**ThreatMetrix SDK integration**

**SDK V6-2**

Monitor user activities across all digital interactions and devices with our ThreatMetrix SDK. Integrate our SDK in your native apps to detect fraud and security vulnerability originating from mobile devices in real time.

**What is ThreatMetrix Platform?**

ThreatMetrix is a threat detection platform which provides historic context that helps detect user accounts being misused in real time. It builds a comprehensive digital identity to authenticate each connecting user by combining information related to device, anonymized identity information, behavior patterns and threat intelligence.

**How Does it work?**

ThreatMetrix SDK provides mobile application developers with Android specific libraries in the form of an SDK that leverages the ThreatMetrix Platform for detecting fraud and security vulnerabilities originating from mobile devices in real time.

The below steps explain how TMX SDK works:

1. The customer includes the ThreatMetrix SDK in their application which is subsequently installed on an end user's device.
2. The customer’s application initializes the SDK by calling the init() method which performs some basic setup tasks.
3. The customer 's application calls the profile() method to begin device profiling . ThreatMetrix SDK will transmit the collected attributes  and a unique Session ID to the ThreatMetrix Platform.
4. The customer's application transmits the Session ID  to the customer's web server.
5. The customer's web server performs a RESTful HTTPS Web API call to the ThreatMetrix Platform which initiates policy based evaluation to determine the potential risk associated with a transaction. Detailed information regarding the session evaluation is returned, enabling the customer to make an educated decision as to whether this transaction should be accepted, rejected or reviewed.

**Modules used in implementing ThreatMetrix SDK for device profiling**

**TMXProfiling:-** This module is responsible for performing profiling exclusively.  It does not transfer anything over the network so the module does not send or receive data on its own. This module needs to be paired with the TMXProfilingConnections module or a custom profiling connections module in order to send and receive data to and from the backend and therefore, for profiling to occur successfully.

**TMXProfilingConnections:-** It is the default networking module provided by ThreatMetrix, this module only transfer data over the network without changing the data.

*Provide Link for downloading modules*

**Get Our ThreatMetrix SDK**

Include AAR files of TMXProfiling and TMXProfilingConnections module into your **app/libs** folder. Then add the implementation into your app-level **build.gradle** file.

implementation files('libs/TMXProfiling-6.2-107.aar')  
implementation files('libs/TMXProfilingConnections-6.2-107.aar')

**Permissions**

**Mandatory Permissions**

The application must include the following permission in its manifest file.

<uses-permission android:name="android.permission.INTERNET" />

**Optional Permissions**

Add these optional permission in the application manifest file for better profiling. Note that these permission will display a popup on enduser mobile for granting access.

<uses-permission android:name="android.permission.READ\_PHONE\_STATE"/>   
<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>   
<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>   
<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />   
<uses-permission android:name="android.permission.CHANGE\_WIFI\_STATE" />

**Profiling**

Identify the pages in your native application that provide the optimal opportunity for profiling. Typically, profiling takes between 2 and 5 seconds to complete, so when selecting the pages, typically, those containing forms that require the maximum data input will ensure that the visitor stays on the page for the time necessary to complete profiling. Please note that the majority of profiling is completed within a fraction of a second but may take up to 5 seconds to collect the full set of profiling attributes. Common pages where we can initiate profiling are Account creation page, Payments page, Login page.

**Details Required**

For creating a config instance to be passed to init() method orgId, FPserver/Profiling domain is required.

Organisation id - For Try: **afevfjm6** / For Production: **dzppsd1h**

Profiling domain - For Try: **ddc-test.worldpay.com** / For production: **ddc.worldpay.com**

**Using ThreatMetrix SDK**

## **Get Instance and Initialize**

The ThreatMetrix SDK is initialized asynchronously at startup by calling the init() function which is configured with the Config object. At a minimum, the Context and Org IDand the FP server/Profiling domain **must** be specified. The method getInstance() is used to return a singleton instance of the ThreatMetrix Object. The instance is only required to be initialized once. Please note that further init() calls do not have any impact. Here’s an example how you can get instance and initialize for profiling.

**Kotlin Code:-**

class MainActivity : AppCompatActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

var Username: EditText = findViewById(R.id.*textInputEditText*)  
 var Password: EditText = findViewById(R.id.*editTextTextPassword*)

/\*Creating a config instance to be passed to the init() method. This instance must

include orgId and application context otherwise the init()method will fail.\*/

var config = TMXConfig()

//(REQUIRED) Organisation ID  
 .setOrgId("afevfjm6")

// (REQUIRED) FPserver/Profiling domain  
 .setFPServer("ddc-test.worldpay.com")

// (REQUIRED) Application Context  
 .setContext(*applicationContext*)

// (Optional) Register for location services

// Requires ACCESS\_FINE\_LOCATION or ACCESS\_COARSE\_LOCATION permission .setRegisterForLocationServices(true)

/\*Getting instance of ThreatMatrix object by passing config object to it\*/

TMXProfiling.getInstance().init(config)  
/\*Once the valid instance is created for further calls, fire a profile

request\*/

doProfile()

}

}

**JAVA Code:-**

public class LoginActivity extends Activity  
{  
 private EditText m\_accountText;  
 private EditText m\_passwordText;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState)  
 **{** m\_accountText = findViewById(R.id.accountText);  
 m\_passwordText = findViewById(R.id.passwordText)  
 /\*Creating a config instance to be passed to the init() method. This instance

must include orgId and application context otherwise the init() method will

fail.\*/

TMXConfig config = new TMXConfig()  
 //(REQUIRED) Organisation ID  
 .setOrgId(ORG\_ID)  
 // (REQUIRED) FPserver/Profiling domain  
 .setFPServer(FP\_SERVER)  
 // (REQUIRED) FPserver/Profiling domain  
 .setContext(getApplicationContext())

// (Optional) Register for location services

// Requires ACCESS\_FINE\_LOCATION or ACCESS\_COARSE\_LOCATION permission

.setRegisterForLocationServices(true);   
  
 /\*Getting instance of ThreatMatrix object by passing config object to it\*/

TMXProfiling.getInstance().init(config);  
 /\*Once the valid instance is created for further calls, fire a profile request\*/  
 doProfile();  
 **}**}

**Start Profiling**

Create a method to start profiling and send some additional attributes along with profiling information. Here’s the code sample to create profile method for starting profiling.

**Kotlin Code:-**

fun doProfile(){

/\*(OPTIONAL) Assign some custom attributes to be included along with the

profiling Information\*/

var list: ArrayList<String> = ArrayList()  
 list.add("attribute 1")   
 list.add("attribute 2")

var options: TMXProfilingOptions = TMXProfilingOptions()  
 .setCustomAttributes(list*)*

//Fire off the profiling request

/\*The instance of TMXEndNotifier must be passed to the profile() method, the

CompletionNotifier is an implementation of TMXEndNotifier which is explained

in **obtaining profiling result** section\*/   
 var profilinghandle:TMXProfilingHandle = TMXProfiling.getInstance()  
 .profile(options, CompletionNotifier())

}

**JAVA Code:-**

void doProfile()  
{

/\*(OPTIONAL) Assign some custom attributes to be included along with the

profiling Information\*/  
 List<String> list = new ArrayList<String>();  
 list.add("attribute 1");  
 list.add("attribute 2");

TMXProfilingOptions options = new TMXProfilingOptions().setCustomAttributes(list);  
 //Fire off the profiling request

/\*The instance of TMXEndNotifier must be passed to the profile() method, the

CompletionNotifier is an implementation of TMXEndNotifier which is explained

in **obtaining profiling result** section\*/  
 TMXProfilingHandle profilingHandle = TMXProfiling.getInstance().profile(options,  
 new CompletionNotifier());  
}

**Obtaining Profiling Result**

The  result of profiling can be obtained from the ***complete*** method which is overridden from TMXEndNotifier interface class. Implementation of TMXEndNotifier is called once the profiling is completed. Here’s the example of its implementation.

**Kotlin Code:-**

/\*TMXEndNotifier implementation called when profiling is completed

class CompletionNotifier: TMXEndNotifier {

/\*This method gets called when the profiling has finished. Be careful here

because we are not going to be called on the UI thread, and if we want to

update UI elements we can only do it from the UI thread.\*/

override fun complete(result: TMXProfilingHandle.Result) {

//Get the session id to use in API call (AKA session query)

var SESSION\_ID = result.*sessionID*  
  
 *println*("Session Id :- "+result.*sessionID*)  
 *println*("TMX Profiling Status :- "+result.*status*)

}  
}

**JAVA Code:-**

/\*TMXEndNotifier implementation called when profiling is completed

private class CompletionNotifier implements TMXEndNotifier  
{  
  
 private String m\_sessionID;

/\*This method gets called when the profiling has finished. Be careful here

because we are not going to be called on the UI thread, and if we want to

update UI elements we can only do it from the UI thread.\*/  
  
 @Override  
 public void complete(TMXProfilingHandle.Result result)  
 **{**

//Get the session id to use in API call (AKA session query)  
 m\_sessionID = result.getSessionID();  
  
 *println*("Session Id :- "+result.getSessionID());  
 *println*("TMX Profiling Status :- "+result.getStatus());

**}**}

If the result of profiling is returned as status okay then the profiling is successful. For example in the above code samples if **result.status/ result.getStatus()** method returns string **“Okay”** then profiling is successful.