

Implementing the challenge handler in Windows 8.1 Universal and Windows 10 UWP applications

Overview

When trying to access a protected resource, the server (the security check) will send back to the client a list containing one or more **challenges** for the client to handle.

This list is received as a `JSON` object, listing the security check name with an optional `JSON` of additional data:

```
{
  "challenges": {
    "SomeSecurityCheck1": null,
    "SomeSecurityCheck2": {
      "some property": "some value"
    }
  }
}
```

The client should then register a **challenge handler** for each security check.

The challenge handler defines the client-side behavior that is specific to the security check.

Creating the challenge handler

A challenge handler is a class responsible for handling challenges sent by the MobileFirst server, such as displaying a login screen, collecting credentials and submitting them back to the security check.

In this example, the security check is `PinCodeAttempts` which was defined in `Implementing the CredentialsValidationSecurityCheck (./security-check)`. The challenge sent by this security check contains the number of remaining attempts to login (`remainingAttempts`), and an optional `errorMsg`.

Create a C# class that extends `Worklight.ChallengeHandler`:

```
public class PinCodeChallengeHandler : Worklight.ChallengeHandler
{
}
```

Handling the challenge

The minimum requirement from the `ChallengeHandler` class is to implement a constructor and a `HandleChallenge` method, that is responsible for asking the user to provide the credentials. The `HandleChallenge` method receives the challenge as a `WorklightResponse`.

Learn more about the `ChallengeHandler` class in the user documentation.

Add a constructor method:

```
public PinCodeChallengeHandler(String securityCheck) {
  this.securityCheck = securityCheck;
}
```

In this `HandleChallenge` example, an alert is displayed asking to enter the PIN code:

```
public override void HandleChallenge(WorklightResponse challenge)
{
    try
    {
        if(challenge.ResponseJSON["errorMsg"]!=null && challenge.ResponseJSON["errorMsg"].Type == JToken.Null)
        {
            showChallenge("This data requires a PIN code.\nRemaining attempts: " + challenge.ResponseJSON["remainingAttempts"]);
            shouldsubmitchallenge = true;
        }
        else
        {
            showChallenge(challenge.ResponseJSON["errorMsg"] +
                \nRemaining attempts: " + challenge.ResponseJSON["remainingAttempts"]);
        }
    } catch (Exception e)
    {
        Debug.WriteLine(e.StackTrace);
    }
}
```

The implementation of `showChallenge` is included in the sample application.

If the credentials are incorrect, you can expect the framework to call `HandleChallenge` again.

Submitting the challenge's answer

Once the credentials have been collected from the UI, use the `ChallengeHandler`'s `ShouldSubmitChallengeAnswer()` and `GetChallengeAnswer()` method to send an answer back to the security check. `ShouldSubmitChallengeAnswer()` returns a boolean indicating if the challenge response should be sent back to the security check. In this example `PinCodeAttempts` expects a property called `pin` containing the submitted PIN code:

```
public override bool ShouldSubmitChallengeAnswer()
{
    JObject pinJSON = new JObject();
    pinJSON.Add("pin", pinCodeTxt.Text);
    this.challengeAnswer = pinJSON;
    return this.shouldsubmitchallenge;
}

public override JObject GetChallengeAnswer()
{
    return this.challengeAnswer;
}
```

Cancelling the challenge

In some cases, such as clicking a "Cancel" button in the UI, you want to tell the framework to discard this challenge completely.

To achieve this, override `ShouldSubmitFailure` and `GetSubmitFailureResponse` methods:

```
public override bool ShouldSubmitFailure()
{
    return shouldsubmitfailure;
}
public override WorklightResponse GetSubmitFailureResponse()
{
    return new WorklightResponse(false, "User cancelled" , new JObject (), "",(int) HttpStatusCode.InternalServerError);
}
```

Handling failures

Not Yet implemented

Handling successes

Not Yet implemented

Registering the challenge handler

In order for the challenge handler to listen for the right challenges, you must tell the framework to associate the challenge handler with a specific security check name.

This is done by initializing the challenge handler with the security check like this:

```
PinCodeChallengeHandler pinCodeChallengeHandler = new PinCodeChallengeHandler("PinCodeAttempts");
```

You must then **register** the challenge handler instance:

```
IWorklightClient client = WorklightClient.CreateInstance();
client.RegisterChallengeHandler(pinCodeChallengeHandler);
```

Sample application

The sample **PinCodeWin8** and **PinCodeWin10** are C# applications that uses `ResourceRequest` to get a bank balance.

The method is protected with a PIN code, with a maximum of 3 attempts.

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/SecurityCheckAdapters/tree/release80>) the SecurityCheckAdapters Maven project.

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/PinCodeWin8/tree/release80>) the Windows 8 project. Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/PinCodeWin10/tree/release80>) the Windows 10 UWP project.

Sample usage

- Use either Maven or MobileFirst Developer CLI to build and deploy the available **ResourceAdapter** and **PinCodeAttempts** adapters (`../..creating-adapters/`).
- Ensure the sample is registered in the MobileFirst Server by running the command: `mfpdev app register` from a **command-line** window.

- Map the `accessRestricted` scope to the `PinCodeAttempts` security check:
 - In the MobileFirst Operations Console, under **Applications** → **PinCode** → **Security** → **Map scope elements to security checks.**, add a mapping from `accessRestricted` to `PinCodeAttempts`.
 - Alternatively, from the **Command-line**, navigate to the project's root folder and run the command: `mfpdev app push`.

Learn more about the `mfpdev app push`/`push` commands in the [Using MobileFirst Developer CLI to manage MobileFirst artifacts \(../../../../using-the-mfpf-sdk/using-mobilefirst-developer-cli-to-manage-mobilefirst-artifacts\)](#).