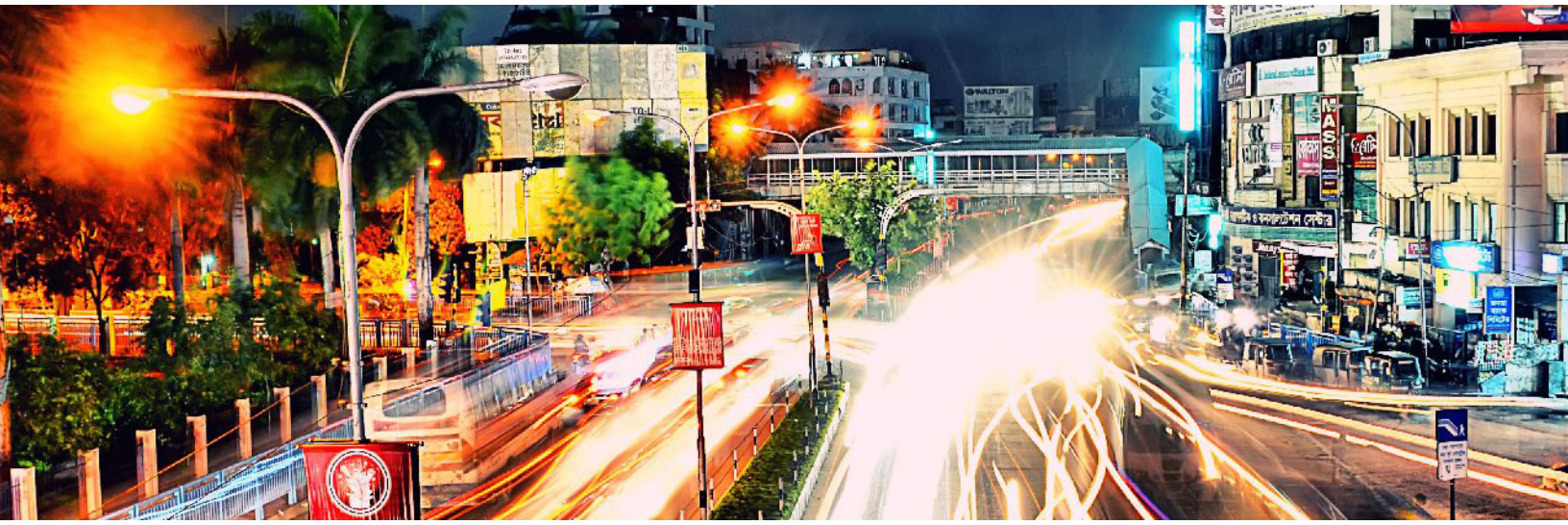
Device Fingerprinting Guide

October 2020



cybersource
A Visa Solution

Cybersource Contact Information

For general information about our company, products, and services, go to http://www.cybersource.com.

For sales questions about any Cybersource service, email sales@cybersource.com or call 650-432-7350 or 888-330-2300 (toll free in the United States).

For support information about any Cybersource service, visit the Support Center:
http://www.cybersource.com/support

Copyright

Restricted Rights Legends

Trademarks

Revision: October 2020

CHAPTER 1

Implementing Device Fingerprinting

Introduction to Device Fingerprinting

Device Fingerprinting service gathers information about the devices that are used to place orders on your web site or about devices that use your mobile application. This information gathering process is called *device profiling*.

Elements of Device Fingerprinting

Device Fingerprints

The device fingerprint, which results from device profiling, is a unique set of identifiers derived from persistent cookies set during device profiling. This device identifier can be the single constant element that you use to detect identity morphing and the true location of a device. When identity morphing occurs, customer and transaction order data might appear to be random and derived from different customers, but the device fingerprint does not change. This fingerprint indicates that the transactions originate from a single device.

Fingerprints enable you to identify many characteristics of a device, for example:

- Connections between accounts and other customer data
- True locations of devices when they are hidden behind a proxy
- Suspicious configurations of devices, such as language settings inconsistent with the country

Device Profiling Options

The following device profiling options can be used within Account Takeover Protection Service business rules:

- Exact ID is a cookie-based identifier that uses a flash guide, HTTP eTags, and HTML5 persistent markers.
- Smart ID is a cookie-less, heuristic-based, device identifier that leverages device attributes and WebRTC internal IP to assign an independent device ID.

- Strong ID uses cryptographic cookies (using a secure, asymmetric, key-pair-based authentication protocol) and, for mobile applications, also uses public key cryptography and Trusted Execution Environments to identify the device. "Enhanced Profiling Using Hosted SSL," page 11 is required to use strong ID.

For information about enhancements to device fingerprinting and profiling, see "Enhanced Profiling Using Hosted SSL," page 11.

How Device Fingerprinting Works

1. You add the device fingerprinting code to your web site, or you add the device fingerprinting code and libraries to your mobile application.
2. A customer opens a page on your web site with their browser or launches your mobile application, and the code you inserted in Step 1 sends information about their device to the device fingerprinting server along with a unique session ID that identifies the session.
3. The device fingerprinting server profiles the device. This profiling process collects device identification information.
4. You send an API request to the server that contains the same session ID that is sent to the fingerprinting server in Step 2.
5. The server returns information from the device profiling performed in Step 3 to Decision Manager or Fraud Management Essentials, which you can use to determine whether the transaction is legitimate or fraudulent.

Enhanced Profiling Using Hosted SSL

Enhanced Profiling is a network-based enhancement that can improve aspects of device fingerprint collection by implementing hosted SSL certificates. The hosted certificates make all browser communication and web objects appear to be from the merchant's domain. The numerous browsers supported on hundreds of device platforms change regularly, potentially creating instances in which the device fingerprint is not reliable or present during a transaction. This situation could be due to a customer explicitly blocking device collection methods or certain default browser settings, neither of which a merchant can control.

The Enhanced Profiling feature also increases the accuracy of device profiling, reduces visitors' potential concerns about third-party content on your web site or mobile application, mitigates third-party cookie blocking, and increases your confidence in blocking transactions from unprofiled devices.

Enhanced Profiling is required for merchants using mobile device fingerprinting SDK implementations and is highly recommended for web site implementations in order to take advantage of strong ID device profiling.

Using Enhanced Profiling can substantially reduce the end-user's ability to filter profiling code. All profiling requests from the visitor's browser or native mobile application are made to a domain that is secured by the merchant's SSL digital certificates. In order to profile a device, profiling tags are placed on your web site, or the Device Fingerprint SDK is embedded in your mobile application. The tags or SDK request objects from and transmit data to the device fingerprinting profiling server. Because our profiling server is a different domain than the original domain from which the initial page or mobile application typically requests resources, visitors who monitor network requests can see that the requests are being made to a third-party server. Enhanced Profiling prevents the blocking of third-party requests, either natively as a feature of the browser or through third-party browser privacy tools.

We can manage your hosted SSL implementation process, which has an associated cost. For more information, contact your account manager.

Setting up Enhanced Profiling:

Step 1 Choose a host name.

Step 2 Provide your host name and the required and optional fields to your services representative. See **"Required fields"** and **"Optional fields"** in the next section.

Step 3 A certificate signing request (CSR) is generated and provided to you. Have the CSR signed by your preferred Certificate Authority (CA).

Step 4 Return the signed certificate, root certificate for your CA, and the chain (or bundle) file to your services representative.

Step 5 The dedicated host name that is assigned for your profiling server is provided. Deploy the SSL certificate to the production environment.

Step 6 Configure your DNS.

Step 7 Update your profiling tags to use the new host name.

The deployment timeframe for SSL certificates is 5 to 7 days upon receipt of the SSL certificate, bundle certificates, and root certificate authority (CA). All of these files are required in order to minimize the deployment time of the SSL certificates.

The following section provide more information about submitting SSL requests. Contact Customer Support if you have questions.

Selecting a Sub-Domain (Host) Name for Your Profiling Server

Choose a host name that does not indicate that the server is used as a security measure. Host name examples include:

- content.yourdomain.com

- img2.yourdomain.com
- cdnA.yourdomain.com

Provide your services representative with your host name and the following required and optional fields:

Required fields

- Country name (two-letter code): US
- Organization name (for example, company) [Internet Widgits Pty Ltd.]: Lemon Bank, Inc.
- Common name or fully qualified domain name (for example, sub-domain that has been chosen): content.lemonbank.com

These required fields are the minimum information needed to create a certificate signing request (CSR). Check with your Certificate Authority (CA) before providing this information to Cybersource to ensure that the CSR is created according to CA requirements.

Optional fields

- State or province name (full name) [some-state]: California
- Locality name (for example, city): San Jose
- Organizational unit name (for example, section): Services

Obtaining the CSR for Your Profiling Server Host Name and a Signed Certificate File

The following examples use *content.yourdomain.com* as the host name. If you decide to use a different name, simply substitute it when using the following instructions.

A certificate signing request (CSR) is generated, which is delivered to you to be signed by your preferred Certificate Authority (CA). You return the signed certificate, root certificate for your CA, and the chain (or bundle) file to Cybersource. The certificate should be in PEM format and named in this format: *content.yourdomain.com.crt*. If the certificate requires a chain file (Intermediate CA bundle file), provide that file using the name *content.yourdomain.com_bundle.crt*. In some cases the CA might require more than one chain; verify this with your CA.

In addition to the CSR, the dedicated host name that is assigned for your profiling server is provided. This unique host name is required in order for you to configure your DNS. The DNS redirection is not available until the SSL certificate is deployed into the production environment. You can direct all traffic to *content.yourdomain.com* by setting up a CNAME record in your DNS.

The entry should resemble the following example:

```
content.yourdomain.com. CNAME h-yourdomain.online-metrix.net.
```

The certificate is then installed on the Profiling Servers.

If your corporate security policy does not allow you to generate the CSR, contact customer support to discuss alternative methods.

In the event that a wildcard certificate is deployed, a Fully Qualified Domain Name (FQDN) is required for monitoring proposes. You must create an entry as outlined in the example above and provide it to Cybersource.

Updating Profiling Code to Use the New Host Name

After you receive confirmation from Cybersource that your SSL certificate is deployed to the profiling servers, change the host name used in the profiling tags and/or SDK to *content.yourdomain.com*

Make this change only after you verify that the certificate is deployed. Making this change before the certificate is deployed (even if the DNS changes are complete) might generate security warnings in your end-users' web browsers.

Notice to European Union Merchants

The European Union's Privacy and Electronic Communications Directive (the "Directive") restricts the deposit and storage of cookies on the devices of customers of online merchants operating in the European Union.

The device fingerprint feature of Decision Manager and Decision Manager Account Takeover Protection Service is one of more than two hundred global fraud detectors and tests. This feature enables the deposit and storage on the customer's computer of a cookie that profiles the specific attributes of the computer used in transactions. This cookie is used to mitigate fraud.

While we cannot provide legal advice to our merchants, we can provide the following information. The restrictions under the Directive require, among other things, that you:

- Provide "clear and comprehensive information" to visitors of your web site about the storage of cookies on their computer.
- Obtain the consent of visitors before depositing and storing cookies on their computer unless certain exceptions apply.

Your compliance with applicable privacy laws depends on how you use the cookies, on what information you disclose to customers, and on what consent you obtain from customers. Because we have no direct connection to your customers, you are responsible for ensuring that cookies are used properly to perform the requested services. We believe that the safest course of action is for you to clearly and conspicuously disclose the use of cookies to your customers and to obtain their consent before placing cookies on their devices. If you operate in Europe and use the device fingerprint, you should consult your legal counsel and other advisors to find out how to comply with the requirements of the Directive and whether an exception might be available for you. We cannot take any position on the storage of cookies on the devices of customers for purposes other than to provide services. When used without the device fingerprint, Decision Manager does not store cookies. Decision Manager Account Takeover Protection Service requires the device fingerprint and must store cookies to operate.

Web Site Implementations

You can deploy device fingerprinting by configuring your web site as described in the following section

To ensure your customers' privacy, fingerprints are encoded as soon as they are received. Fingerprints persist for approximately 24 hours. This interval begins when the customer opens the HTML page with the tags, and it ends when the transaction request is sent. Add the fingerprinting code to your request as early in the transaction process as soon as possible.

Device fingerprinting stops working when the IP address of the domain name changes. To avoid interruptions in device fingerprinting, use domain names instead of using IP addresses and relying on domain name resolution.

Adding the Fingerprinting Code to Your Web Site

The profiling tag specifies these parameters:

- <***org ID***> (mandatory): to obtain this value, contact your sales or support representative and specify whether it is for testing or production.
- <***merchant ID***> (mandatory): your unique merchant ID.
- <***session ID***> (mandatory): a session ID must be a unique identifier for the transaction, such as an order number. It can contain lowercase and uppercase English letters, digits, hyphens (-), and underscores (_). The maximum length is 88 characters. The session ID must be unique for each transaction and for each merchant ID. You can use any string that you are already generating, such as an order number or web session ID. Do not use the same uppercase and lowercase letters to indicate different session IDs.

  The session ID must be unique for each page load regardless of an individual's web session ID. If a user navigates to a profiled page and is assigned a web session, navigates away from the profiled page, then navigates back to the profiled page, the generated session ID should be different and unique. You may use a web session ID, but it is preferable to use an application GUID (Globally Unique Identifier). This measure ensures that a unique ID is generated every time the page is loaded, even if it is the same user reloading the page.

- ***Custom Profiling Domain*** (optional): domain that serves the JavaScript tag. The default is `h.online-metrix.net`. As an alternative, you can implement Enhanced Profiling so that all profiling requests are made to a domain that is secured by your SSL digital certificates. See "Enhanced Profiling Using Hosted SSL," page 11, for more information.

Be sure to copy all characters correctly and to omit the angle brackets (< >) when substituting your values for the variables.

Supported Tag Deployments

Two methods are available to deploy the profiling tag as described in the "Tag Placement" section.

To allow device profiling time to complete, ensure that 3 to 5 seconds elapse between the execution of the profiling code and when your customers submit their orders.

Tag Placement

Place the basic <script> tag in the <head> tag for optimal performance for either method.

For the basic <script> tag with <noscript> tag method, place the <noscript> tag in the <body> tag, as shown in Example 1. Do not place the <noscript> tag in the <head> tag because it contains an <iframe> tag. Placing iframes in the head tag violates W3C validation and might cause problems.

JavaScript Code

```
<head>

    <script type="text/javascript" src="https://h.online-metrix.net/fp/
tags.js?org_id=<org ID>&session_id=<merchant ID><session ID>"></
script>

</head>

<body>

    <noscript>

       <iframe style="width: 100px; height: 100px; border: 0; position:
absolute; top: -5000px;" src="https://h.online-metrix.net/fp/tags?org_
id=<org ID>&session_id=<merchant ID><session ID>"></iframe>

    </noscript>

</body>
```

Basic <script> Tag with <noscript> Tag (Recommended)

This deployment method includes a basic `<script>` tag, which loads Javascript resource `tags.js`, as well as an additional `<noscript>` tag. Using the `<noscript>` tag ensures that profiling occurs, even if JavaScript is disabled.

Example 1 Basic <script> Tag with <noscript> Tag

```
<head>

    <script type="text/javascript" src="https://h.online-metrix.net/fp/
tags.js?org_id=sample_orgID&session_id=sample_merchantIDsample_
sessionID"></script>

</head>

<body>

    <noscript>

       <iframe style="width: 100px; height: 100px; border: 0; position:
absolute; top: -5000px;" src="https://h.online-metrix.net/fp/tags?org_
id=sample_orgID&session_id=sample_merchantIDsample_sessionID"></iframe>

    </noscript>

</body>
```

Basic <script> Tag without <noscript> Tag

This deployment method includes a single `<script>` tag that loads JavaScript resource `tags.js`. If JavaScript is disabled, this tag does not load, and profiling does not occur.

Example 2 Basic <script> Tag without <noscript> Tag

```
<head>

    <script type="text/javascript" src="https://h.online-metrix.net/fp/
tags.js?org_id=sample_orgID&session_id=sample_merchantIDsample_
sessionID"></script>

</head>
```

Mobile Implementations

You can deploy device fingerprinting in Android and iOS applications.

Implementing device fingerprinting in mobile applications requires either Android or iOS platform application programming skills.

Implementing the Device Fingerprinting SDK in Android Applications

To implement the device fingerprinting mobile SDK for Android, you must use Android 4.4 KitKat / Android SDK 19.0 or later.

To implement device fingerprinting in Android applications:

Step 1 Download the *CybersourceTMDeviceFingerprintingMobileSDK_for_Android_v6.0.zip* file from the Business Center Documentation page, and add it to your project.

Device Fingerprint consists of two frameworks: **Tmxprofiling** and **TmxProfilingConnections**.

Step 2 *Add the Tmxprofiling-<version>.aar and TmxProfilingConnections<version>.aar* to your project.

The standard way of including an AAR dependency to Android is described at: https://developer.android.com/studio/projects/android-library.html. Reference the "**Add Your Library as a Dependency**" section.

When updating with the new AAR file, ensure that you select "Clean Project" to delete the cached version of the library.

Step 3 Include the following permission in the mobile application manifest file:

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

</uses-permission>
```

Step 4 Import the SDK to your Activity.

```
import com.threatmetrix.TrustDefender.TMXConfig;
import com.threatmetrix.TrustDefender.TMXEndNotifier;
import com.threatmetrix.TrustDefender.TMXProfiling;
import com.threatmetrix.TrustDefender.TMXProfilingHandle;
import com.threatmetrix.TrustDefender.TMXProfilingOptions;
```

Step 5 Configure and Initialize the SDK.

IMPORTANT:

Org ID	Contact Cybersource Customer Support for this value, and specify whether the Org ID is for testing or production.
Fingerprint server URL	Fully qualified domain name (FQDN) of the server. It must be specified in an FQDN format, for example, *host.domain.com;* NOT in a URL format.

```
TMXConfig config = new TMXConfig()
  .setOrgId("<ORG ID>")
  .setFPServer("<FINGERPRINT SERVER URL>")
  .setContext(getApplicationContext());
TMXProfiling.getInstance().init(config);
```

Step 6 Create your unique session ID for each profiling.

Specify your merchant ID and the session ID as a concatenated value for a variable that is passed to the profiling.

my_variable=your merchant ID + the session ID as a concatenated value.

A session ID must be a unique identifier for the transaction, such as an order number. It can contain lowercase and uppercase English letters, digits, hyphens (-), and underscores (_). Maximum length is 88 characters. The session ID must be unique for each transaction and for each merchant ID. You can use any string that you are already generating, such as an order number or web session ID. Do not use the same uppercase and lowercase letters to indicate different session IDs.

The session ID must be unique for each page load, regardless of an individual's web session ID. If a user navigates to a profiled page and is assigned a web session, navigates away from the profiled page, then navigates back to the profiled page, the generated session ID should be different and unique. You may use a web session ID, but it is preferable to use an application GUID (Globally Unique Identifier). This measure ensures that a unique ID is generated every time the page is loaded, even if it is the same user reloading the page.

Step 7 Implement Profiling with EndNotifier.

```
TMXProfilingOptions options = new TMXProfilingOptions().setCustomAttributes(null);
options.setSessionID(my_variable)
TMXProfilingHandle profilingHandle = TMXProfiling.getInstance().profile(options,
        new CompletionNotifier());
```

```
class CompletionNotifier implements TMXEndNotifier
{
   @Override
   public void complete(TMXProfilingHandle.Result result)
   {
// Once Profile is done. Check the status code in the results dictionary, and use the
session Id in the API.
    }
}
```

Android Code Example

See Appendix B, "Code Examples for Mobile Implementation," on page 62 for the Android code example.

After you add the device fingerprinting mobile SDK to your application, you must specify the session ID in the API request that you send to the system by using the **deviceFingerprintID** Simple Order API request field or the **device_fingerprint_id** SCMP API request field.

Android Return and Error Codes

The following table lists the codes you may encounter when implementing the Device Fingerprinting SDK in an Android application.

The return profiling code **THM_OK** must be present before you send the API request. This code ensures the presence of a complete profile.

Table 1 Android Return and Error Codes

Value	Description
TMX_NotYet	The profiling request is not yet complete.
TMX_OK	Device profiling completed with no errors.
TMX_Connection_Error	A connection issue was encountered between the device and the fingerprint online server. Ensure that you are referencing the server correctly and that you can access it from the device.
TMX_HostNotFound_Error	The host name of the fingerprint server could not be resolved. Ensure that you are referencing the server correctly and that you can access it from the device.

Table 1 Android Return and Error Codes (Continued)

Value	Description
TMX_NetworkTimeout_Error	A timeout occurred while the device was communicating with the fingerprint server. A timeout can occur if the device's internet connection is disabled while it is communicating with the server.
TMX_HostVerification_Error	The fingerprint server host name in use does not match the host name in the certificate, which may be evident when you use the Advanced Profiling feature, or implement in a proxied scenario with the custom URL option. Ensure that a valid certificate is in use on the target host name specified in the custom URL option.
TMX_Internal_Error	A miscellaneous error was detected. Check the input/options used when calling the library.
TMX_Interrupted_Error	The profiling request was interrupted or canceled mid-flight.
TMX_InvalidOrgID	This code is returned if an invalid or NULL value is present in the **org_id** calling option.
TMX_PartialProfile	A connection error resulted in partial profiling.
TMX_Blocked	The profiling request cannot be processed because profiling is blocked due to some conditions. The most common scenario is that the phone screen is off longer than the amount of time specified by the **screenOffTimeout** value. You can customize this value by calling the **Config.setScreenOffTimeout**() method during init(). Config config = new Config() .setContext(getApplicationContext()) .setScreenOffTimeout(180); profile.init(config);
TMX_ConfigurationError	Mobile SDK is not activated for the customer.
TMX_Already_Initialised	SDK is already initialized; use the current instance.
TMX_EndNotifier_Not_Found	**EndNotifier** is not passed to **doProfileRequest,** and it is mandatory in the profile request.
TMX_ThirdPartyLibrary_Not_Found	OkHttp library is not available; include the library. For information about how to include the library see http://square.github.io/okhttp/ **Note** Both OkHttp 2 and OkHttp 3 are supported.

Implementing the Device Fingerprinting SDK in iOS Applications

To develop iOS applications you must be enrolled in the iOS Developer Program, which enables you to upload your applications to the Apple App Store. To link to the device fingerprinting mobile SDK, you must use iOS 8 or later and Apple Xcode 6.0 IDE.

To implement device fingerprinting in iOS applications:

Step 1 Download the *CybersourceTMDeviceFingerprintingMobileSDK_for_iOS_v6.0.zip* (or iOS_Bitcode) file from the Business Center Documentation page and add it to your project.

Step 2 Device Fingerprint consists of two frameworks. The zip file contains two frameworks: **Tmxprofiling** and **TmxProfilingConnections**. Import the device fingerprinting SDK libraries and frameworks into your iOS application:

```
import TMXProfiling
```

Step 3 Specify your merchant ID and the session ID as a concatenated value for a variable that is passed to the TrustDefenderMobile class in your iOS application. In the following example, my_variable = your merchant ID + the session ID as a concatenated value:

```
self.profile.sessionID = @"my_variable";
```

The TrustDefenderMobile class is contained in the: **CybersourceTMDeviceFingerprintingMobileSDK_for_iOS_v6.0.zip** file.

A session ID must be a unique identifier for the transaction, such as an order number. It can contain lowercase and uppercase English letters, digits, hyphens (-), and underscores (_). Maximum length is 88 characters.

The session ID must be unique for each transaction and for each merchant ID. You can use any string that you are already generating, such as an order number or web session ID. Do not use the same uppercase and lowercase letters to indicate different session IDs.

The session ID must be unique for each page load, regardless of an individual's web session ID. If the same user navigates to a profiled page and is assigned a web session, navigates away from the profiled page, then navigates back to the profiled page, the generated session ID should be different and unique. You may use a web session ID, but it is preferable to use an application GUID (Globally Unique Identifier). This measure ensures that a unique ID is generated every time the page is loaded, even if it is the same user reloading the page.

Step 4 Add the `doProfileRequest()` function to your application, and specify the following calling options:

Option	Description
Org ID	Contact customer support for this value and specify whether it is for testing or production.
Fingerprint server URL	Fully qualified domain name (FQDN) of the server. It must be specified in an FQDN format, for example, *host.domain.com*, NOT in URL format.

To fix the “selector not recognized” runtime exceptions when trying to use category methods from a static library, see the following article for more information:

https://developer.apple.com/library/mac/qa/qa1490/_index.html

iOS Code Examples

See Appendix B, "Code Examples for Mobile Implementation," on page 62 for the iOS code examples.

After you add the device fingerprinting mobile SDK to your application, you must specify the session ID in the API request that you send to the system by using the deviceFingerprintID Simple Order API request field or the device_fingerprint_id SCMP API request field.

iOS Return and Error Codes

The following table lists the codes you may encounter when implementing the Device Fingerprinting SDK in an iOS application.

The return profiling code **THMStatusCodeOk** must be present before you send the API request. This code ensures the presence of a complete profile.

Table 2 iOS Return and Error Codes

Value	Description
TMXStatusCodeNotYet	The profiling request is not yet complete.
TMXStatusCodeOk	Device profiling has completed with no errors.

Table 2 iOS Return and Error Codes (Continued)

Value	Description
TMXStatusCodeConnectionError	A connection issue was encountered between the device and the fingerprint online server. Ensure that you are referencing the server correctly and that you can access it from the device.
TMXStatusCodeHostNotFoundError	The host name of the fingerprint server could not be resolved. Ensure that you are referencing the server correctly and that you can access it from the device.
TMXStatusCodeNetworkTimeoutError	A timeout occurred while the device was communicating with the fingerprint server. A timeout can occur if the device's internet connection is disabled while it is communicating with the server.
TMXStatusCodeHostVerificationError	The fingerprint server host name in use does not match the host name in the certificate, which may be evident when you use the Advanced Profiling feature or implement in a proxied scenario with the custom URL option. Ensure that a valid certificate is in use on the target host name specified in the custom URL option.
TMXStatusCodeInternalError	A miscellaneous error was detected. Check the input/options used when calling the library.
TMXStatusCodeInterruptedError	The profiling request was interrupted or canceled mid-flight.
TMXStatusCodePartialProfile	A connection error resulted in partial profiling.
TMXStatusCodeInvalidOrgID	This code is returned if an invalid or NULL value is present in the **org_id** calling option.
TMXStatusCodeNotConfigured	This code is returned if the object is not configured properly. You may receive this code if you pass the wrong options to the configure method.
TMXStatusCodeCertificateMismatch	This code is returned when there is a mismatch between the certificate pinned and the certificate used in the host.

APPENDIX B

Code Examples for Mobile Implementation

This appendix includes code examples for Android and iOS mobile implementations.

Android Code Example

Example 14 Android Code

```
protected void onCreate(Bundle savedInstanceState)
{
    Log.d(TAG, "onCreate");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.login_activity);
    //Request required permissions
    requestPermissions();
    /*
     * For ease of use, we do all the view finding now
     */
    m_loginButton  = (Button)      findViewById(R.id.loginButton);
    m_accountText  = (EditText)    findViewById(R.id.accountText);
    m_passwordText = (EditText)    findViewById(R.id.passwordText);
    m_progressBar  = (ProgressBar) findViewById(R.id.progressBar);
    /*
    * And set up the onClick handlers:
    */
    m_loginButton.setOnClickListener(this);
    /*
     * When the device rotates, the activity will be destroyed and created again, although
before
     * destruction, the state of the app will be written into a Bundle. To avoid
unnecessary profiling
     * (i.e. when device rotates), the app checks the bundle and skips profiling if the
device is
     * rotated.
     * */
    if(savedInstanceState == null)
    {
        //Resetting the status of the application to make sure we have a clean activity
        reset();
        /*
         * Optionally, you can configure TMXProfilingConnections.  If so, pass the
configured
```

```
         * instance to TMXConfig. On the other hand, if you prefer to use
TMXProfilingConnection
         * default settings, there is no need to create an instance of it.
         * */
        TMXProfilingConnectionsInterface profilingConnections = new
TMXProfilingConnections()
                .setConnectionTimeout(20, TimeUnit.SECONDS)    // Default value is 10
seconds
                .setRetryTimes(3);                             // Default value is 0 (no
retry)
        /*
         * Creating a config instance to be passed to the init() method. Please note this
instance
         * must include orgId and application context otherwise the init() method will
fail.
         * */
        TMXConfig config = new TMXConfig()
                                    // For more information about configure method please
see kb.threatmetrix.com
                                     // (REQUIRED) Organisation ID
                                     .setOrgId(ORG_ID)
                                     // (REQUIRED) Enhanced fingerprint server
                                     .setFPServer(FP_SERVER)
                                     // (REQUIRED) Application Context
                                     .setContext(getApplicationContext())
                                     // (OPTIONAL) Pass the configured instance of
TMXProfilingConnections to TMX SDK.
                                     // If not passed, init() method tries to create an
instance of TMXProfilingConnections
                                     // with the default settings.
                                     .setProfilingConnections(profilingConnections)
                                     // (OPTIONAL) Set timeout for entire profiling
                                     .setProfileTimeout(20, TimeUnit.SECONDS)
                                     // (OPTIONAL) Register for location services
                                     // Requires ACCESS_FINE_LOCATION or ACCESS_COARSE_
LOCATION permission
                                     .setRegisterForLocationServices(true);
        /*
         * Calling init is mandatory to perform initial setup and requires, at a minimum,
         * the application context and orgId.
         *
         * Only the first call to init() will use the configuration object; subsequent
calls
         * will be ignored. init() can fail due to an illegal argument or state, in which
cases throw
        * exception(s) to draw attention to the programming problem. Failed cases include
malformed
         * organisation id, malformed fingerprint server.
         */
        TMXProfiling.getInstance().init(config);
        //Init was successful or there is a valid instance to be used for further calls.
Fire a profile request
        Log.d(TAG, "Successfully init-ed ");
        doProfile();
    }
}
void doProfile()
{
    // (OPTIONAL) Assign some custom attributes to be included 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. You can use a more complex request,
    // but the minimum works fine for this purpose.
    TMXProfilingHandle profilingHandle = TMXProfiling.getInstance().profile(options,
                                                                            new
CompletionNotifier());
// (REQUIRED) The end notifier must be passed to the profile() method.
// Session id can be collected here:
    Log.d(TAG, "Session id = "+ profilingHandle.getSessionID());
    /*
    * profilingHandle can also be used to cancel this profile if needed
    *
    * profilingHandle.cancel();
    * */
}
private class CompletionNotifier implements TMXEndNotifier
{
    /**
     * This gets called when the profiling has finished.
        * You must be careful here because you will not be called on the UI thread.
        * If you want to update the UI elements, you can only do it from the UI thread.

* If you want to update the UI elements, you 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();
        Log.i("LemonBankCompletion", "Profile completed with: " +
result.getStatus().toString()+ " - " + result.getStatus().getDesc());
        /*
         * Profiling is complete, so login can proceed when the Login button is clicked.
         */
        setProfileFinished(true);
        /*
         * The Login button is clicked before the profiling is finished, therefore we
should login...
         * */
        if(isLoginClicked())
        {
            login();
        }
    }
}
```

iOS Code Examples

Objective-C

Example 15 Objective-C Code

```
- (instancetype) init
{
    self = [super init];
    _sessionID         = nil;
    _profileTimeout    = @20;
    /*
     * Optionally you can configure TMXProfilingConnections. If so, pass the configured
     * instance to configure method. On the other hand, if you prefer to use the
TMXProfilingConnection
     * default settings, there is no need to create an instance of it.
     * */
    TMXProfilingConnections *profilingConnections = [[TMXProfilingConnections alloc]
init];
    profilingConnections.connectionTimeout    = 20; // Default value is 10 seconds
    profilingConnections.connectionRetryCount = 2;  // Default value is 0 (no retry)
    // The profile.configure method is effective only once, and subsequent calls to it
will be ignored.
    // Please note that configure may throw an NSException if the NSDictionary key/
value(s) are invalid.
    // This only happen due to programming error(s); therefore, if you don't catch the
exception, make sure there is no error in your configuration dictionary.
    [[TMXProfiling sharedInstance] configure:@{
                                        // (REQUIRED) Organization ID
                                           TMXOrgID              : ORG_ID,
                                        // (REQUIRED) Enhanced fingerprint server
                                           TMXFingerprintServer  : FP_SERVER,
                                      // (OPTIONAL) Set the profile timeout, in seconds
                                           TMXProfileTimeout     : _profileTimeout,
                                       // (OPTIONAL) If Keychain Access sharing groups
are used, specify it
                                     TMXKeychainAccessGroup: @"<TEAM_ID>.<BUNDLE_ID>",
                                    // (OPTIONAL) Register for location service updates.
                                    // Requires permission to access to device location.
                                           TMXLocationServices   : @YES,
                                      // (OPTIONAL) Pass the configured instance of
TMXProfilingConnections to TMX SDK.
                                     // If not passed, the configure method tries to
create an instance of TMXProfilingConnections
                                        // with the default settings.
                                           TMXProfilingConnectionsInstance :
profilingConnections,
    }];
    return self;
```

```
}
-(void)doProfile
{
    // (OPTIONAL) Assign some custom attributes to be included with the profiling
information
    NSArray *customAttributes = @[@"attribute 1", @"attribute 2"];
    self.loginOkay = NO;
    // Fire off the profiling request.
    TMXProfileHandle *profileHandle = [[TMXProfiling sharedInstance]
profileDeviceUsing:@{TMXCustomAttributes : customAttributes}

callbackBlock:^(NSDictionary * _Nullable result) {
       TMXStatusCode statusCode = [[result valueForKey:TMXProfileStatus] integerValue];
        // If we registered a delegate, this function will be called once the profiling
is complete.
        if(statusCode == TMXStatusCodeOk)
        {
        // No errors, profiling succeeded!
        }
        NSLog(@"Profile completed with: %s and session ID: %@", statusCode ==
TMXStatusCodeOk ? "OK"
              : statusCode == TMXStatusCodeNetworkTimeoutError ? "Timed out"
              : statusCode == TMXStatusCodeConnectionError     ? "Connection Error"
              : statusCode == TMXStatusCodeHostNotFoundError   ? "Host not found error"
              : statusCode == TMXStatusCodeInternalError       ? "Internal Error"
              : statusCode == TMXStatusCodeInterruptedError    ? "Interrupted"
              : "other",
              [result valueForKey:TMXSessionID]);
        [self setLoginOkay:YES];
    }];
    // Session id can be collected here (to use in an API call (AKA session query).
    self.sessionID = profileHandle.sessionID;
    NSLog(@"Session id is %@", self.sessionID);
    /*
     * profileHandle can also be used to cancel this profile if needed
     *
     * [profileHandle cancel];
     * */
}
```

Swift

Example 16 Swift

```
override init()
{
    super.init()
```

```
    /*
     * Optionally you can configure TMXProfilingConnections.  If so, pass the configured
     * instance to configure method. On the other hand, if you prefer to use the
TMXProfilingConnection
     * default settings, there is no need to create an instance of it.
     * */
    let profilingConnections: TMXProfilingConnections  = TMXProfilingConnections.init()
    profilingConnections.connectionTimeout    = 20; // Default value is 10 seconds
    profilingConnections.connectionRetryCount = 2;  // Default value is 0 (no retry)
    //Get a singleton instance of TrustDefenderMobile
    profile = TMXProfiling.sharedInstance()
    // The profile.configure method is effective only once; subsequent calls to it will
be ignored.
    // Please note that configure may throw an NSException if the NSDictionary key/
value(s) are invalid.
    // This only happens due to programming error; therefore, if you don't catch the
exception, make sure there is no error in your configuration dictionary.
    profile.configure(configData:[
                        // (REQUIRED) Organization ID
                        TMXOrgID              :"invalid",
                        // (REQUIRED) Enhanced fingerprint server
                        TMXFingerprintServer  :"enhanced-fp-server",
                        // (OPTIONAL) Set the profile timeout, in seconds
                        TMXProfileTimeout     : profileTimeout,
                       // (OPTIONAL) If Keychain Access sharing groups are used, specify
like this
                        TMXKeychainAccessGroup: "<TEAM_ID>.<BUNDLE_ID>",
                        // (OPTIONAL) Register for location service updates.
                        // Requires permission to access device location
                        TMXLocationServices   : true,
                        // (OPTIONAL) Pass the configured instance of
TMXProfilingConnections to the TMX SDK.
                      // If not passed, the configure method tries to create an instance
of TMXProfilingConnections
                        // with the default settings.
                        TMXProfilingConnectionsInstance:profilingConnections,
                        ])
}
func doProfile()
{
    let customAttributes : [String : Array<String>] = [TMXCustomAttributes: ["attribute
1", "attribute 2"]]
    loginOkay = false
    // Fire off the profiling request.
    let profileHandle: TMXProfileHandle =
profile.profileDevice(profileOptions:customAttributes, callbackBlock:{(result:
[AnyHashable : Any]?) -> Void in
            let results:NSDictionary! = result! as NSDictionary
            let status:TMXStatusCode  = TMXStatusCode(rawValue:(results.value(forKey:
TMXProfileStatus) as! NSNumber).intValue)!
            self.sessionID = results.value(forKey: TMXSessionID) as! String
            if(status == .ok)
            {
                // No errors, profiling succeeded!
            }
        let statusString: String =
            status == .ok                  ? "OK"                  :
            status == .networkTimeoutError ? "Timed out"           :
            status == .connectionError     ? "Connection Error"    :
            status == .hostNotFoundError   ? "Host Not Found Error" :
            status == .internalError       ? "Internal Error"      :
```

```
                    status == .interruptedError    ? "Interrupted Error"    :
                    "Other"
                    print("Profile completed with: \(statusString) and session ID:
            \(self.sessionID)")
                    self.loginOkay = true
                })
                // Session id can be collected here (to use in API call (AKA session query))
                self.sessionID = profileHandle.sessionID;
                print("Session id is \(self.sessionID)");
                /*
                 * profileHandle can also be used to cancel this profile if needed
                 *
                 * profileHandle.cancel()
                 * */
}
```
