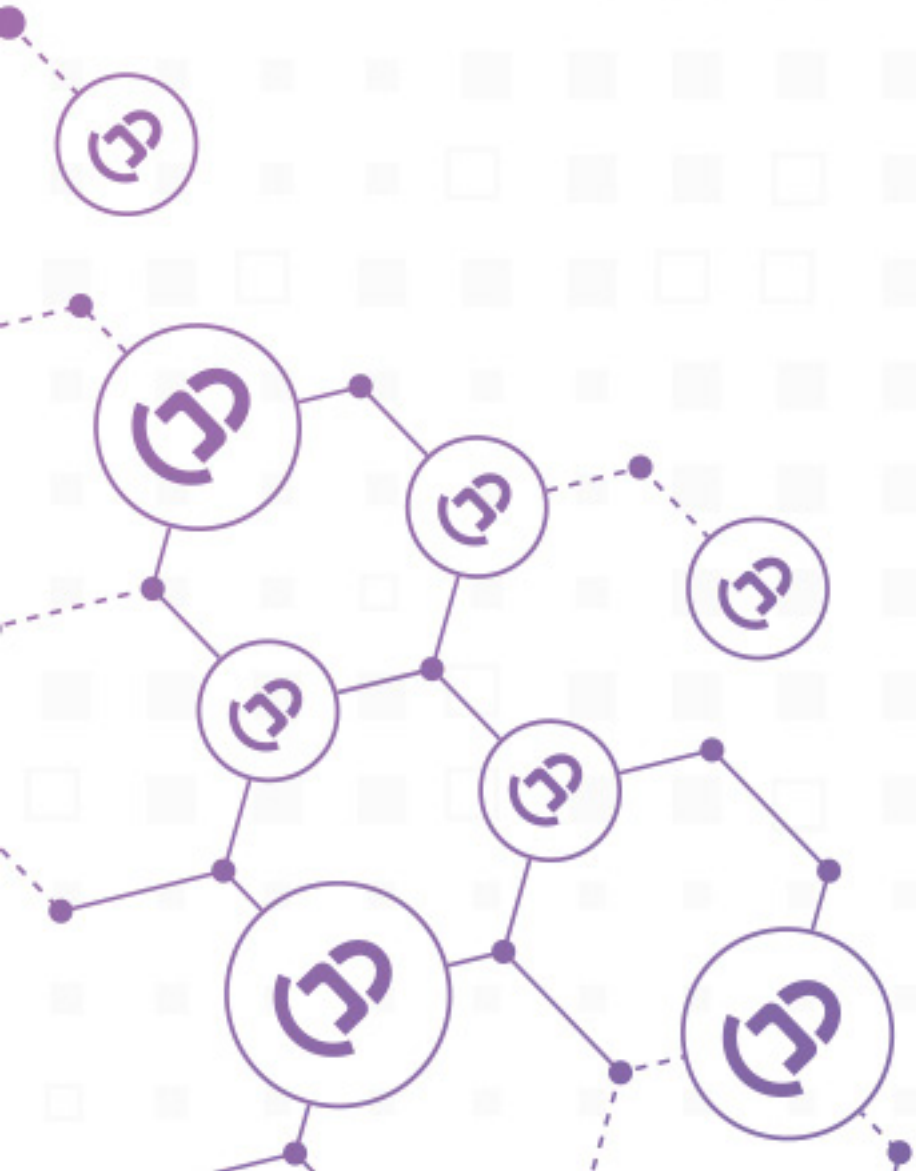
Version 1.0

APPSEALING SECURE

WEBVIEW USER GUIDE



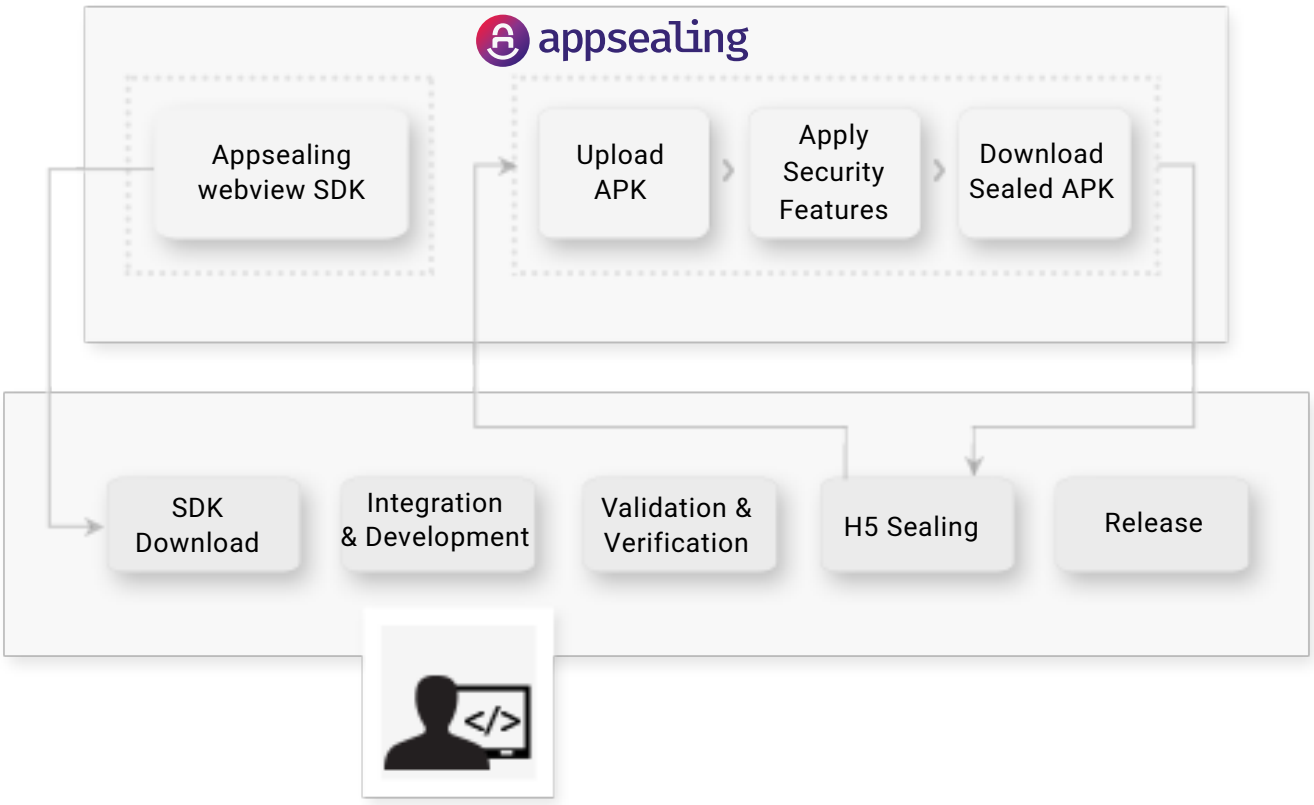
appsealing

TABLE OF CONTENTS

개요

안드로이드에서 제공하는 Android WebView는 네이티브 애플리케이션과 웹을 연결하거나, 로컬 웹 콘텐츠를 기반으로 하는 애플리케이션 제작에 사용되는 등 다양한 형태의 하이브리드 애플리케이션 개발에 활용되고 있습니다.
이러한 하이브리드 어플케이션의 장점에도 불구하고, 하이브리드 애플리케이션의 특성상 네이티브 애플리케이션과 웹 애플리케이션이 가지고 있는 각각의 보안 위협 요소가 모두 존재합니다. 특히 하이브리드 애플리케이션은 JavaScript로 작성된 원본 소스가 그대로 노출됩니다. 이로 인해서 비즈니스의 주요 알고리즘, 키 정보 및 민감한 데이터에 대한 공격에 취약해집니다
AppSealing Secure WebView를 사용하여 개발된 하이브리드 애플리케이션은 주요 비즈니스 로직 코드와 지적 재산권을 안전하게 보호합니다. AppSealing Secure WebView는 Chromium 기반으로 제작되었으며, Android WebView와 유사한 API를 제공하여 개발자가 손쉽게 AppSealing Secure WebView를 적용할 수 있도록 지원합니다.
HTML5 콘텐츠 암호화 절차에 대한 상세 설명은 AppSealing 홈페이지 또는 본 사용자 가이드를 참고해 주십시오. SDK 사용과 관련된 추가 기술 문의는 AppSealing 헬프센터(https://helpcenter.appsealing.com)를 이용해 주십시오.
NOTE : AppSealing Secure Webview SDK는 ADC 가입 후 신청할 수 있습니다. 가입 완료 후 승인 절차를 거치게 되며, 이후 ADC 사이트에서 다운로드할 수 있습니다

HYBRID APP 보안 적용 시나리오



1. AppSealing Secure WebView SDK를 다운로드합니다.
2. AppSealing Secure WebView SDK에 포함된 문서(본 문서와 AppSealing Secure WebView API 문서 및 샘플 앱)를 참고하여 AppSealing Secure WebView SDK가 적용된 Hybrid App을 개발합니다.
3. AppSealing Secure WebView SDK가 적용된 Hybrid App에 대하여 고객사 내부의 통상적인 개발 절차를 통해서 검증합니다.
4. 내부 검증을 마친 해당 앱에 AppSealing의 Hybrid App 보안 서비스(이하 Hybrid App 보안 서비스)를 적용하여 내부 콘텐츠 및 개발된 App 전체에 대한 보안을 적용합니다.
5. Hybrid App 보안 서비스가 적용된 App의 동작성을 확인한 후, 고객사의 통상적인 개발 절차에 따라서 배포합니다.

INTEGRATION & DEVELOPMENT

개발 지원 환경

Android 5.0버전 이상

본 SDK는 Android SDK 3.6.2에서 테스트 되었습니다.

SDK Package 구성물

구성물	설명
AppSealingSecureWebviewLibrary.aar	AppSealing Secure WebView를 구성하는 각종 리소스, asset 및 네이티브 라이브러리를 포함하는 android 라이브러리 파일
AppSealingSecureWebViewSample.zip	샘플 앱을 포함하는 안드로이드 스튜디오 프로젝트 파일
AppSealingSecureWebViewAPIDoc.zip	AppSealing Secure WebView에서 제공하는 API목록 및 설명

Android Studio 프로젝트 설정

다음과 같은 과정으로 AppSealing Secure WebView Android SDK의 구성물을 프로젝트에 추가할 수 있습니다. 압축 해제된 구성물 별로 사용하고 있는 프로젝트에 아래와 같이 추가합니다.

<AppSealingSecureWebviewLibrary.aar> 추가

사용하고 있는 프로젝트의 루트 폴더(`$Project_Dir`)안에 라이브러리 파일 저장 공간에 <AppSealingSecureWebviewLibrary.aar> 파일을 복사합니다. 사용하고 있는 프로젝트 폴더 안에 라이브러리 저장용 폴더가 없다면, `$Project_Dir₩app₩` 폴더에 “libs” 폴더를 생성하고 <AppSealingSecureWebviewLibrary.aar> 파일을 복사합니다. 그리고 `build.gradle` 파일에 아래 내용을 추가합니다

```
dependencies {
...
    implementation fileTree(dir: 'libs', include: ['*.aar'])
...
}
```

Java 1.8지원을 위한 설정

JAVA 1.8 지원을 위해서 아래 내용을 build.gradle에 반영합니다.

```
android {
...
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
...
}
```

비압축 Asset 지원을 위한 설정

Assets 파일에 대한 압축 비압축 설정을 수정합니다. AppSealing Secure WebView 내부에서 사용하는 데이터에 대해서 비압축 옵션을 설정하여야만 정상적으로 파일을 읽을 수 있습니다. build.gradle에 아래와 같이 수정합니다.

```
android {
  ...
    aaptOptions { noCompress 'dat', 'pak' }
  ...
}
```

현재 AppSealing Secure WebView에서는 확장자 “dat', “pak”에 대해서 비압축 설정을 필요로 합니다.

Build Tool 설정

모듈 build.gradle에 아래 SDK 을 설정합니다. AppSealing Secure WebView SDK 버전 16 이상을 지원합니다.

```
defaultConfig {
    minSdkVersion 16
    targetSdkVersion 28
}
```

상기의 기술된 설정은 SDK에 배포되는 샘플 프로젝트에 적용되어 있으니 참고하시기 바랍니다.

Application 작성 가이드

AppSealing Secure WebView 초기화

사용하는 Activity에서 AppSealing Secure WebView를 이용하려면, <AppSealing Secure WebView>를 사용하기 위한 초기화 과정이 필요합니다. onCreate()에서 아래의 순서로 AppSealing Secure WebView를 시작하기 위한 초기화 과정을 수행합니다.

```
@Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
.
.
.
        setContentView(R.layout.inka_webview_activity);
        mPreferences = new ASPreferences();

        mASInitializer = new ASInitializer(new ASInitializer.
        ASInitListener() {
            @Override
            public void onASInitStarted() {
                Log.d(TAG, "onASInitStarted");
            }
```

```
@Override
            public void onASInitCancelled() {
                Log.d(TAG, "onASInitCancelled");
            }

            @Override
            public void onASInitFailed() {
                Log.d(TAG, "onASInitFailed");
            }

            @Override
            public void onASInitCompleted() {
                Log.d(TAG, "onASInitCompleted");
                initASetting();
            }

            @Override
            public void onASFirstRenderProcessReady() {

            }
        }, this);

        if (!mASInitializer.initSync(disableSandbox)) {
          // Something is wrong.
        }
```

Page loading

초기화 과정이 끝나면 AppSealing Secure WebView 인스턴스를 생성하고, 원하는 페이지를 로딩합니다. 페이지를 로딩하기 위해서는 아래와 같이 코딩합니다.

```
boolean incognito = false;
 mASWebView = new ASWebView(MainActivity.this, incognito);
 mASWebView.loadUrl("http://google.com");
```

로컬 페이지 접속 예제

```
mASWebView.loadUrl("file:///android_asset/index.html");
```

이 경우는 [Applicaiontion_Root]/app/src/main/asset/index.html 이 존재하여야 합니다. 샘플 어플리케이션을 참조합니다.

원격 웹페이지 접속 예제

```
mASWebView.loadUrl ("http://www.naver.com");
```

setUIClient 사용법

AppSealing Secure WebView는 Android WebView와 마찬가지로 기본 웹브라우저의 동작을 재정의할 수 있는 방법을 제공합니다. AppSealing Secure WebView는 Android WebView의 `setWebChromeClient()` 메서드에 매칭되는 `setUIClient()`라는 메서드를 제공합니다. 구체적인 사용법은 아래의 예제를 참고하시기 바랍니다.

```
mASWebView.setUIClient(new ASUIClient(mASWebView) {
            @Override
            public boolean onJsAlert(ASWebView view, String url, String
message, ASJavascriptResult result) {
                return super.onJsAlert(view, url, message, result);
            }

            @Override
            public void onGeolocationPermissionsShowPrompt(ASWebView view,
java.lang.String origin,
                    ASGeoLocationPermissionsCallback callback) {
                callback.invoke(origin, true, false);
            }

            @Override
            public void onPageLoadStarted(ASWebView view, java.lang.String
url) {
                Log.e(TAG, "onPageLoadStarted =" + url);
                mIsLoading = true;
                if (mUrlTextView != null)
                    mUrlTextView.setText(url);


mStopReloadButton.setImageResource(android.R.drawable.ic_menu_close_clear_
cancel);
            }

            @Override
            public void onPageLoadStopped(ASWebView view, java.lang.String
url, ASUIClient.LoadStatus status) {
                Log.e(TAG, "onPageLoadStopped =" + url + " status=" +
status);
                mIsLoading = false;
                mStopReloadButton.setImageResource(R.drawable.ic_refresh);

           }
```

```
@Override
            public void onFullscreenToggled(ASWebView view, boolean
enterFullscreen) {
                super.onFullscreenToggled(view, enterFullscreen);
                Log.d(TAG, "onFullscreenToggled : " +
enterFullscreen);

                 LinearLayout toolBar = (LinearLayout)
findViewById(R.id.toolbar);
                toolBar.setVisibility(enterFullscreen ? GONE :
VISIBLE);

            }
              @Override
            public boolean onConsoleMessage(ASWebView view, String
message, int lineNumber, String sourceId,
                    ASUIClient.ConsoleMessageType messageType) {
                Log.d(TAG, "onConsoleMessage:" + message + "
source=" + sourceId + " line#=" + lineNumber);
                super.onConsoleMessage(view, message, lineNumber,
sourceId, messageType);
                return true;
            }
            // ...
        });
```

setResourceClient 사용법

Android WebView와 마찬가지로 웹브라우저가 주고받는 각각의 네트워크 연결을 관련된 웹브라우저의 동작을 재정의 할 수 있는 방법을 제공합니다. AppSealing Secure WebView는 `setResourceClient()` 메서드를 제공합니다. 이 메서드는 Android WebView의 `setWebViewClient()` 메서드에 매핑됩니다. 아래와 같이 사용하면 됩니다.

```
mASWebView.setResourceClient(new ASResourceClient(mASWebView) {
            @Override
            public void onLoadStarted(ASWebView view, String url) {
                Log.d(TAG, "onLoadStarted url=" + url);
                super.onLoadStarted(view, url);
            }
```

```
 @Override
            public void onLoadFinished(ASWebView view, String url)
{
                Log.d(TAG, "onLoadFinished url=" + url);
                super.onLoadFinished(view, url);
            }
@Override
            public ASWebResourceResponse
shouldInterceptLoadRequest(ASWebView view, ASWebResourceRequest
request) {
                Log.d(TAG, "shouldInterceptLoadRequest");
                return super.shouldInterceptLoadRequest(view,
request);

            }

            @Override
            public void onProgressChanged(ASWebView view, int
progressInPercent) {
                Log.d(TAG, "onProgressChanged Progress=" +
progressInPercent);
                super.onProgressChanged(view, progressInPercent);

mUrlTextView.removeCallbacks(mClearProgressRunnable);
                mProgressDrawable.setLevel((int) (progressInPercent
* 100));
                if (progressInPercent == 100)

mUrlTextView.postDelayed(mClearProgressRunnable,
COMPLETED_PROGRESS_TIMEOUT_MS);
            }

            @Override
            public void onReceivedLoadError(ASWebView view, int
errorCode, String description, String failingUrl) {
                Log.d(TAG,
                        "onReceivedLoadError errorCode=" +
errorCode + " url=" + failingUrl + " Desc=" + description);
                super.onReceivedLoadError(view, errorCode,
description, failingUrl);
            }

        });
```

별도로 제공되는 sample application 소스와 다양한 AppSealing API를 활용하여, 사용자가 원하는 형태로 application을 작성할 수 있습니다. AppSealing API에 대한 자세한 내용은 AppSealing API Document를 참조하세요.

HYBRID APP 보안 서비스 적용

AppSealing Secure WebView SDK를 탑재하고 있는 애플리케이션의 로컬 HTML5 콘텐츠 보안 및 애플리케이션 자체 보안 기능을 동작시키기 위해서는, 잉카엔트웍스에서 제공하는 Hybrid App 보안 서비스를 적용하여야 합니다.

다음과 같은 순서로 Hybrid App 보안 서비스를 적용할 수 있습니다.

1. AppSealing Developer Console 페이지(https://developer.appsealing.com/)에 접속합니다.
2. Hybrid App 보안 서비스를 이용하기 위한 계정 생성 또는 로그인 작업을 수행합니다.
3. AppSealing Secure Webview가 적용된 Hybrid App을 업로드하고 고객이 원하는 Hybrid App 보안 서비스를 설정합니다. 그리고 설정된 Hybrid App 보안 서비스를 적용합니다.(수 분 정도 소요됩니다)
4. Hybrid App 보안 서비스가 마무리되면, 보안 서비스가 적용된 APK를 다운로드할 수 있습니다.
5. APK를 다운로드하고, 서명을 다시 적용합니다.
6. 보안이 적용된 APK를 검증합니다.

자세한 내용은 해당 페이지의 설명을 참조합니다.

HYBRID APP 보안 서비스 적용 확인

로컬 테스트

1.android asset 폴더에 JavaScript를 포함하는 HTML 콘텐츠를 저장합니다.
2. Hybrid App 보안 서비스 적용하기 전에, 로컬 HTML 콘테츠가 정상적으로 AppSealing Secure WebView에서 로딩되는지 확인합니다.
3. 2번 단계 확인 후, APK를 빌드 하여 Hybrid App 보안 서비스를 적용합니다.
4. 보안 서비스 적용 후, 만들어진 APK를 다운로드합니다.
5. 다운로드한 APK를 zip 툴을 이용하여 열고, android asset 폴더에 같은 이름에 암호화된 JavaScript가 존재하는지 확인합니다.
6. 5번 사항 확인 후, 다운로드한 APK에서 로컬에 저장된 HTML 페이지가 정상적으로 로딩되는지 확인합니다.

원격 테스트

1. 원격 웹서버에 JavaScript를 포함하는 HTML 콘텐츠를 저장합니다.
2. Hybrid App 보안 서비스 적용하기 전에, 원격 웹서버의 HTML 콘테츠가 정상적으로 AppSealing Secure WebView에서 로딩되는지 확인합니다.
3. 2번 단계 확인 후, 원격 웹서버의 HTML 콘텐츠 중 암호화 및 보안 서비스를 적용하고자 하는 HTML 콘텐츠를 zip 하고 Hybrid App 보안 서비스를 적용합니다.
4. 보안 서비스 적용 후, 만들어진 zip 파일을 다운로드합니다.
5. 다운로드한 zip 파일을 열고, 원본 zip 파일과 같은 경로와 같은 이름으로 암호화된 JavaScript가 존재하는지 확인합니다.
6. 5번 단계 확인 후, 다운로드한 zip 파일을 압축 해제하고, 원격 웹서버에 저장합니다.
7. Hybrid App 보안 서비스가 적용된 HTML 페이지가 Hybrid App 보안 서비스가 적용된 AppSealing Secure WebView에서 정상적으로 로딩되는지 확인합니다.

테스트 진행 중 문의사항이나, 기술적인 지원은 AppSealing 헬프센터(https://helpcenter.appsealing.com/)를 통해서 문의하시기 바랍니다.

> APK에 Hybrid App 보안 서비스가 적용되기 전에는 암호화된 Javascript 페이지를 정상적으로 로딩할 수 없습니다.