Trusteer for Android

Overview

Trusteer Mobile SDK collects multiple mobile device risk factors and provides them to the mobile app, enabling organizations to restrict mobile app functionality based on risk levels. In your IBM MobileFirst Platform Foundation application, you may want to protect access to some specific resources or procedures based risk levels, such as detected malware or whether the device is jailbroken or rooted. For example, you could prevent a malware-ridden device from logging into your banking app, and prevent rooted devices from using the “transfer funds” feature.

Obtain Trusteer SDK for Android

To use Trusteer within an IBM MobileFirst Platform Foundation application, you first need to separately obtain the Trusteer SDK, license information, required libraries and manifest files. See Trusteer documentation or contact Trusteer support if you are missing any of those items.

You may receive those items either as separate files, or as an MobileFirst Application Component (WLC). You still need to follow these steps if you use a WLC file.

Install Trusteer SDK

If you’ve received an Application Component from Trusteer, search the IBM MobileFirst documentation for “Adding application components to MobileFirst projects”. Then make sure the file “**tas.license**” contains your license information.

If you’ve received the Trusteer SDK, follow the Trusteer documentation to add the Trusteer SDK to your Android project. Make sure you have the “**tas.license**” file in your “**assets**” folder. Its format should be:

```
vendorId=com.mycompany
clientId=my.client.id
clientKey=YMAQAABNFUWS2L
```

Access Risk Items in JavaScript

Optionally, you can access the client-side generated Trusteer data object using the following API:

```
WL.Trusteer.getRiskAssessment(onSuccess, onFailure);
```

Where `onSuccess` is a function that will receive a `JSON` object containing all the data processed by Trusteer. See Trusteer documentation to get information on each risk item.

```
function onSuccess(result) {
  //See in the logs what the full result looks like
  WL.Logger.debug(JSON.stringify(result));
  //Check for a specific flag
  if (result["os.rooted"]["value"] != 0) {
    alert("This device is jailbroken!");
  }
}
```

Access Risk Items in Java

Optionally, you can access the client-side generated Trusteer data object using the following API:

```
WLTrusteer trusteer = WLTrusteer.getInstance();
JSONObject risks = trusteer.getRiskAssessment();
```

This returns an `JSONObject` of all the data processed by Truster. See Trusteer documentation to get information on each risk item.

```
JSONObject rooted = (JSONObject) risks.get("os.rooted");
if(rooted.getInt("value") > 0){
    //device is rooted
}
```

Authentication Config

To prevent access to specific resources when a device is at risk, you can protect your adapter procedures or your applications with a custom security test containing a Trusteer realm. See MobileFirst documentation for general information on security tests and realms.

The Trusteer realm will check the data generated by the Trusteer SDK and allow/reject the request based on the parameters you set.

Here is an example of a Trusteer realm and login module you can add to your **authenticationConfig.xml**.

```
<realms>
  ...
  <realm name="wl_basicTrusteerFraudDetectionRealm" loginModule="trusteerFraudDetectionLogin"
>
    <className>
      com.worklight.core.auth.ext.TrusteerAuthenticator
    </className>
    <parameter name="rooted-device" value="block" />
    <parameter name="device-with-malware" value="block" />
    <parameter name="rooted-hiders" value="block" />
    <parameter name="unsecured-wifi" value="alert" />
    <parameter name="outdated-configuration" value="alert" />
  </realm>
</realms>
```

```
<loginModules>
  ...
  <loginModule name="trusteerFraudDetectionLogin"
>
    <className>
      com.worklight.core.auth.ext.TrusteerLoginModule
    </className>
  </loginModule>
</loginModules>
```

This realm contains 5 parameters:

- rooted-device - indicates whether the device is rooted (android) or jailbroken (iOS)
- device-with-malware - indicates whether the device contains malware
- rooted-hiders - indicate that the device contains root hiders applications that hides the fact that the device is rooted/jailbroken
- unsecured-wifi - indicates that the device is currently connected to an insecure wifi
- outdated-configuration - indicates that Trusteer SDK configuration hasn't updated for some time (didn't connect to the Trusteer server)

The possible values are `block`, `alert` or `accept`.

JavaScript Challenge Handler

Assuming you've added a Trusteer realm to your server's authentication configuration file, you can register a challenge handler to receive the responses from the authenticator.

```
var trusteerChallengeHandler = WL.Client.createWLChallengeHandler("wl_basicTrusteerFraudDetection
Realm");
```

Notice that you are registering a `WLChallengeHandler` and not a `ChallengeHandler`. See IBM MobileFirst documentation on `WLChallengeHandler`.

If you have set one of your realm options to `block`, a blocking event will trigger `handleFailure`.

```
trusteerChallengeHandler.handleFailure = function(error) {
    WL.SimpleDialog.show("Error", "Operation failed. Please contact customer support (reason code: " +
error.reason + ")",  [{text:"OK"}]);
};
```

`error.reason` can be one of the following:

- `TASROOT`
- `TAS ROOTEVIDENCE`
- `TAS MALWARE`
- `TASWIFI`
- `TAS OUTDATED`
- `TASINVALIDHEADER`
- `TASNOHEADER`

If your have set one of your realm options to alert, you can catch the alert event by implementing the `processSuccess` method.

```
trusteerChallengeHandler.processSuccess = function(identity) {
  var alerts = identity.attributes.alerts; //Array of alerts codes
  if (alerts.length > 0) {
    WL.SimpleDialog.show("Warning", "Please note that your device is: " + alerts, [{
      text: "OK "
    }]);
  }
};
```

Native Challenge Handler

Assuming you've added a Trusteer realm to your server's authentication configuration file, you can register a challenge handler to receive the responses from the authenticator.

Create a class that extends `WLChallengeHandler`:

```
public class TrusteerChallengeHandler extends WLChallengeHandler{
//...
}
```

In your application code, register your newly created challenge handler for your Trusteer realm:

```
WLClient.getInstance().registerChallenge(
  new TrusteerChallengeHandler("wl_basicTrusteerFraudDetectionRealm")
;
);
```

If you have set one of your realm options to block, a blocking event will trigger `handleFailure`.

```
public class TrusteerChallengeHandler extends WLChallengeHandler {
//...
  public void handleFailure(JSONObject error){
      String errorReason = error.getString("reason");
      //Show user that the request was denied.
  }
//...
}
```

`error.reason` can be one of the following:

- `TASROOT`
- `TAS ROOTEVIDENCE`
- `TAS MALWARE`
- `TASWIFI`
- `TAS OUTDATED`
- `TASINVALIDHEADER`
- `TASNOHEADER`

If your have set one of your realm options to alert, you can catch the alert event by implementing the `handleSuccess` method.

```
public class TrusteerChallengeHandler extends WLChallengeHandler{
//…
  public void handleSuccess(JSONObject identity){
    JSONArray alerts = identity.getJSONObject("attributes").getJSONArray("alerts")
;

    if(alerts.length() > 0) {
        //Alert the user of potential issues.
    }
  }
//...
}
```

Sample application

Click to download (https://github.ibm.com/MFPSamples/TrusteerAndroid) the MobileFirst project.

Sample Setup

Download the TrusteerIntegrationProject. Import it into your MobileFirst Studio workspace. For pure native, download and import TrusteerAndroidNativeProject as well (update **wlclient.properties** with your IP address) Install the Trusteer SDK into your application(s) following the steps explained previously. Deploy and run it on your Android device.

The sample will display whether or not it successfully loaded the Trusteer SDK. It will display whether your device is rooted or not. It features a button to "**get sensitive data**", which invokes a dummy procedure protected by a Trusteer realm. Depending on the state of your device, you should see "**this is sensitive data**", or an error explaining why your request was rejected.