Check Point
SOFTWARE TECHNOLOGIES LTD.

18 August 2021

SANDBLAST APP PROTECT MOBILE SDK

Version 3.7

Integration Guide



© 2019 Check Point Software Technologies Ltd.

All rights reserved. This product and related documentation are protected by copyright and distributed under licensing restricting their use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form or by any means without prior written authorization of Check Point. While every precaution has been taken in the preparation of this book, Check Point assumes no responsibility for errors or omissions. This publication and features described herein are subject to change without notice.

RESTRICTED RIGHTS LEGEND:

Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

TRADEMARKS:

Refer to the Copyright page http://www.checkpoint.com/copyright.html for a list of our trademarks.

Refer to the Third Party copyright notices http://www.checkpoint.com/3rd_party_copyright.html for a list of relevant copyrights and third-party licenses.

Important Information

Latest Software

We recommend that you install the most recent software release to stay up-to-date with the latest functional improvements, stability fixes, security enhancements and protection against new and evolving attacks.

Check Point SandBlast Mobile

For more about this release, see the SandBlast Mobile home page https://supportcenter.checkpoint.com/supportcenter/portal?eventSubmit_doShowproductpage&product=491.

Latest Version of this Document

Download the latest version of this document http://downloads.checkpoint.com/dc/download.htm?ID=57703.

Feedback

Check Point is engaged in a continuous effort to improve its documentation.

Please help us by sending your comments mailto:cp_techpub_feedback@checkpoint.com?subject=Feedback on SandBlast Mobile SDK Integration Guide.

Revision History

Date	Description
03 September 2017	First release of this document
04 October 2018	Update iOS Integration for Xcode 10
24 February 2019	Add MiTM and Trusted Certificate to iOS
16 April 2019	Update iOS Integration for Xcode 10.2
12 May 2019	SDK version 3.4
13 May 2019	Update iOS Integration
21 May 2019	Add getVersion API and update iOS Integration
28 May 2019	Add guide for React Native, memory usage requirements
27 August 2019	Added instructions for Android P targeting
7 January 2020	Added a note for Flutter support
20 January 2020	Added documentation for Android advanced features
19 October 2020	Added iOS integration via CocoaPods
17 August 2021	iOS API changes

Contents

Important Information 3
Introduction 5
Technical Overview 5
Authorization Token 5
iOS Integration 6
Prerequisites 6
Supported iOS versions 6
Supported Xcode versions 6
Supported for iOS Simulator 6
Bitcode support 6
Memory Requirements 6
XCode Instructions 7
React Native Support 14
Flutter and Ionic Support 14
Enabling HTTP Requests 15
API Documentation 16
Register 16
Device ID 16
Detect 16
Examples of API's response 19
Additional Classes \ Enums 20
Android Integration 21
Prerequisites 21
Min API Level 21
Foreground Service 21
Memory Requirements 21
HTTP Traffic Requirements 21
Advanced Detection Capabilities 22
Android Studio Instructions 23
React Native Support 24
Flutter Support 24
API Documentation 25
Initialization and Authorization 25
Detection 26
Examples of API's response 29
Work in Background 32
Logs 33
Exceptions 33
Additional Classes 33
Appendix - SDK Detection Capabilities 35

Introduction

Check Point SandBlast Mobile SDK allows organizations to embed its market-leading mobile security solution into their mobile apps.

This document describes the different methods to embed the SDK into Android and iOS apps, and outlines the requirements, authorization and security aspects of the SDK when used inside an organization's apps.

Technical Overview

Authorization Token

An authorization token is provided to SDK users to embed in the initial registration API as described in the sections below. The authorization token identifies the hosting app developer and is valid for both Android and iOS devices.

iOS Integration

Prerequisites

Supported iOS versions

iOS 11 and higher

Supported Xcode versions

Xcode 10.3 and higher

Supported for iOS Simulator

Bitcode support

SBM SDK supports bitcode

Memory Requirements

SBM SDK for iOS might require up to 10MB of memory

XCode Instructions

There are 2 options to integrate the SandBlast Mobile SDK:

1. Via CocoaPods
2. Manual integration

In order to integrate via CocoaPods, you need to add the following lines to your podfile:

```
platform :ios, ’11.0’
inhibit_all_warnings!
use_frameworks!

target :<YOUR_TARGET_NAME> do

    pod ‘sandblast_app_protect’

post_install do |installer|
    installer.pods_project.targets.each do |target|
      if ['SwiftKeychainWrapper', 'AWSS3', 'AWSCore',
'RNCryptor', 'Alamofire', 'SSZipArchive'].include? target.name
        target.build_configurations.each do |config|
          config.build_settings['BUILD_LIBRARY_FOR_DISTRIBUTION']
= 'YES'
        end
      end
    end
  end
end
```

Run ‘pod install’ command and the SDK is ready to be used in your project.

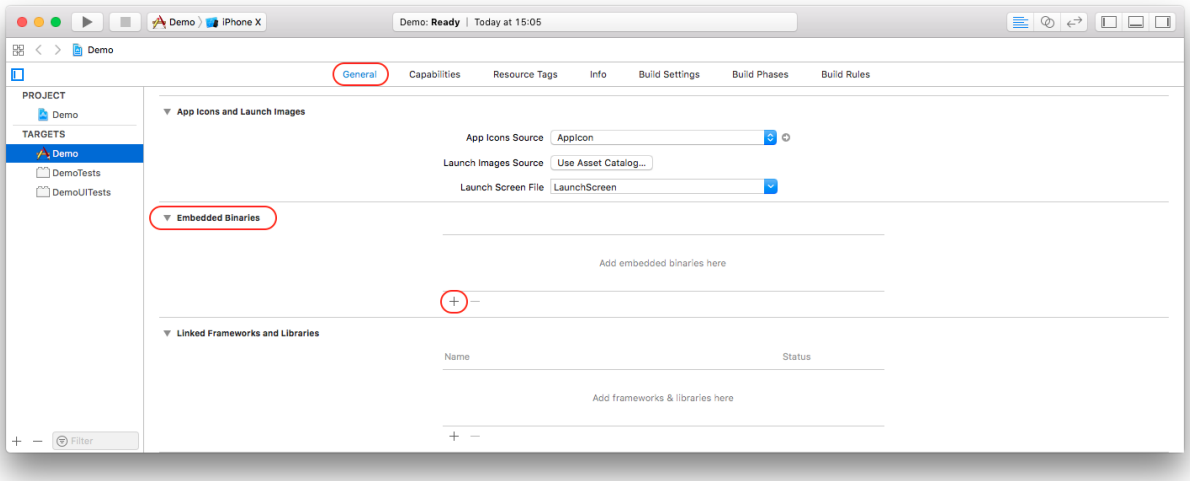
To integrate the SandBlast framework into your XCode project, link the SBM.framework to it.

Using Xcode 10:

To link SBM.framework:

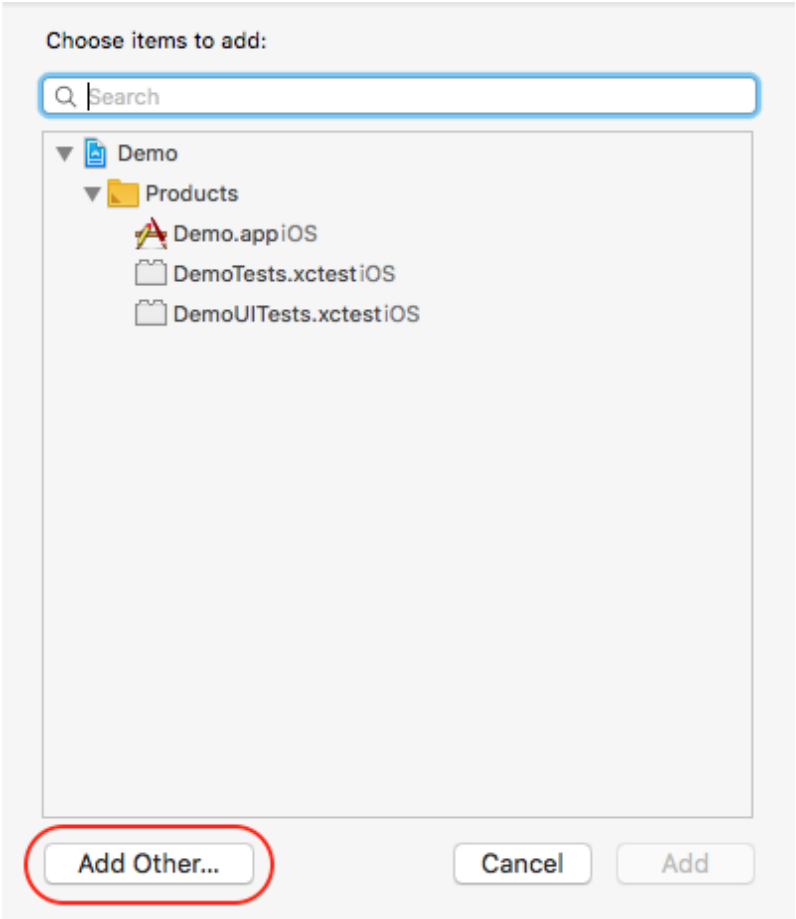
1. Select your project file and make sure that your target is selected.

2. Select **General** > **Embedded Binaries** and click the **+** icon.

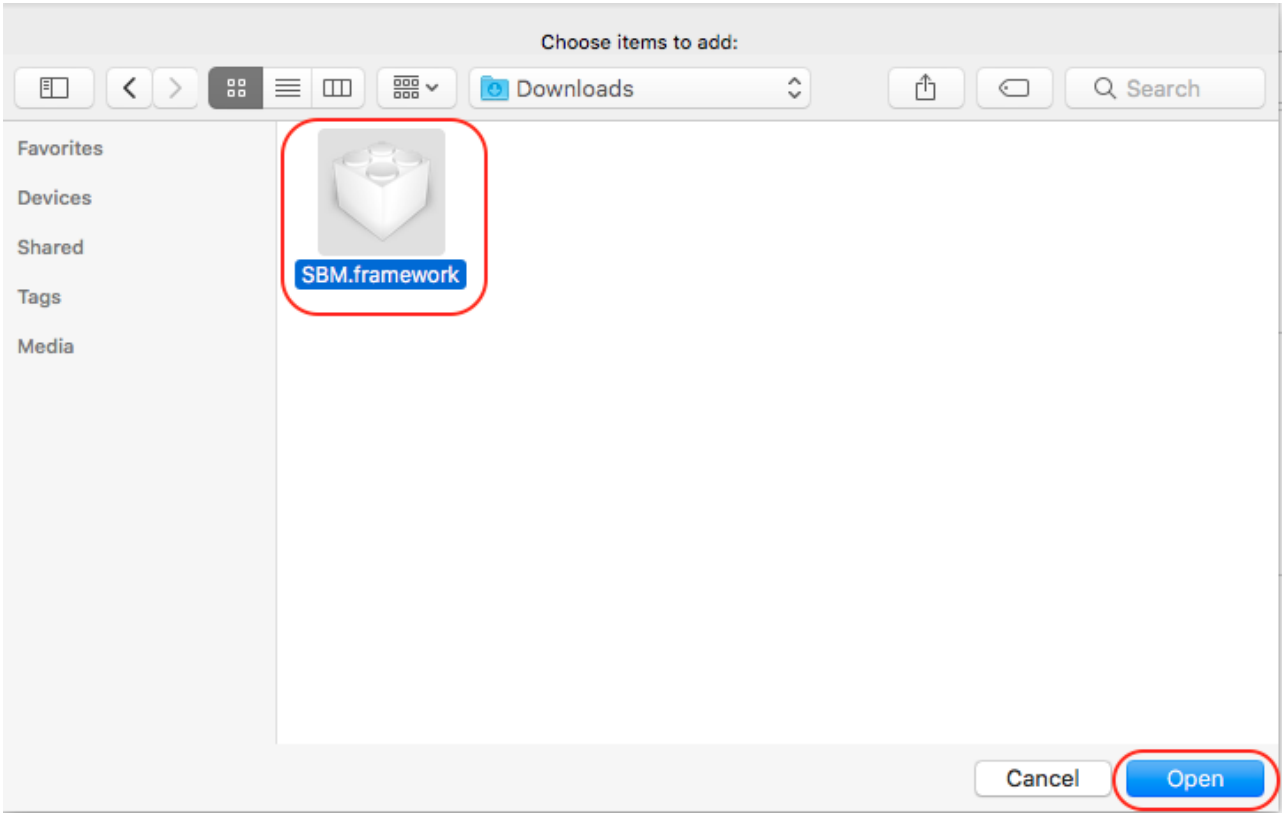


The items window opens.

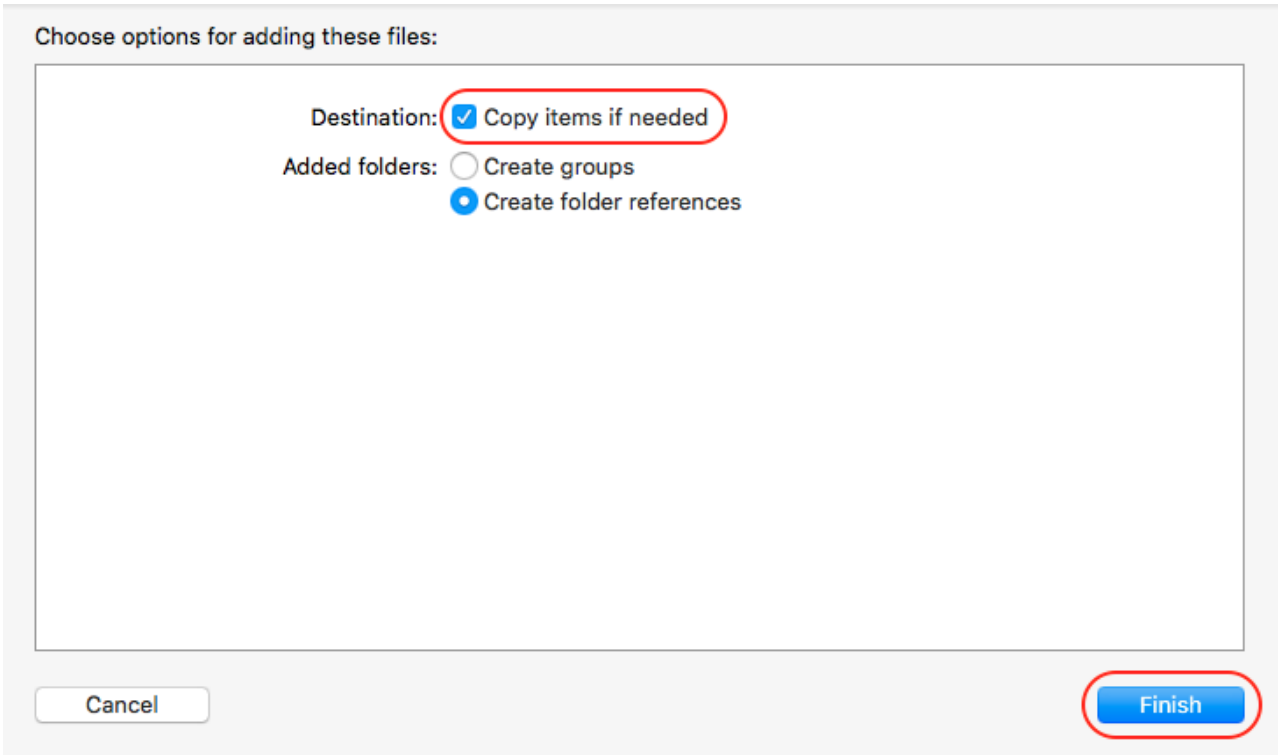
3. Click **Add Other**.



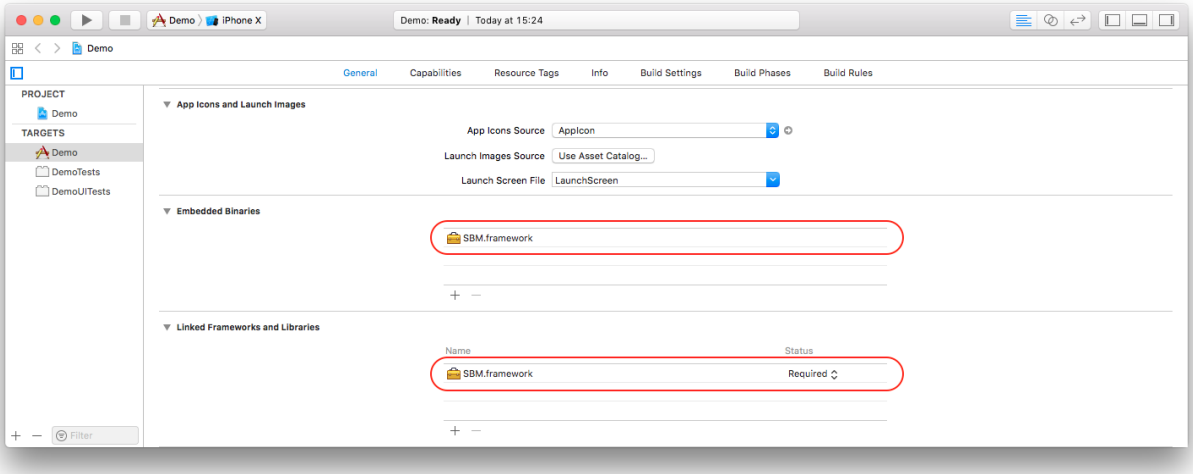
4. Select the SBM.framework package and click **Open**.



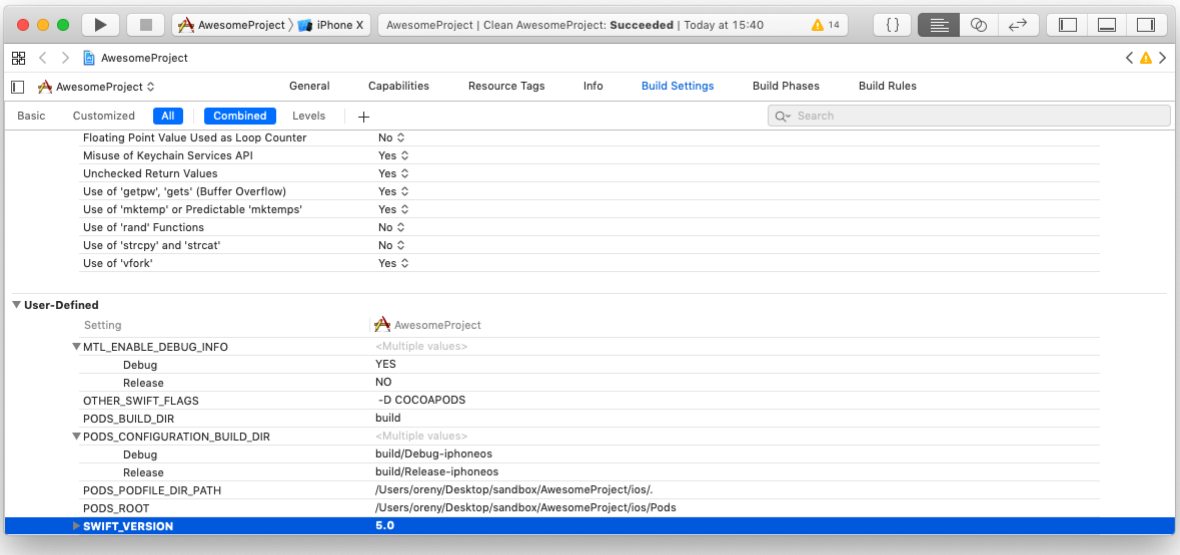
5. Make sure **Copy items if need** is selected and click **Finish**.



6. The SBM.framework now appears in the **Embedded Binaries** and **Linked Frameworks and Libraries**.



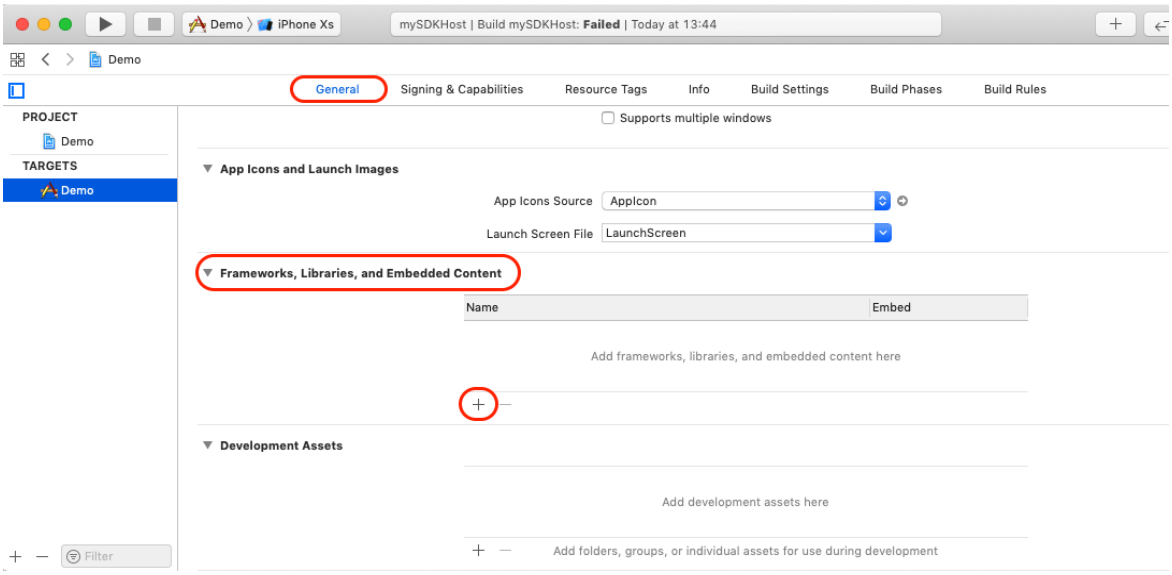
7. If your app uses <u>React Native</u>: select **Build Settings** and scroll to **User-Defined**. Add new key **SWIFT_VERSION** with value **5.0**.



Using Xcode 11:

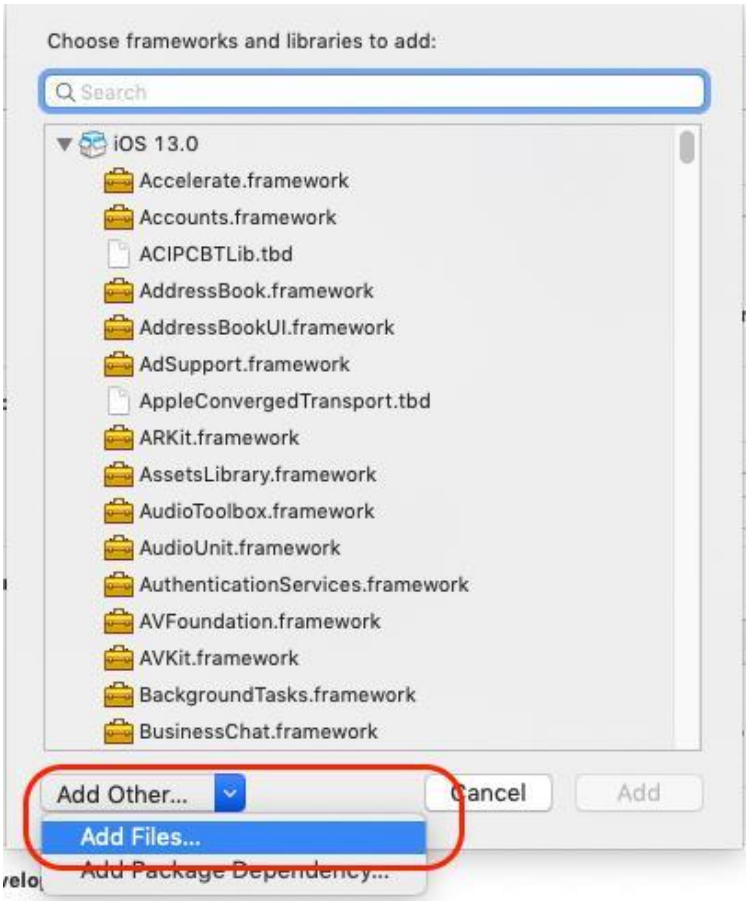
To link SBM.framework:

1. Move the SBM.framework file into the same folder of your Xcode project
2. Select your project file and make sure that your target is selected.
3. Select **General** > **Frameworks, Libraries, and Embedded Content** and click the **+** icon.

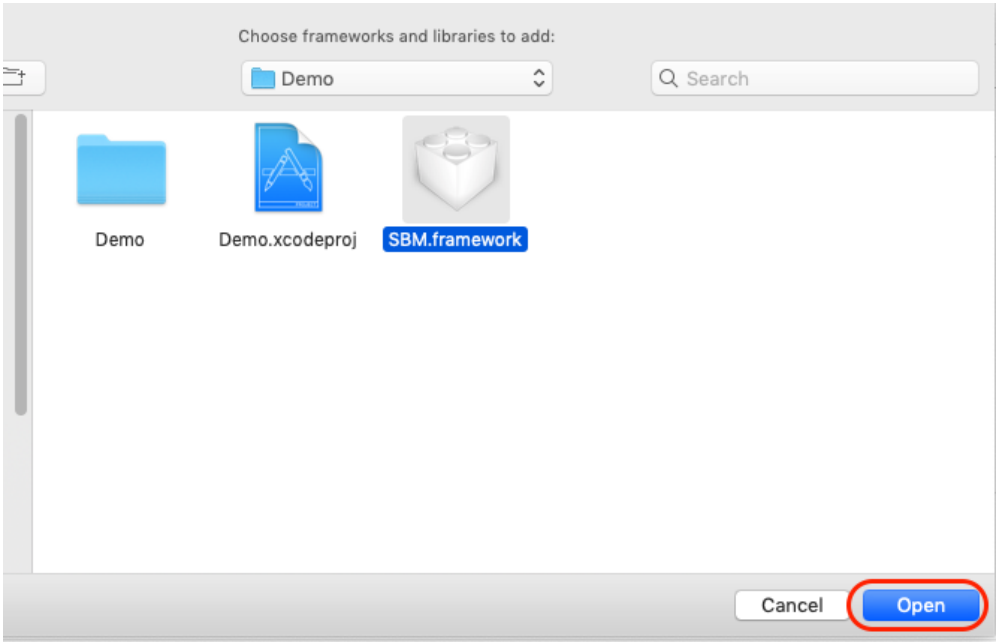


The items window opens.

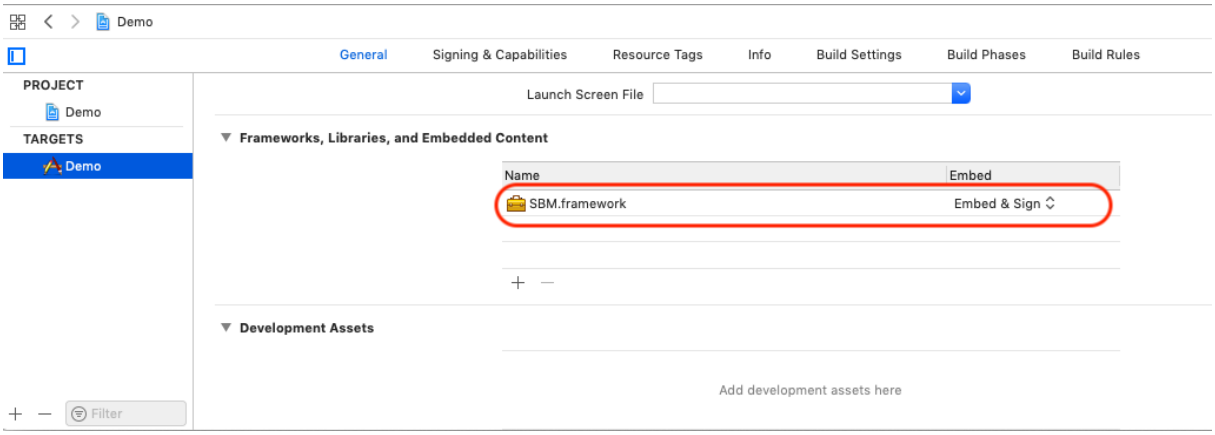
4. Click **Add Other...** and then **Add Files...**.



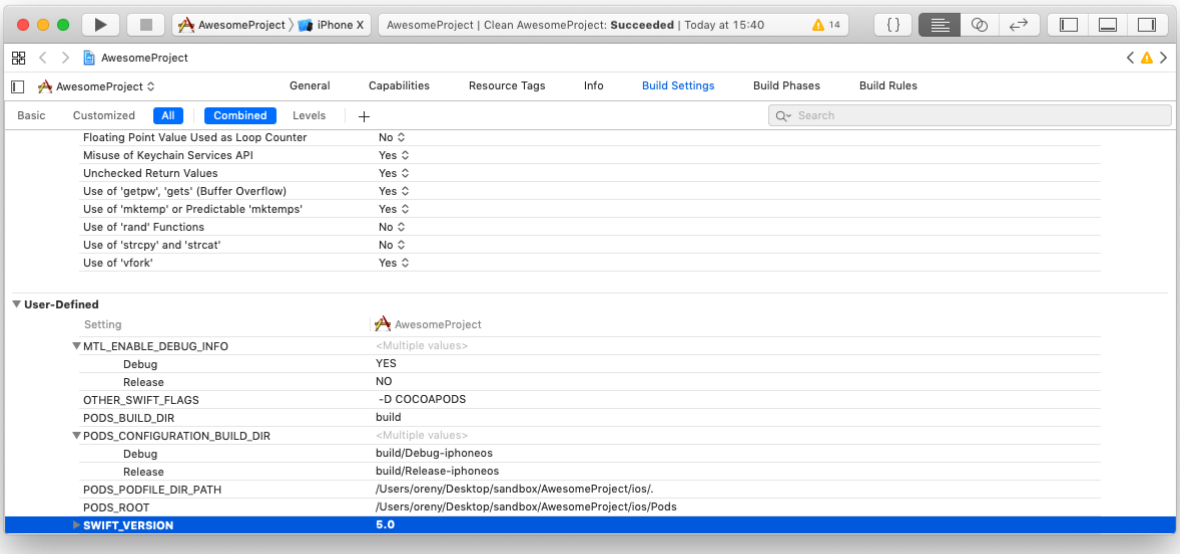
5. Select the SBM.framework package and click **Open**.



6. The SBM.framework now appears in the **Frameworks, Libraries, and Embedded Content** section.



7. If your app uses React Native: select **Build Settings** and scroll to **User-Defined**. Add new key **SWIFT_VERSION** with value **5.0**.



Continue for Xcode 10/11:

8. Add support for **CocoaPods** in your project (see https://cocoapods.org).

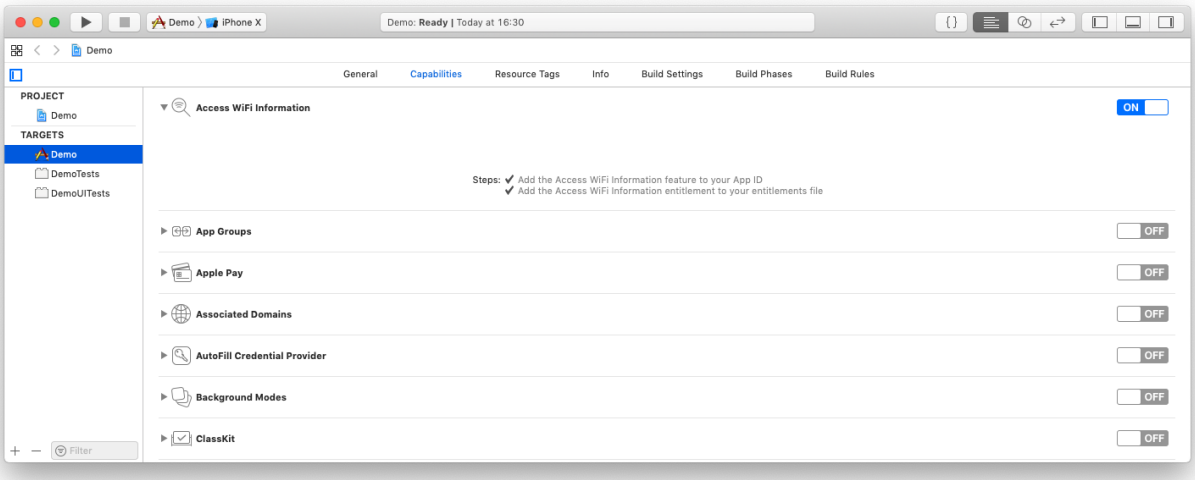
9. Add this line to your **Podfile**.

```
platform :ios, '11.0'
inhibit_all_warnings!
use_frameworks!

def sbm_pods
    pod 'AWSS3', '2.10.0'
    pod 'Alamofire', '4.9.1'
    pod 'SwiftKeychainWrapper', '3.4.0'
    pod 'SSZipArchive', '2.2.2'
    pod 'RNCryptor', '5.1.0'
end

target :<YOUR_TARGET_NAME> do
   sbm_pods
end
```

10. Add “Access WiFi Information” capability to receive Wi-Fi network SSID during MiTM attack.



React Native Support

See step 8 in the previous section (XCode integration).

Flutter and Ionic Support

No need any adjustment.

Enabling HTTP Requests

Network detection detects any network attacks. To run network detection, the SDK must perform a HTTP call which by default is not enabled for apps. The hosting app must add an exception to enable HTTP access to the SandBlast Mobile detection site.

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSExceptionDomains</key>
    <dict>
        <key>bosko.locsec.net</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionAllowsInsecureHTTPLoads</key>
            <true/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <true/>
            <key>NSExceptionMinimumTLSVersion</key>
            <string>TLSv1.2</string>

<key>NSThirdPartyExceptionAllowsInsecureHTTPLoads</key>
            <false/>

<key>NSThirdPartyExceptionRequiresForwardSecrecy</key>
            <true/>
            <key>NSThirdPartyExceptionMinimumTLSVersion</key>
            <string>TLSv1.2</string>
            <key>NSRequiresCertificateTransparency</key>
            <false/>
        </dict>
    </dict>
</dict>
```

API Documentation

SandBlast Mobile SDK is accessible by the *SBM* class.

Register

```
 + (void)initializeWithAPIKey:(NSString * _Nonnull)apiKey
callbackQueue:(NSOperationQueue * _Nonnull)callbackQueue completion:(void
(^_Nonnull)(SBMClientRegistrationResult result))completion;
```

These are the parameters to configure the SDK:

Parameter	Type	Description
apiKey	String	API key for the SDK. Provided by SandBlast Mobile during registration.
callbackQueue	NSOperationQueue	The callback queue which the completion block will be called in.
completion	Block	A completion block to process the registration response.

Device ID

```
+ (NSString * _Nullable)deviceId;
```

Returns the client identifier, a random UUID for this device used in the SBM SDK. This is the only identifier to be used in order to identify SBM SDK in the SDK server backend and it is recommended for the hosting app to save/print it in order to report issues with the SDK.

Detect

```
+ (void)scanFeatures:(NSArray<NSNumber*>* _Nonnull)features
callbackQueue:(NSOperationQueue * _Nonnull)callbackQueue
completion:(void (^_Nonnull)(NSArray<SBMScanResult*> * _Nonnull
scanResults))completion;
```

Parameters:

Parameter	Type	Description
features	NSArray<NSNumber*>	Array of features that should be scanned. contains array of NSNumber objects which represnt SBMClientDetectionType.
callbackQueue	NSOperationQueue	The callback queue which the completion block will be called in.

Parameter	Type	Description
completion	Block	Called at the end of the scan

The method returns the current device risk status.

```
+ (void)getCurrentStatusWithCallbackQueue:(NSOperationQueue *
_Nonnull)callbackQueue completion:(void
(^_Nonnull)(NSDictionary<NSString*, NSString*> * _Nonnull stauts,
NSDate * _Nullable lastSuccessfullScanDate))completion;
```

Parameters:

Parameter	Type	Description
callbackQueue	NSOperationQueue	The callback queue which the completion block will be called in.
completion	Block	Returning mapping of group to description. Related scan date.

Get much more details information regarding the latest status

```
+ (void)getRisksWithCallbackQueue:(NSOperationQueue *
_Nonnull)callbackQueue completion:(void
(^_Nonnull)(NSArray<SBMRisk *> * _Nullable risks, NSDate *
_Nullable lastSuccessfullScanDate))completion;
```

Parameters:

Parameter	Type	Description
callbackQueue	NSOperationQueue	The callback queue which the completion block will be called in.
completion	Block	Contains object representation of the status and the last successful scan date.

LOGs

The method set the logger type.

```
+ (void)setLoggerType:(SBMClientLoggerType)loggerType;
```

Parameters:

Parameter	Type	Description
loggerType	SBMClientLoggerType	The logger type (debug or release)

The method set the logger file handle.

```
+ (void)setLoggerOutput:(NSFileHandle * _Nullable)fileHandle;
```

Parameters:

Parameter	Type	Description
fileHandle	NSFileHandle	The file handle to the log file. The file should be open and ready for write operations. You should set this to nil before you close the file handle

Version

The method returns the SDK version.

```
+ (NSString * _Nonnull)getVersion;
```

Examples of API's response

Here all options to SDK's responses for *getCurrentStatus* and *getDetails* APIs

- *getCurrentStatus* return a dictionary represented in JSON string.
- *getRisks* return an array of 'SBMRisk' objects.

- MiTM attack (SSL stripping , SSL Interception - invalid certificate **or** pinning):
    1. *getCurrentStatus*:
        - {"MITM": "Detected possible man-in-the-middle attack attempt. This attack allows unauthorized 3rd party to gain access to the device\'s network traffic, which may contain sensitive information such as emails, user names, passwords etc."}
    2. *getRisks*:
        - [{"findings" : ["group" : "MITM","threatFactors" : ["MITM_SSL_STRIPING", "MITM_INVALID_CERT", "MITM_CERT_PINNING_FAIL"]}],"metadata" : {"type" : "network", {"value" : "my home" }}]
- Jailbroken device (regular , advanced):
    1. *getCurrentStatus*:
        - ["JAILBROKEN_ROOTED": "The device has been found to be rooted using an advanced logic"]
    2. *getRisks*:
        - [{"findings" : ["group" : "JAILBROKEN_ROOTED ","threatFactors" : ["JAILBROKEN", "JAILBREAK_ADV"]}],"metadata" : {"type" : "jailbreak "}}]
- Enterprise certificate (enterprise , developer):
    1. *getCurrentStatus*:
        - ["SUSPICIOUS_ENTERPRISE_CERTIFICATE": "An enterprise certificate profile is installed on the device, enterprise certificate should only be used for official internal apps. It can be abused by malicious app"]
    2. *getDetails*:
        - [{"findings" : ["group" : "SUSPICIOUS_ENTERPRISE_CERTIFICATE","threatFactors" : ["IOS__DEVELOPER_PROFILE", "IOS__ENTERPRISE_PROFILE"]}],"metadata" : {"type" : "trusted_certificate", "value" : "iPhone Distribution: My Company LTD"}}]

- No risk was detected:
    1. *getCurrentStatus*:
        - Empty dictionary (JSON).
    2. *getDetails*:
        - Empty array (SBMRisks).

- In case there are multiple risks detected:
    - *getCurrentStatus* will contain all key-value pairs according to the list above.
    - *getRisks* will contain all findings according to the list above.

Additional Classes \ Enums

To set the logger level:

```
typedef NS_ENUM(NSInteger, SBMClientLoggerType) {
    SBMClientLoggerTypeDebug,
    SBMClientLoggerTypeRelease
};
```

To determine the enum for the different detection types the SDK can run:

```
typedef NS_ENUM(NSInteger, SBMClientDetectionType) {
    SBMClientDetectionTypeJailbreak,
    SBMClientDetectionTypeTrustedCertificate,
    SBMClientDetectionTypeNetwork
};
```

To represent the result of a registration:

```
typedef NS_ENUM(NSInteger, SBMClientRegistrationResult) {
    SBMClientRegistrationResultUnknown,
    SBMClientRegistrationResultNetworkError,
    SBMClientRegistrationResultInvalidInput,
    SBMClientRegistrationResultSuccess
};
```

To represent the result of a scan request:

```
typedef NS_ENUM(NSInteger, SBMClientScanResult) {
    SBMClientScanResultNotInitialized,
    SBMClientScanResultTimeout,
    SBMClientScanResultFailure,
    SBMClientScanResultFeatureDisabled,
    SBMClientScanResultSuccess
};

@interface SBMScanResult : NSObject {}

@property (nonatomic, readonly) SBMClientDetectionType
detectionType;
@property (nonatomic, readonly) SBMClientScanResult result;

@end
```

Android Integration

Prerequisites

Min API Level

The SDK supports minimum Android API level 16, Android 4.1 (JELLY_BEAN).

Foreground Service

Starting from Android Oreo (API level 26), Android requires apps to run in the foreground in order to perform background services and to get implicit broadcasts.

In order to maximize our detection capabilities, we would like you to run a foreground service. See example in the SDKDemo project.

Memory Requirements

SBM SDK for Android averages 20-50MB of memory usage, while during a scan of a large apk file it might peak up to a maximum of 120MB momentarily depending on the size of the apk.

HTTP Traffic Requirements

If your app targets Android Pie (API level 28) or higher, Android blocks HTTP traffic by default, unless the application explicitly declares Network Security Configuration in *AndroidManifest.xml* that permits HTTP traffic.

While this requirement provides better overall security, SDK's MiTM detection relies upon opening HTTP connections. Therefore, you need to add a Network Security Configuration in *AndroidManifest.xml*, or edit an existing one.

There are two ways to configure Network Security Configuration to permit HTTP traffic (see Android Studio Instructions):

1. Permit all HTTP traffic by default.
2. Permit HTTP traffic to *bosko.locsec.net*.

We recommend the second way since it's much more secure. If you choose to use this (second) configuration and you have custom legacy trust managers for https connections, you need to upgrade each of them to an extended trust manager. For more info, see Android Nougat changes.

Advanced Detection Capabilities

(Added in version 3.6.3.x)

This feature includes the following capabilities:

- Anti-debugging – debugger attach detection
- Anti-emulation – detecting when a host application is being run on an emulator
- Anti-instrumentation – detecting hooking frameworks (e.g. Frida), which usually used to inspect host application logic

When one of these cases is detected, the application will be terminated in order to protect the host application. The feature is automatically enabled once you have initialized the SDK, and there is no way to disable it.

Notice that this is the best way to protect the app from being inspected, simply returning the status via the Java API can easily be patched and thus bypassed.

Please note that this can impose some challenges for debugging your own application during development, in which case you can use one of solutions below:

1. Not to initialize the SDK during the development.
2. We have published an additional (stub) artifact, which you can use during development. i.e. (from build.gradle):

```
dependencies {
 debugImplementation "com.checkpoint.sbm.sdk:stub:$SBM_VERSION"
}
```

One can also refer to SDKDemo example project.

Important note: the stub artifact does not include any detection logic and should be used only for development purposes!

Android Studio Instructions

To integrate SandBlast SDK into your application, you must add Check Point's repository. The repository is password protected, ask your contact person for credentials.

Google repository is required since the SDK uses the android architecture library.

Add this to your module *build.gradle*:

Repository:

```
repositories {
        …
        maven {
            url "https://us-artifactory01.locsec.net/artifactory/sand-blast-mobile-sdk"
            credentials {
                username <YOUR_USER>
                password <YOUR_PASSWORD>
            }
        }
    }
```

Dependencies:

```
implementation("com.checkpoint.sbm.sdk:sbm:<LATEST_VERSION>@aar") {
    transitive = true
}
```

Network Security Configuration – the following instructions are for apps targeting Android Pie (API level 28) or above, see HTTP Traffic Requirements.

1. **AndroidManifest.xml**:

```
<application
        …
        android:networkSecurityConfig="@xml/network_security_config">
        …
</application>
```

2. **res/xml/network_security_config.xml** – one of the following:
   - Permit HTTP traffic to a specific domain (*bosko.locsec.net*):

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
        <domain-config cleartextTrafficPermitted="true">
                <domain includeSubdomains="false">bosko.locsec.net</domain>
                …
        </domain-config>
…
</network-security-config>
```

   - Permit all HTTP traffic:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
        <base-config cleartextTrafficPermitted="true">
                <trust-anchors>
                        <certificates src="system" />
                </trust-anchors>
                …
        </base-config>
```

…
</network-security-config>

React Native Support

No special requirements are needed for React Native integration

- Open the android project part in Android-Studio.
- In build.gradle add the dependency + us-artifactory repository.
- In Application object init the SDK.
- For any other JavaScript interaction with SDK is like the official RN documentation at https://facebook.github.io/react-native/docs/native-modules-android

Flutter Support

No special requirements are needed for Flutter integration

- Open the android project part in Android-Studio.
- In build.gradle add the dependency + us-artifactory repository.
- In Application object init the SDK.
- Any other Android platform-specific interaction with the SDK can be done using a well-known methods, as per official Flutter documentation at https://flutter.dev/docs/development/platform-integration/platform-channels

API Documentation

SandBlast Mobile SDK monitors the device risks at all time to ensure that threat detection is as accurate as possible. It uses broadcasts to inform the hosting app of any status.

You can access the SDK API through a *SBMClient* class. You must obtain an instance of *SBMClient* on your application's onCreate method, but the initialization can be done later.

```
public class MyApplication extends Application {

    private SBMClient sandbladClient;

    @Override
    public void onCreate() {
        super.onCreate();
        sandbladClient = new SBMClient.Builder(
        this, "my-api-
key").setAuthorizationCallback(myCallback).build();
    }
}
```

The instance exposes several methods:

```
// Broadcast constants
public static final String ACTION_FIRST_SCAN;
public static final String ACTION_STATUS_CHANGED;
public static final String ACTION_APP_INSTALLATION_CHANGE;

public boolean isFirstScanCompleted()

public void initialize()

public Map<String, String> getCurrentStatus()

public JSONArray getDetails()

public void scan(SBMScanCallback callback, int timeout, @Nullable
TimeUnit timeUnit, @SBMDetectionType.DetectionType int...
detectionTypes)

public void scanFile(@NonNull SBMFileScanCallback callback,
@NonNull String path)

public void setForegroundServiceRunning(boolean running)

public String getDeviceId()

public String getVersion()
```

Initialization and Authorization

You should always initialize the SDK at application's startup and wait for authorization to complete before calling the various APIs.

The result of the authorization process is sent to the callback. On the callback, you can see if the authorization was successful. If not, you must query the event result to see what went wrong and how to fix it. To retry authorization after failure (due to general *AuthorizationFailure* or *MissingPermissions*), recall the *initialize* method after you resolve the issues. ***Currently no permissions are required but we might add then in future releases of the SDK***.

The ***Builder*** method requires these parameters:

Parameter	Type	Required	Description
context	Context	Yes	Android context
apiKey	String	Yes	API key for the SDK. Provided by SandBlast during registration.
callback	SBMAuthorizationCallback	No	Callback to receive authorization result.

Detection

When the SDK is authorized and ready, it runs background detections and monitors changes, such as when a new app is installed, to inform the hosting app of any supported risk. At any time, you can query the *SBMClient* for the current status of the device risk or to perform a scan for a specific detection type.

When the SDK completes the first-time scan of the device it broadcasts an *ACTION_FIRST_SCAN* action so the hosting app can rely on the results. Afterward, whenever the SDK finds any risk changes, it broadcasts *ACTION_STATUS_CHANGED* to inform the hosting app of the changes. When it receives any of these broadcasts the hosting app should ask the SDK for the new status and decide what to do.

```
/**
 * Get all the threats that were found on the device aggregated
into groups.
 *
 * @return Current device risk status aggregated into groups with
summaries.
 * If the first scan was not completed, this result is not fully
reliable.
 * You should wait for the first scan to complete or to run scan
on the
 * detections  you need.
 * Only after the detection finished, query the status using this
method.
 * @throws IllegalStateException If tried to be executed before
the client
 * is authorized
 */
public Map<String,String> getCurrentStatus() throws
IllegalStateException;
```

getCurrentStatus returns the current risk assessment of the device. The returned *Map* object contains the relevant aggregated group names as keys and additional text information of why it was found as values. The additional information should NOT be used by the developer for any reason except for logging.

```
/**
 * Get all the threats that were found on the device with full
details.
 *
 * @return Current device risk status with full details as a
JSONArray of findings
 * and metadata. If the first scan was not completed, this
result is not fully
 * reliable.
 * You should wait for the first scan to complete or to run scan
on the detections
 * you need,
 * and only after the detection finished, query the status using
this method.
 * @throws IllegalStateException If tried to be executed before
the client is
 * authorized
 */
public JSONArray getDetails()
```

getDetails is similar to the *getCurrentStatus* but returns a lot more details on the actual threat.

```
/**
* On Android OS versions greater or equal to 8 (Android O, api
level 26) we need * a foreground service running in order to use
services in the background and to * receive events in our
broadcast receivers to provide the best security coverage.
*
* @param running boolean state of the foreground services that
is managed by the * hosted app. Set to true when the service is
running and to false when destroyed.
*/

public void setForegroundServiceRunning(boolean running)
```

setForegroundServiceRunning is used to run in foreground. Need in Android O and higher.

```
/**
* Get the client identifier (available after initialization)
*
* @return the client device id generated by the sdk
*/
```

```
public String getDeviceId()
```

getDeviceId returns the client identifier, a random UUID for this device used in the SBM SDK. This is the only identifier to be used in order to identify SBM SDK in the SDK server backend and it is recommended for the hosting app to save/print it in order to report issues with the SDK.

```
/**
* Get the SDK version identifier
*
* @return SBM SDK version number
*/

public String getVersion()
```

getVersion returns the SBM SDK version number, e.g. “3.4.0.266”.

Examples of API's response

Here are options to SDK's responses for *getCurrentStatus* and *getDetails* APIs.

See MiTM example in iOS section

- Malware app:
  1. *getCurrentStatus*:

     "MALWARE" -> "The application is a malware"

  2. *getDetails*:

     [{"findings":[{"group":"MALWARE","threatFactors":["GENERIC_MALWARE"]}],"metadata":{"appName":"Test Virus","packageName":"com.androidantivirus.testvirus","type":"app"}}]

- Rooted device:
  1. *getCurrentStatus*:

     "JAILBROKEN_ROOTED" -> "Device is Jailbroken"

  2. *getDetails*:

     [{"findings":[{"group":"JAILBROKEN_ROOTED","threatFactors":["ROOTED"]}],"metadata":{"type":"root"}}]

- Various device settings (properties):
  1. *getCurrentStatus*:

     "OS_INTEGRITY_COMPROMISED" -> "Device OS is compromised."

  2. *getDetails*:

[{"findings":
[{"group":"OS_INTEGRITY_COMPROMISED","threatFactors":["ADB_ENABLED"]}],
"metadata":{"type":" device_settings"}}]

Available device settings threat factors: ADB_ENABLED, UNKNOWN_SOURCES, NO_SCREEN_LOCK_PROTECTION, SECURITY_PATCH__NOT_UPDATED, DATA__NOT_ENCRYPTED

Here is a general example of the getDetails return value:

```
{
    "metadata": {
        "type": "app/network/property"
        //More properties regarding the threat
    },
    "findings": [
        {
            "group": "GROUP_NAME",
```

```
            "threatFactors": ["TF1","TF2"] //TFs linked to this
group
        }
    ]
}
```

```
/**
 * Checks if first scan of the device threats is completed.
before it is, the result
 *  of getCurrentStatus will
 * not reflect the real and full state of the device.
 *
 * @return true if the first scan of the device threat is
finished.
 * @throws IllegalStateException if tried to be executed before
the client is
 *  initialized
 */
public boolean isFirstScanCompleted() throws
IllegalStateException;
```

isFirstScanCompleted - When the SDK completes initialization, it runs a full scan for all the valid detection types. Until the scan is completed, the device risk status is only partially visible (depending on the scans that were finished). To check if the initial scan is completed and the risk status is fully updated, the hosting app should call this method.

As soon as the first scan is completed (depends on the number of apps installed on device and network connectivity but may take up to a few minutes) the hosting app can rely on the device risk status at all times because the SDK continues to monitor changes. If the first scan is not yet complete, the hosting app should listen to the *ACTION_FIRST_SCAN* broadcast for notification when it is complete.

```
/**
 * Performing specific detections with a callback.
 *
 * @param callback Callback to be called when the detections are
completed,
 * or the timeout expires
 * @param timeout How long to wait for result. If passed timeout
is not
 * positive, there will be no timeout.
 * @param timeUnit The time unit of the timeout. @see
 * java.util.concurrent.{@link TimeUnit}. If passed null, there
will be no timeout.
 * @param detectionTypes The detections you want to run. Empty
list means all the
 * detections your eligible to.
 * @throws IllegalStateException If tried to be executed with an
unknown detection
```

```
 * type
 * @throws SBMMissingPermissionException If permissions are
missing for the
 * requested detections
 */
public void scan(SBMScanCallback callback, int timeout, @Nullable
TimeUnit timeUnit, @SBMDetectionType.DetectionType int...
detectionTypes) throws SBMMissingPermissionException,
IllegalStateException
```

scan - In addition to constantly monitoring the device for changes, the SDK can also be triggered to rerun scans for specific detection types. This should only be used before running a highly sensitive operation and not as a default use-case for the SDK. The result of the triggered scan is saved to the device current status (it also triggers an *ACTION_STATUS_CHANGED* broadcast if there is a change), and the callback for the method receives detailed information on which detections were completed and which failed due to timeout or another reason.

Parameter	Type	Required	Description
callback	SBMScanCallback	No	Callback for receiving the detection result.
timeout	Int	Yes	Timeout for the detection process. If the time passed, the callback will be called and indication will pass for each detection status. It usually takes between a couple of seconds to a few tens of seconds to complete a scan depending on the network connectivity, number of apps and time from last scan.
timeUnit	TimeUnit	Yes	
detectionTypes	SBMDetectionType ...	No	The detections you want to run. You can specify which detections you want, but if none is passed will run all detections.

```
/**
 * Scan apk file on device
 *
 * @param callback Callback to be called when the file scan is
completed.
```

```
 * @param path Path to an apk file
 */
public void scanFile(@NonNull SBMFileScanCallback callback,
@NonNull String path)
```

scanFile – Scan apk file according to its path on the device. You need to handle the external storage permission approval before calling this API.

This detection is stateless and does not affect device risk. It might be used to scan apps pre installation.

Work in Background

There a few APIs for configuring how the SDK should work in the background. By default, the SDK receives events from the OS when an app is installed/updated/removed and initiate a scan to check if the risk was changed on the device. In addition, it performs scheduled scans that keep the device up to date. You can control this behavior using the following APIs and decide if it should trigger scans in the background.

```
/**
 * Set the detection settings for apps detection type. Default
is running in the
 * background and handle new/updated/removed apps events. Also
run periodic scans
 * for malware.
 * @param settings com.sandblast.sdk.{@link
SBMDetectionSettings}
 */
public void
setAppsDetectionSettings(@SBMDetectionSettings.Settings int
settings)

/**
 * Set the detection settings for device settings detection
type. Default is
 * performing periodic scans.
 * @param settings com.sandblast.sdk.{@link
SBMDetectionSettings}
 */
public void
setDeviceDetectionSettings(@SBMDetectionSettings.Settings int
settings)
```

SBMDetectionSettings - Contains various settings for the SDK such as running in the background.

- BACKGROUND_ALL – default, run scans in background.
- BACKGROUND_OFF – do not run periodic scans and do not trigger scan when an app is installed/updated or removed.
- BACKGROUND_SCHEDULED – run periodic scans in the background and do not listen to app installation events or network changes

- BACKGROUND_EVENTS – Do not run scheduled scans. Run scan when only when something changes, like app installation, network change.

Logs

SBMLoggerCallback interface needs to be implemented and set in the SBMClient.Builder setLoggerCallback() class. This will enable the hosting app to integrate the SDK logs into their own log handling mechanism. Setting the builder in debug mode will cause the logger to print debug logs.

Exceptions

- *IllegalArgumentException* is thrown if a required parameter is not set.
- *IllegalStateException* is thrown if the client tries to perform a scan with an invalid detection type
- *SBMIllegalAccessException* is thrown if the client tries to perform detections that are not permitted by the configuration.
- *SBMMissingPermissionException* is thrown if permissions are missing for the requested detections. *getMissingPermission()* is the method to retrieve the missing permission name.

Additional Classes

Constants

There are several detection types you can run on the SDK:

```
public final class SBMDetectionType {
     public static final int DETECTION_ROOT = 1;
     public static final int DETECTION_APP = 2;
     public static final int DETECTION_NETWORK = 3;
     public static final int DETECTION_DEVICE_SETTINGS = 4;

}
```

Listeners

SBMAuthorizationCallback interface

```
/**
 * Callback method that will be called when the authorization
process is finished
 *
 * @param success Indication if the authorization completed with
success.
 * You can also check the eventResult type equals to
 * AUTHORIZATION_SUCCESS_EVENT_TYPE
 * @param eventResult Contains additional data on the result. If
authorization
 * fails, it contains information that can help you to resolve
it.
 */
void onAuthorizationCompleted(boolean success, SBMEventResult
eventResult);
```

SBMScanCallback interface

```
/**
 * The requested detections completed.
 *
 * @param scanResults List of the requested detections with
indication of which
 * of whom completed with success, and which otherwise. @see
{@link SBMScanResult}
 */
void onScanCompleted(List<SBMScanResult> scanResults)
```

Event Results

SBMEventResult is passed as an extra method on the *ACTION_AUTHORIZATION* broadcast. It contains additional information on the authorization process. If the authorization result is not successful, the hosting app can retry the authorization process by recalling the *SBMClient.initalialize* method after fixing the required issues.

The possible types of results:

1. AUTHORIZATION_SUCCESS_EVENT_TYPE – Authorization is complete. Starting background detection. No action is needed.
2. AUTHORIZATION_FAILURE_EVENT_TYPE - Authorization process failed. Look at the message for additional information. It may be due to network issues, invalid API key, etc.
3. MISSING_PERMISSIONS_EVENT_TYPE – Unable to run the first scan. There are missing permissions for some of the needed detections.

SBMScanResult contains the status of all the requested detections, if it was completed, failed or timed out.

Possible results:

1. SUCCESS - detection finished successfully
2. FAILURE - detection failed by internal reason
3. TIMEOUT - detection did not finished or failed during the time frame.
4. NOT_SUPPORTED - detection is not supported
5. UNAUTHORIZED - unauthorized user error. Used if tried to be executed before the client is initialized

Appendix - SDK Detection Capabilities

The SBM SDK has a large range of detection capabilities both for app classification and device security configuration.

The full list of detection capabilities are listed in Table A.

Column A - Detection capability
Column B - App or Device indicator.
Column C - Detection Category (Check Point Suggestion)
Column D - Category description

The SDK can be configured to expose ALL or only partial list of the detection capabilities, this is done using a category layer that can be configure per hosting app.

The category layer groups several detection capabilities into one category for providing a simpler consolidate interface to the hosting app.

Based on Check Point security experience we have created a default recommendation for the SDK Category layer with the following categories in column (C). In case a different configuration is needed please submit Table A with the desire category configuration.

Suggested Categories
JAILBROKEN_ROOTED – device sandbox is compromised
MALWARE – App is identified as malicious
AD_MALWARE - App is identified as ad-network with dangerous capabilities
SUSPICIOUS_APP - App is suspected as malicious
MITM – network man-in-the-middle attack
OS_INTEGRITY_COMPROMISED – Device OS is compromised.
DEVICE_COMPROMISED – Device configuration enables malicious behavior.
SUSPICIOUS_ENTERPRISE_CERTIFICATE – iOS only. A certificate profile has been installed allowing the installing of apps on device from unknown sources.

Detection Capabilities (A)	APP/ Device (B)	SDK category layer (C)	Platform	Description (D)
DATA__NOT_ENCRYPTED	Device	DEVICE_COMPROMISED	Android	Device configuration enables application malicious behavior.
NO_SCREEN_LOCK_PROTECTION	Device	DEVICE_COMPROMISED	Android/ iOS	Device configuration enables application malicious behavior.
SECURITY_PATCH__NOT_UPDATED	Device	DEVICE_COMPROMISED	Android	Device configuration enables application malicious behavior.
UNKNOWN_SOURCES	Device	DEVICE_COMPROMISED	Android	Device configuration enables application malicious behavior.
ADB_ENABLED	Device	DEVICE_COMPROMISED	Android	Device configuration enables application malicious behavior.
IOS__DEVELOPER_PROFILE	Device	SUSPICIOUS_ENTERPRISE_CERTIFICATE	iOS	An enterprise certificate profile is installed on the device, enterprise certificate should only be used for official internal apps. It can be abused by malicious app
IOS__ENTERPRISE_PROFILE	Device	SUSPICIOUS_ENTERPRISE_CERTIFICATE	iOS	An enterprise certificate profile is installed on the device, enterprise certificate should only be used for official internal apps. It can be abused by malicious app
AUTO_CLICKER	App	AD_MALWARE	Android	Advert malware found on device
ROUGH_ADNETWORK	App	AD_MALWARE	Android	Advert malware found on device
BANKER	App	MALWARE	Android	Malware found on device.
BOTNET	App	MALWARE	Android	Malware found on device.
DROPPER	App	MALWARE	Android	Malware found on device.
FAKE_APP	App	MALWARE	Android	Malware found on device.
GENERIC_MALWARE	App	MALWARE	Android	Malware found on device.
HACKING_TOOL	App	MALWARE	Android	Malware found on device.
INFO_STEALER	App	MALWARE	Android	Malware found on device.
MITM_APP	App	MALWARE	Android	Malware found on device.
MRAT	App	MALWARE	Android	Malware found on device.
PHISHING_APP	App	MALWARE	Android	Malware found on device.
PREMIUM_DAILER	App	MALWARE	Android	Malware found on device.

Detection Capabilities (A)	APP/ Device (B)	SDK category layer (C)	Platform	Description (D)
RANSOMWARE	App	MALWARE	Android	Malware found on device.
ROOTKIT	App	MALWARE	Android	Malware found on device.
SMSBOT	App	MALWARE	Android	Malware found on device.
DANGEROUS_APP	App	SUSPICIOUS_APP	Android	An application suspected of malware-like activity has been detected.
GENERIC_MALWARE__SUSPECTED	App	SUSPICIOUS_APP	Android	An application suspected of malware-like activity has been detected.
PHONE_TRACKING_TOOL	App	SUSPICIOUS_APP	Android	An application suspected of malware-like activity has been detected.
REMOTE_ACCESS_TOOL	App	SUSPICIOUS_APP	Android	An application suspected of malware-like activity has been detected.
LOCATION_TRACKING_TOOL	App	SUSPICIOUS_APP	Android	An application suspected of malware-like activity has been detected.
ROOTING_TOOL	App	SUSPICIOUS_APP	Android	An application suspected of malware-like activity has been detected.
SUSPICIOUS_APP	App	SUSPICIOUS_APP	Android	An application suspected of malware-like activity has been detected.
MITM_ARP_POISONING	Device	MITM	Android	Possible man in the middle attack was detected.
MITM_CERT_PINNING_FAIL	Device	MITM	Android/ iOS	Possible man in the middle attack was detected.
MITM_INVALID_CERT	Device	MITM	Android/ iOS	Possible man in the middle attack was detected.
MITM_SSL_STRIPING	Device	MITM	Android/ iOS	Possible man in the middle attack was detected.
DM_VERITY__DISABLED	Device	OS_INTEGRITY_COMPROMISED	Android	Operating system presets may lead to application malicious behavior.
JAILBROKEN	Device	OS_INTEGRITY_COMPROMISED	iOS	Operating system presets may lead to application malicious behavior.
ROOTED	Device	OS_INTEGRITY_COMPROMISED	Android	Operating system presets may lead to application malicious behavior.
ROOTED__NON_COMMUNITY	Device	OS_INTEGRITY_COMPROMISED	Android	Operating system presets may lead to application malicious behavior.

Detection Capabilities (A)	**APP/ Device** (B)	**SDK category layer** (C)	**Platform** 	**Description** (D)
SELINUX__NON_ENFORCING	Device	OS_INTEGRITY_COMPROMISED	Android	Operating system presets may lead to application malicious behavior.