Zimperium In-App Protection (zIAP)

Product Overview

Contents

Zimperium In-App Protection (zIAP) Overview

zIAP (In-App Protection) is a comprehensive software development kit (SDK) designed to provide third party Apple iOS and Google Android application developers an easy way to integrate real-time threat detection and adaptive threat protection natively into their applications.

Zimperium offers application developers two integration options:

1. **Embedded zIAP** is a standalone library that contains all of the real-time monitoring and detection code natively.
2. **Linked zIAP** is a lightweight library that is designed for scenarios where the device already has the zIPS solution installed. The Linked zIAP solution will proxy messages from the zIPS agent to the Host Application through AIDL (Android Interface Definition Language).

1 – Embedded zIAP Architecture / Framework

Embedded zIAP is comprised of the following components:

1. **Host Application**
2. **Threat Processor**
3. **Threat Detection Engine**

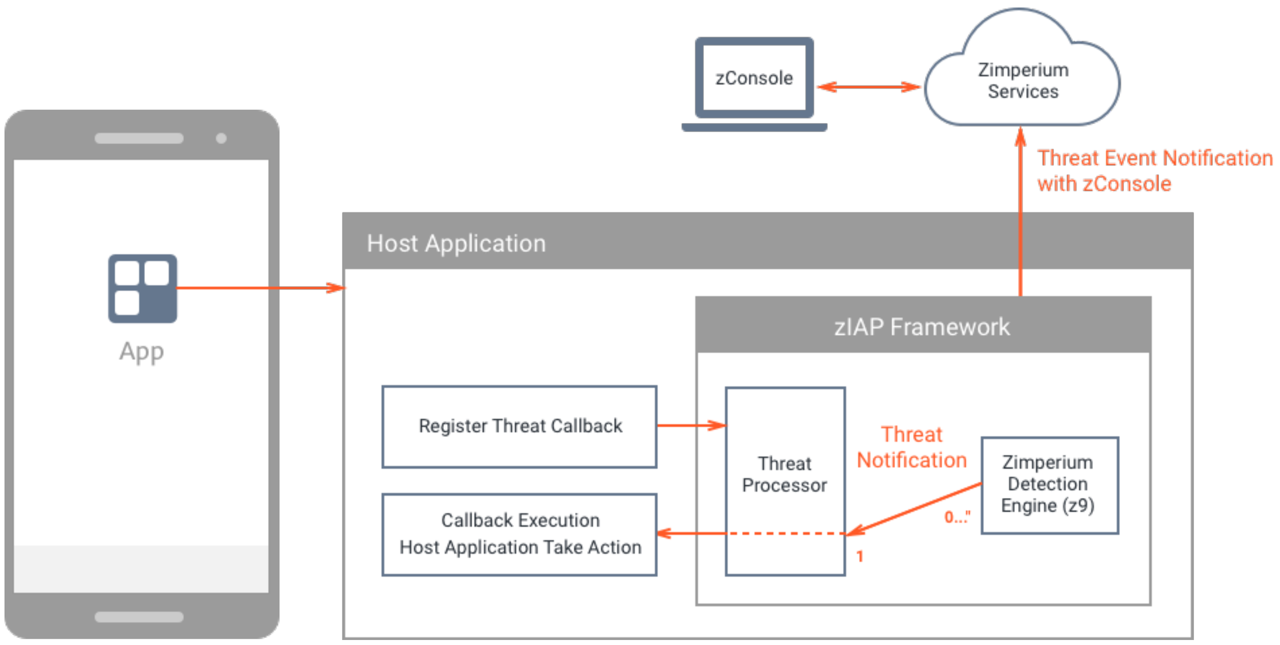


Figure 1: Architecture for Embedded zIAP

1.1 – Host Application

The host application is the native mobile application that will embed the zIAP SDK to detect threats on the device in real time and to apply actions to protect its user or data. The host application uses registered callbacks to get notifications of the status of the device (for example exploit or malware installation) and can apply appropriate actions depending on the type of threat. Alternatively, zIAP can provide an interface to perform on-demand checks of the device's state (for example, is the device rooted, or has the device been jailbroken).

Example uses cases include:

1. A banking application that would like to ensure that the communication(s) between the mobile application and a remote server are not being tampered with or intercepted by an attacker.
2. A payment application that will wipe a user's data from the mobile device when and if malware attempts to compromise the user's credentials or credit card details from within the application.

1.2 – Zimperium Detection Engine (z9)

At the core, zIAP uses Zimperium's z9 detection engine, the same engine found in our award winning zIPS product. z9 is a cyberattack detection engine that uses machine-learning to dynamically detect advanced network, device and application attacks on mobile devices.

The detection engine monitors several parameters and events on the mobile device in order to detect threats in real time. When z9 detects a threat on the device, the host application will receive a notification. Additionally, zIAP will send the event to the cloud with additional forensics on each device running zIAP for offline inspection by the IT Admin on zConsole.

1.3 – Threat Processor

The main function of this component is to process each threat detected by z9 and expose it to the host application on a high-level interface (obj-c or Java). It will provide high level information of the threat and the context of the attack to the host application.

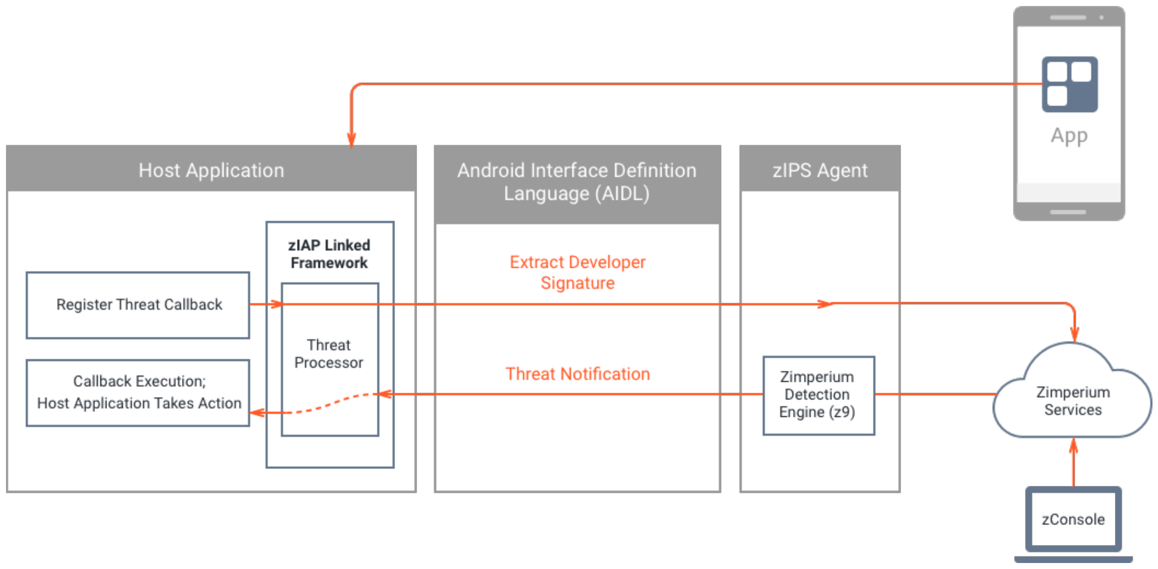
2 – Linked zIAP (Android only)

In the Figure 1 above, both the threat processor and detection engine are hosted on the same process. With the Linked zIAP configuration, only the **Threat Processor** inside the **Host Application** and AIDL (Android Interface Definition Language) are used to securely notify threats to an application.

It is important to note that the set of APIs are exactly the same when using **Embedded zIAP** and **Linked zIAP**. The only difference is the library that needs to be added to the application project.

In both cases the API is a high-level interface (described in later sections of this document). Developers do not need to deal with the low-level AIDL-related code. zIPS will only allow registered developers to exchange messages through AIDL by checking the developer public key. The list of valid developer public keys is updated dynamically from zCloud.

NOTE – Linked zIAP is currently supported on Android OS. Because of OS level restrictions, Linked zIAP is currently not supported on iOS.



App
Host Application
Android Interface Definition Language (AIDL)
zIPS Agent
zIAP Linked Framework
Register Threat Callback
Extract Developer Signature
Threat Processor
Threat Notification
Callback Execution; Host Application Takes Action
Zimperium Detection Engine (z9)
Zimperium Services
zConsole

Figure 2: simplified diagram of the Linked zIAP architecture

3 – API overview

This section describes the functionality and features of the zIAP Framework.

1. **Check rooted / jailbroken device:** The caller is able to check if the device is rooted / jailbroken at any time.
2. **Scan for malware (Android only):** The host application is able to scan all installed applications on the device at any time. Please note that zIAP can listen for installed application events and scan them without any user interaction and notify the host application if malware is installed on the device without initiating a manual scan.
3. **SSL connection:** The host application can check at any time if an attacker in the middle of the communication is trying to compromise a secure SSL connection.
4. **Callback registration:** The host application can register a callback to handle every threat reported by zIAP. Additionally, the callback can be specified by threat type (network, device, or application threats). When a threat is detected, zIAP will call the specified handler providing an object with contextual information of the threat. The following information is accessible to the handler when zIAP detects a threat:

Name	Type	Required	Description
attack_time	string	YES	Date and time of the detection in ISO 8601 format (example : “2014-06-06T12:21:29Z”)
name	string	YES	Name of the threat reported (example, “ARP MITM”, “SSL STRIP”, “TCP SCAN”, etc.)
severity	string	YES	Severity of the threat reported (“CRITICAL”, “IMPORTANT”, “LOW”)
description	string	YES	Description of the attack detected by zIAP. The description returned might depend on the localization of the device.
ssid	string	NO	SSID of the access point when the attack was detected. Only present when the device was connected to a WiFi network.
package_name	string	NO	Package (app) name correlated to a given host threat. A package name uniquely identify an application installed on the device. Example : “com.geohot.towelroot”

4 – Implementation (Android)

4.1 – Implement aar inside the application

Android libraries are distributed as .aar files which contain all the code and resources for running zIAP. This file can be placed anywhere inside the ‘app module’ (for example, the appt/aars/ folder)

4.2 – Add zdetection.aar as a dependency

For the Android Gradle build system to leverage the zIAP classes, it must be added as a dependency. This is done by editing the app module’s build.gradle file and adding the following into dependencies:

```
dependencies {
  ...
  compile 'com.zimperium.zdetection:zdetection:1.0@aar'
  ...
}
```

4.3 - Specify the local application package name for zIAP

zIAP requires the following line be added to build.gradle so that the data provider does not conflict with other apps running zIAP. This should be added into the default configuration:

```
defaultConfig {
  ...
  manifestPlaceholders = [ localApplicationId:applicationId]
  ...
}
```

4.4 - Initialize threat monitoring

zIAP needs to monitor for threats in order to notify the calling application to take action of when threats occur. To ensure comprehensive coverage, zIAP should be monitoring for threats whenever the app is running.

The example code below is added into the onCreate()method of the Application class. When a threat is detected, the callback will trigger and any custom actions can be taken. In this example below, we are just printing some basic information about the detection on the logs.

```
protected void onCreate(Bundle savedInstanceState) {

  ZDetection.startDetection(getApplicationContext(), new ThreatCallback() {
       // Get generic information about the threat
       this.logText("\n");
       this.logText("Detected threat : " + threat.getHumanThreatName());
       this.logText("  - Severity : " + threat.getSeverity());
       this.logText("  - Type : " + threat.getHumanThreatType());
       this.logText("  - Description : " + threat.getDescription());

       // ---- Get additional (optional) data, might not be always present

       // SSID
       String ssid = threat.getSSID();
       if(ssid.length() > 0) {
       this.logText("  - SSID : " + ssid);
       }

       // Package name
       String packageName = threat.getPackageName();
       if(packageName.length() > 0) {
       this.logText("  - Malware Name : " + packageName);
       }
  });
}
```

5 - Implementation (iOS)

5.1 - Copy ZDetection.framework into the Xcode project

iOS libraries are distributed as .framework folders which contain all the code and resources for running zIAP. The same zIAP library powers zIPS as well.

5.2 – Add ZDetection.framework as an embedded binary

Once the file is added to the Xcode project, you must mark it as an 'embedded binary'. This is done by selecting your project details within Xcode, selecting the correct target, and dragging ZDetection.framework under the Embedded Binaries heading.

5.3 – Initialize threat monitoring

zIAP needs to monitor for threats in order to notify the calling application for zIAP to take action of when threats occur. To ensure comprehensive coverage, we want to ensure zIAP is monitoring for threats whenever the app is running.

The example code below is added into the application:didFinishLaunchingWithOptions: method of the Application Delegate class. When a threat is detected, the callback will trigger and any custom actions can be taken. In this example, we are just printing some basic information about the detection on the logs.

```
#import <ZDetection/ZDetection.h>

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
	[ZDetectionApi startDetectionWithThreatCallback:^(ZThreat *newThreat) {
	NSLog(@"Detected threat : %@", [newThreat humanThreatName]);
	NSLog(@"  - Severity : %@", [newThreat severity]);
	NSLog(@"  - Type : %@", [newThreat humanThreatType]);
	NSLog(@"  - Description : %@", [newThreat description]);
	NSLog(@"  - SSID : %@", [newThreat SSID]);
	}
}
```

6 – Support

For questions or assistance, please reach out to General Support at support@Zimperium.comor contact krishna.vishnubhotla@zimperium.com directly.