Device provisioning concepts

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.0/authentication-security/device-provisioning-concepts.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

Device provisioning is one of the most advanced and complex security features that IBM MobileFirst Platform Foundation provides.
Device provisioning is the process of attaching a certificate to the device identity.
Device identity (or in short – device ID) is similar to user identity, but is used to uniquely identify a specific device.
Device identity is essential for various features, such as

- Push notifications – You want to know which device you are sending notification to.
- Reports – You want to know how many devices are using your server.

Knowing the device identity opens a wide array of security integration possibilities. For example, you can decide which devices are allowed to communicate with MobileFirst Server.
In this tutorial, you learn what device provisioning is, what types of device provisioning are supported by IBM MobileFirst Platform Foundation and what artifacts are involved in the process of device provisioning.

This tutorial covers the following topics:

- Types of device provisioning
- Device ID
- Understanding device provisioning
- No provisioning
- Auto provisioning
- Custom provisioning

Types of device provisioning

IBM MobileFirst Platform Foundation supports three types of device provisioning:

- No provisioning
- Auto provisioning
- Custom provisioning

This tutorial focuses on the first two types.
For more information about custom provisioning, see Custom device provisioning (../../authentication-security/custom-device-provisioning/).

Device ID

The device ID is automatically generated by the client-side framework when requested by MobileFirst Server.
The device ID is used to uniquely identify a specific device with the server.
Similar to the way a user ID is used for user authentication, the device ID is used for device authentication.
Device provisioning is based on the device ID and supported on the Android and iOS platforms.

Understanding device provisioning

Device provisioning is a process where a certificate is issued by MobileFirst Server for a specific device.
The issued certificate contains device information that is obtained during the provisioning process.
Before a certificate is issued to a specific device, the server can perform extra validations on the received device credentials.
You can configure your own CA keystore for the generation of the device provisioning certificate.

No provisioning

No provisioning is appropriate for development environments.
Using **No provisioning** means that the provisioning process is not triggered (requested) by MobileFirst Server.
The application obtains the device ID and sends it to the server as-is.
The server does not validate whether this device is allowed to communicate with it.
The certificate is not issued and not requested at any stage.
No provisioning is the default setting for mobile applications.

- If you use the default security settings, you do not have to enable **No provisioning** manually.
- If you use a custom security test to protect a resource that requires device identity and you want to use **No provisioning**, add the realm to your security test. For example:

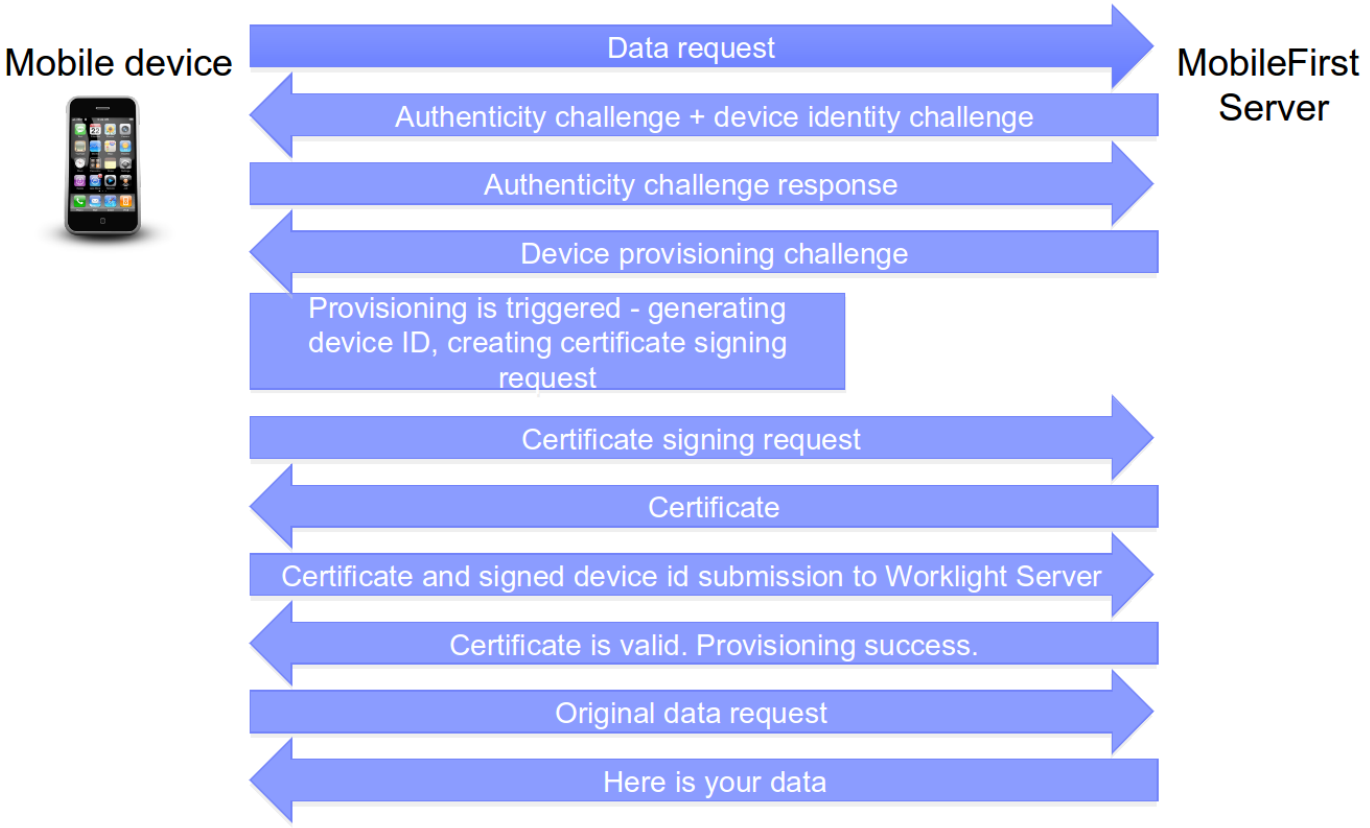
```
<customSecurityTest name="customTests">
    ...
    <test realm="wl_anonymousUserRealm" isInternalUserID="true" />
    <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" /
>
</customSecurityTest>
```

Auto provisioning

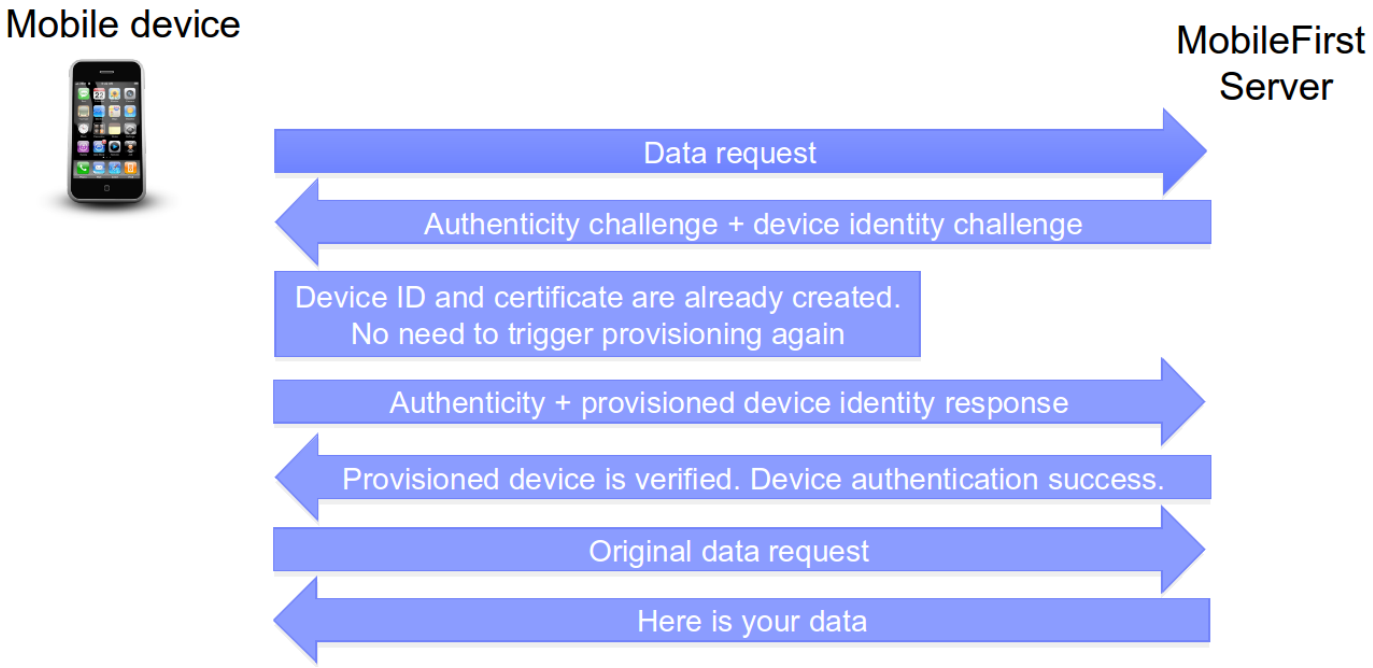
Auto provisioning is an automated one-time process during which a certificate is issued by MobileFirst Server and sent to the client application.
Auto provisioning is triggered by a server when it requests a provisioned device identity.
The application obtains the device ID and starts an automated provisioning process.
The server collects the supplied device information and issues a certificate by using the server-side CA keystore.
The certificate is issued to any device that requests it, therefore **Auto provisioning** makes sense only when it is used after a successful application authenticity check.

First application start

Authenticity check is a proprietary MobileFirst technology, which makes sure that the application is the authentic one and was not modified by anyone.



Subsequent application starts



Enabling auto provisioning

To enable **Auto provisioning**, add the following realms to your `authentication-config.xml` file.

```
<customSecurityTest name="customTests">
    ...
    <test realm="wl_authenticityRealm" />
    <test realm="wl_deviceAutoProvisioningRealm" isInternalDeviceID="true" /
>
</customSecurityTest>

<mobileSecurityTest name="mobileTests">
    ...
    <testAppAuthenticity/>
    <testDeviceId provisioningType="auto" />
</mobileSecurityTest>
```

Keystore

By default, MobileFirst Server uses its internal keystore to issue a certificate.
You can tell the server to use your own keystore by adjusting the `worklight.properties` file.
Note: The `wl.ca.keystore.path` property value can be either relative to the `/server/` folder of the MobileFirst project or absolute to the file system.

```
#################################################################################################
################################
#    Worklight Default Certificate (For device provisioning)
#################################################################################################
################################
# You can change the default behavior with regard to CA certificates. You can also implement custom provis
ioning.
# If you want to change the auto-provisioning mechanism to use different granularity (application, device or g
roup) or a
# different list of pre-required realms, you can create your own customized authenticator, login module and c
hallenge handler.
# For more information, see the "Custom Authenticator and Login Module" Getting Started training module.</
p>
<p>#The path to the keystore, relative to the server folder in the Worklight Project, for example: conf/my-cer
t.jks
#wl.ca.keystore.path=
#The type of the keystore file. Valid values are jks or pkcs12.
#wl.ca.keystore.type=
#The password to the keystore file.
#wl.ca.keystore.password=
#The alias of the entry where the private key and certificate are stored, in the keystore.
#wl.ca.key.alias=
#The password to the alias in the keystore.
#wl.ca.key.alias.password=
#################################################################################################
################################
#    Worklight SSL keystore
#################################################################################################
################################
#SSL certificate keystore location.
ssl.keystore.path=conf/default.keystore
#SSL certificate keystore type (jks or PKCS12)
ssl.keystore.type=jks
#SSL certificate keystore password.
ssl.keystore.password=worklight
```

You must use **Auto provisioning** together with *application authenticity* protection.
For more information, see Application Authenticity Protection (../../authentication-security/application-authenticity-protection/).

Custom provisioning

Custom provisioning is an extension of Auto provisioning.
With Custom provisioning, you can add custom CSR and certificate validation functions to define custom device provisioning rules.
For more information, see Custom device provisioning (../../authentication-security/custom-device-provisioning/).