Trusteer iOS Integration

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/product-integration/6.3/trusteer-ios.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new) ### 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 iOS

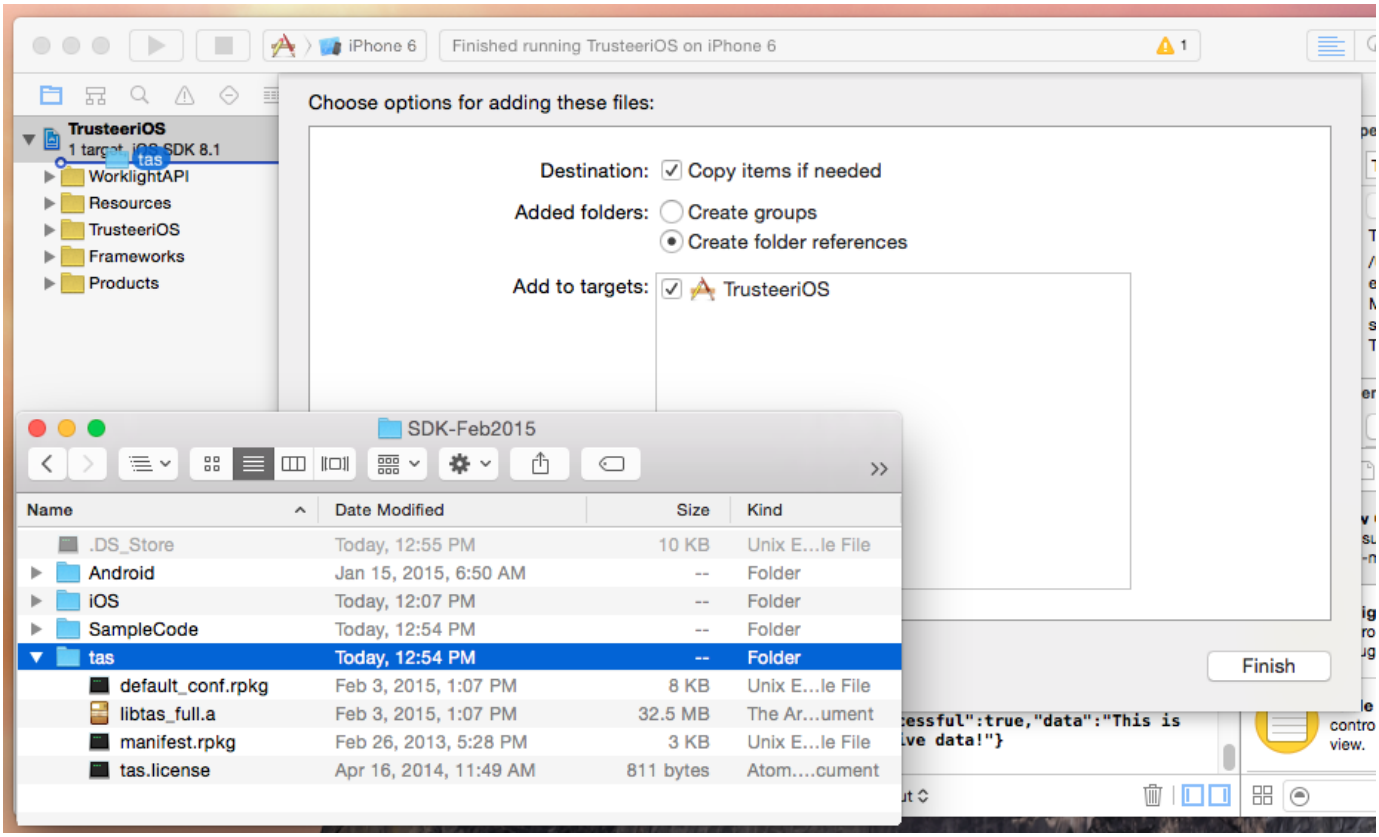
Before starting make sure you have the following items:

- Trusteer Mobile iOS library (libtas_full.a)
- A MobileFirst-compatible Trusteer license file (tas.license)
- A Trusteer configuration package (default_conf.rpkg)
- A Trusteer Application Security Manifest (manifest.rpkg)

See Trusteer documentation or contact Trusteer support if you are missing any of those items.
You may receive those items either as standalone files, or as an MobileFirst Application Component.
To learn more about installing an MobileFirst Application Component (*.WLC) see MobileFirst documentation.
Note: You still need to follow the following steps if you use a WLC file.

Copy Files Into XCode Project

Create a directory on your system called “tas” containing the 4 above files. If you’ve installed Trusteer as an Application Component, this folder will be created for you in the native folder.
In your XCode project (whether MobileFirst-generated Hybrid, or your own using MobileFirst Native API), drag the folder created above onto your project navigator.
Check “Copy items into destination group’s folder (if needed)”.
Select “Create folder references for any added folders”. Make sure your target is selected.

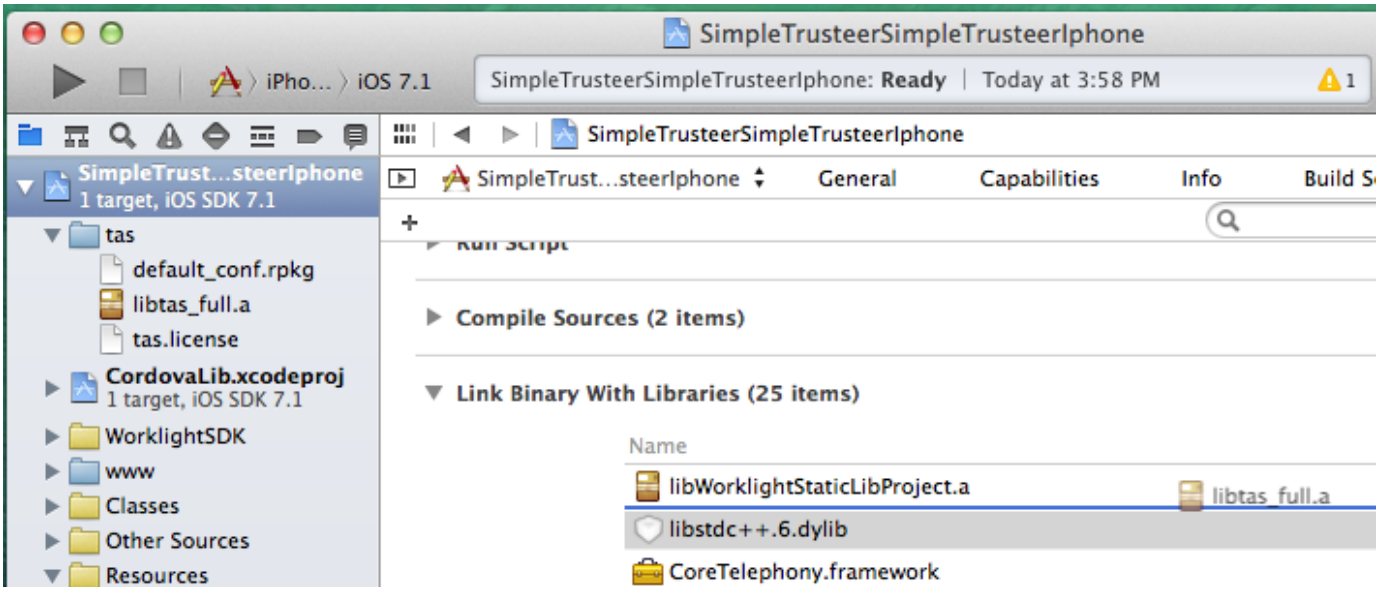


Drag the "**tas.license**" file from the tas folder into the "**Resources**" folder.
Make sure the file contains your license information.
The format of the file should be:

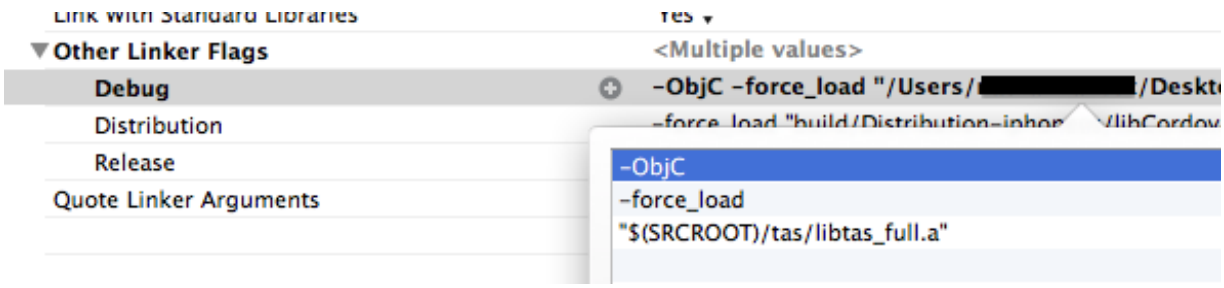
```
vendorId=com.mycompany
clientId=my.client.id
clientKey=YMAQAABNFUWS2
```

Link Trusteer Libraries

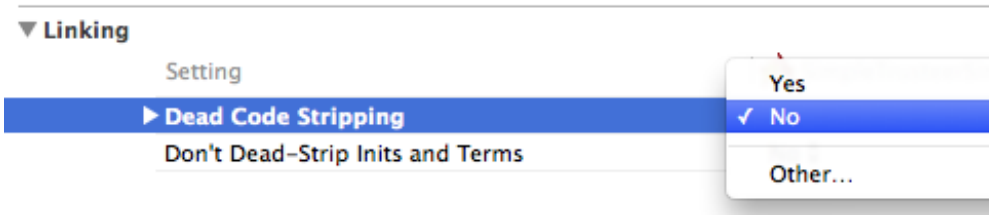
In **Build Phases → Link Binary With Libraries**, drag and drop **libtas_full.a** to link your project with the Trusteer library. If you get linker errors, move **libWorklightStaticLibProject.a** to the top of the list.



In **Build Settings → Linking → Other Linker Flags**, add: `-force_load "$(SRCROOT)/tas/libtas_full.a"`.



In **Build Settings → Linking → Dead Code Stripping**, select `NO`.



In **Build Settings → Deployment → Strip Linked Product**, select `NO`.

Note : If the iOS project is native, please follow the standard MobileFirst native requirements are described in the MobileFirst documentation as well as the requirements described in the Trusteer documentation. For example, Trusteer requires CoreMotion.framework in addition to MobileFirst's standard requirements.

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 Objective-C

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

```
#import “WLTrusteer.h”
NSDictionary* risks =[[WLTrusteer sharedInstance] riskAssessment];<br />
```

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

```
//See in the logs what the full result looks like
NSLog(@"%@",risks);
//Check for a specific flag
NSNumber* rooted = [[risks objectForKey:@"os.rooted"] objectForKey:@"value"];
if([rooted intValue]!= 0){
 NSLog(@"Device is jailbroken!");
}
```

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"}]);<br />
};
```

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

- `TAS_ROOT`
- `TAS_ROOT_EVIDENCE`
- `TAS_MALWARE`
- `TAS_WIFI`
- `TAS_OUTDATED`
- `TAS_INVALID_HEADER`

- `TAS_NO_HEADER`

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`:

```
#import "WLChallengeHandler.h"
@interface TrusteerChallengeHandler : WLChallengeHandler
@end
```

Register your newly created challenge handler for your Trusteer realm:

```
[[WLClient sharedInstance] registerChallengeHandler:[[TrusteerChallengeHandler alloc] initWithRealm:
@"wl_basicTrusteerFraudDetectionRealm"]];
```

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

```
@implementation TrusteerChallengeHandler
//...
-(void) handleFailure: (NSDictionary *)failure {
   NSLog(@"Your request could not be completed. Reason code: %@",failure[@"reason"]);
}
//...
@end
```

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

- `TAS_ROOT`
- `TAS_ROOT_EVIDENCE`
- `TAS_MALWARE`
- `TAS_WIFI`
- `TAS_OUTDATED`
- `TAS_INVALID_HEADER`

- `TAS_NO_HEADER`

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

```
@implementation TrusteerChallengeHandler
//...
-(void) handleSuccess:(NSDictionary *)success{
   NSArray* alerts = success[@"attributes"][@"alerts"]
;
   if(alerts &amp;amp;&amp;amp; alerts.count){
      for(NSString* alert in alerts){
         NSLog(@"This device is %@", alert);
      }
   }
}
//...
@end
```

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/TrusteerIntegrationProject.zip) the Studio project.
Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/TrusteeriOSNativeProject.zip) the Native project.

Sample Setup

Download the **TrusteerIntegrationProject**.
Import it into your IBM MobileFirst Studio workspace.
For pure native, download and open **TrusteeriOSNativeProject** as well (update worklight.plist with your IP address).
Install the Trusteer SDK into your application(s) following the steps explained previously.
Deploy and run it on your iOS device.

The sample will display whether or not it successfully loaded the Trusteer SDK.
It will display whether your device is jailbroken 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.