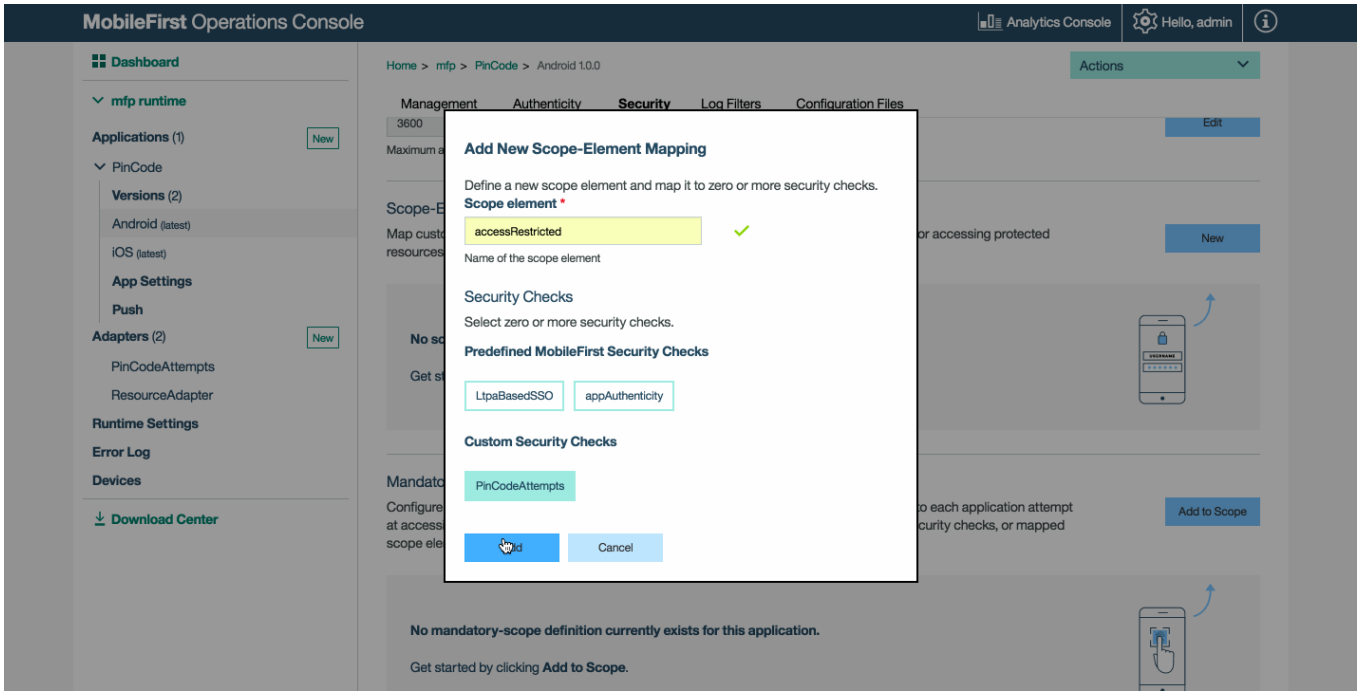
Scope Mapping allows to map scope elements to security checks. When the client asks for a scope element, this configuration defines which security checks should be applied.
For example you can map the scope element `access-restricted` to your `PinCodeAttempts` security check.

This can be useful if you want to protect a resource differently depending on which application is trying to access it.
You can also map it to a list of zero or more security checks.

For example:
scope = `access-restricted deletePrivilege`

- In app A
  - `access-restricted` is mapped to `PinCodeAttempts`
  - `deletePrivilege` is mapped to an empty string
- In app B
  - `access-restricted` is mapped to `PinCodeAttempts`
  - `deletePrivilege` is mapped to `UserLogin`

> To map your scope element to an empty string, do not select any security check in the "Add New Scope Element Mapping" popup.

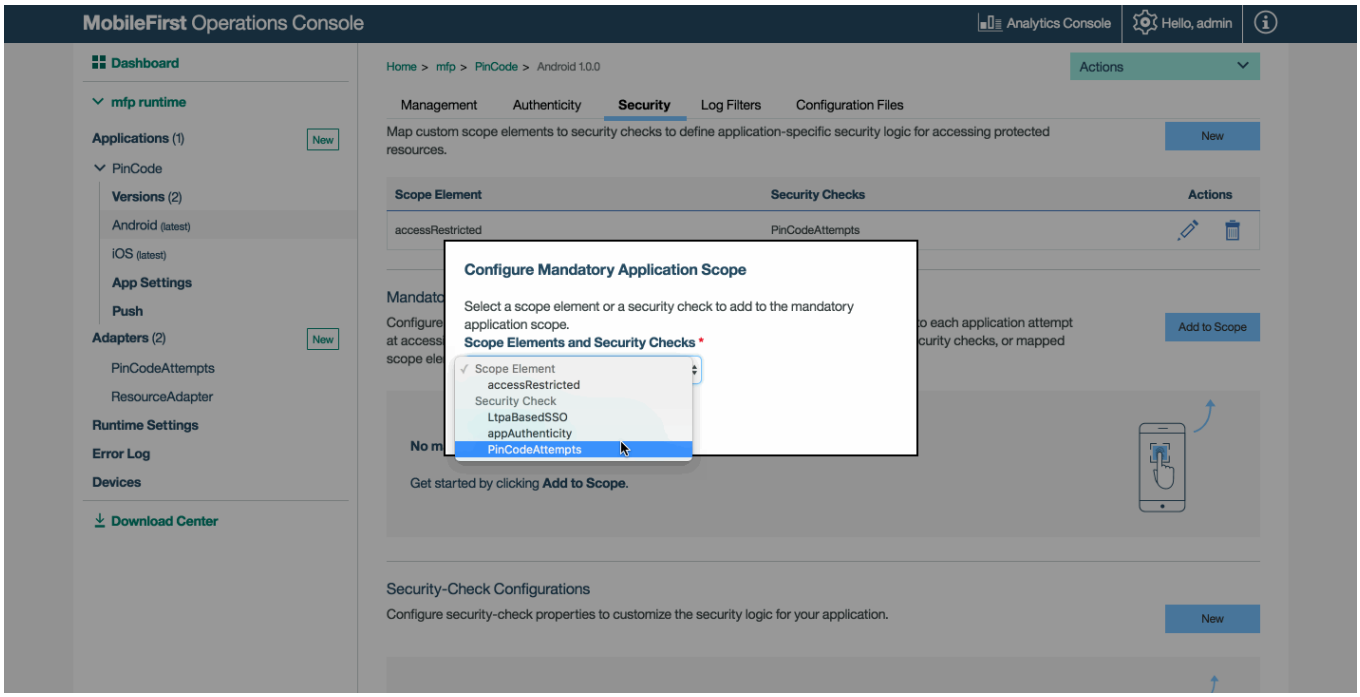


Protecting resources

Your resources can be protected by one of several ways:

Mandatory application scope

At the application level, you can define a scope that will apply to all the resources used by this application. In the MobileFirst Operations Console, select **[your application] → Security tab**. Under **Mandatory Application Scope** click on **Add to Scope**.



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, or set the scope to an empty string, the MobileFirst security framework still requires an access token for any incoming request.

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: `scope="element1 element2 element3"`

If you do not specify any scope, or use an empty string - the MobileFirst security framework still requires an access token for any incoming request.

Disabling protection

Disabling protection allows any client to access the resource, the MobileFirst security framework will **not** require an access token.

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"`.

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

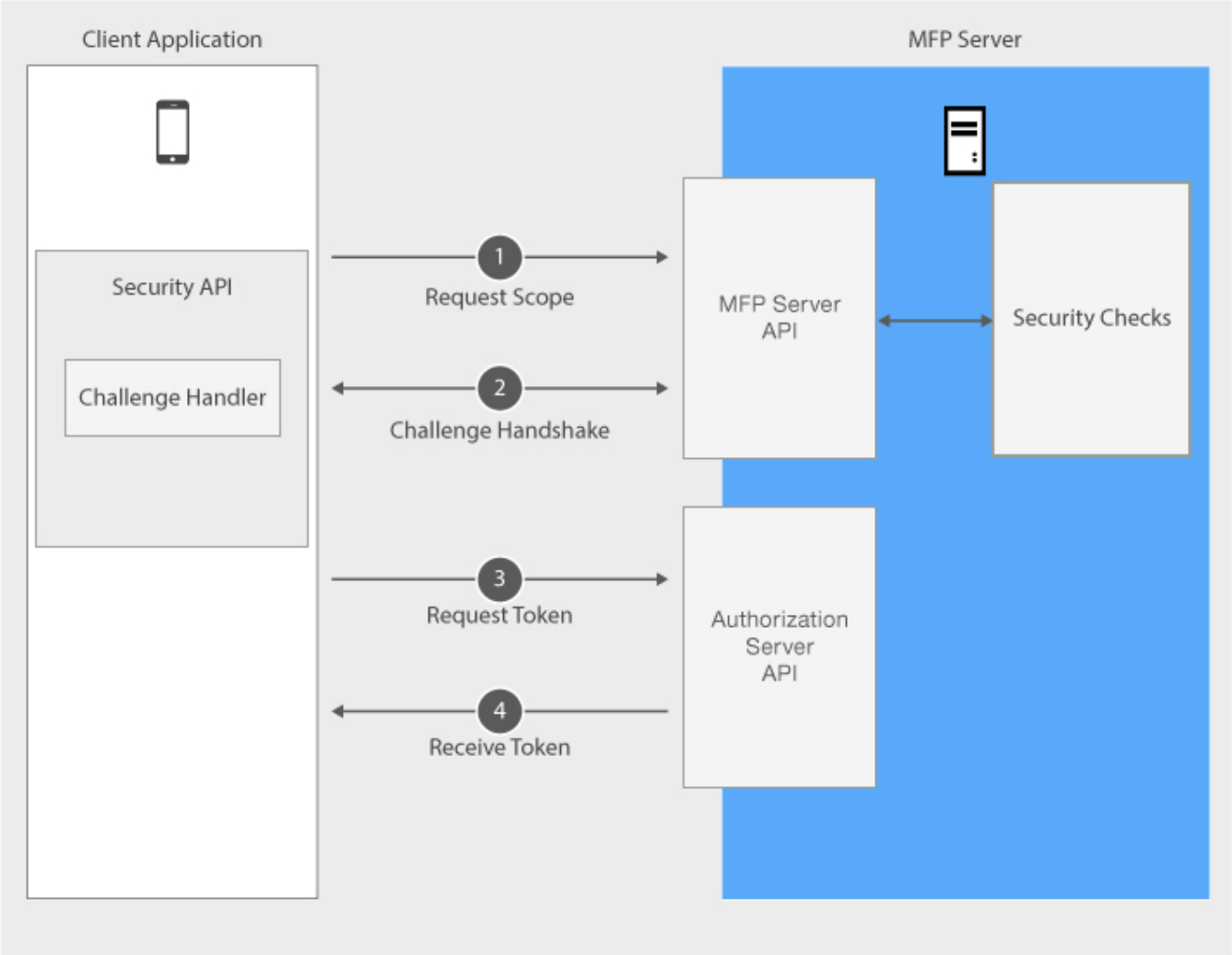
Authorization flow

The authorization flow has two phases:

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

Obtaining an access token

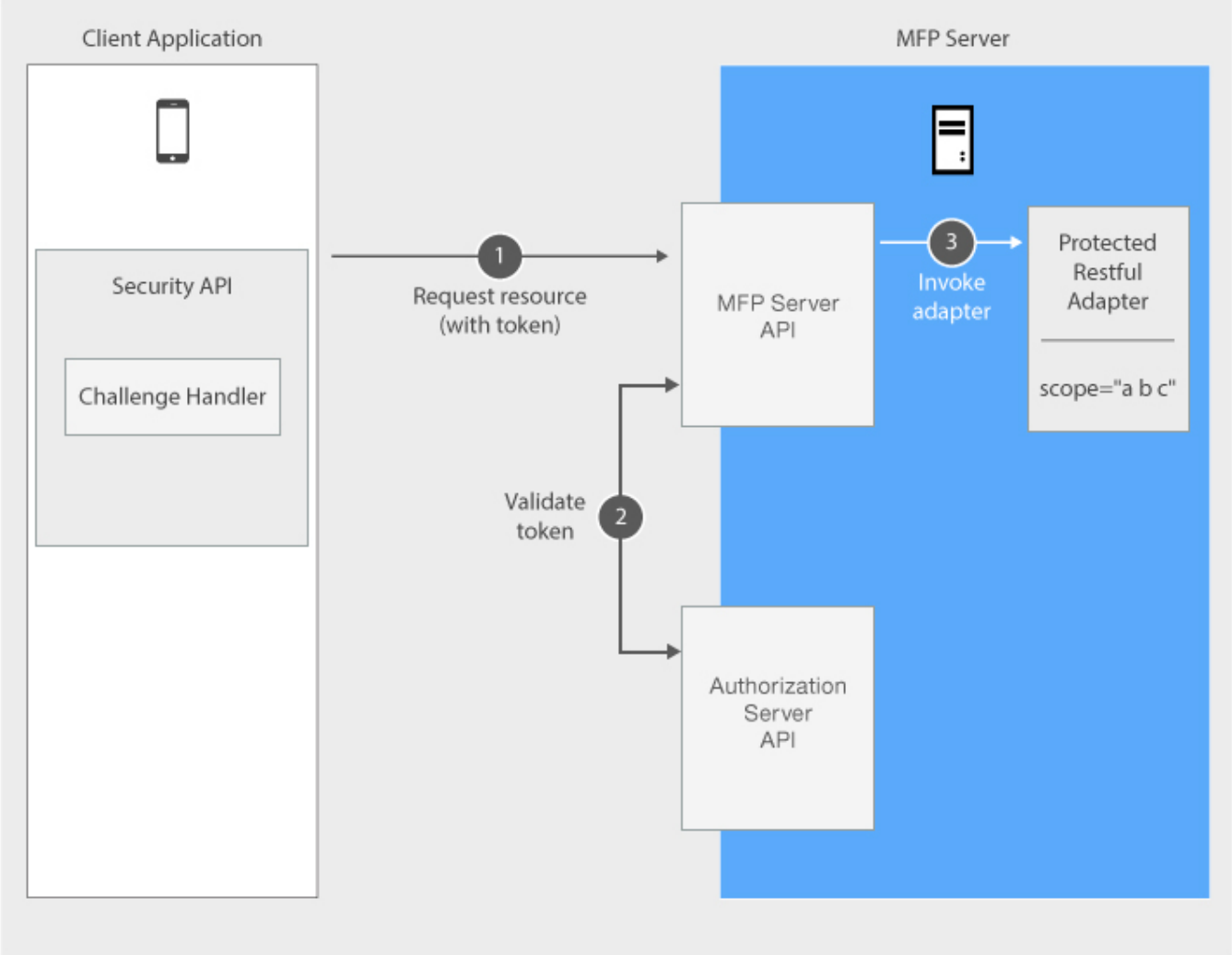
In this phase, the client undergoes **security checks** in order to receive an access token.



1. Client application sends a request to obtain access token for a specified scope.
2. Client application undergoes security checks according to the requested scope.
3. After a successful completion of the challenge process, client application forwards the request to the authorization Server.
4. Client application receives the access 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 (../protecting-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.

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)