AppSuit Cloud Android 적용 가이드

v 24.0.2

(주)스틸리언

Update to 2024-05-21

Table of Contents

1. 개요

- 본 문서는 AppSuit Cloud 솔루션을 Android Application(APK, AAB) 에 적용하기 위한 절차와 기능 설명 등에 대한 내용을 포함하고 있습니다.
- AppSuit Cloud 솔루션은 Android Application(APK, AAB) 에 대한 루팅 탐지, 위변조 탐지, 소스코드 난독화 및 암호화 등의 기능을 제공합니다.

a. 적용 환경

- 개발 환경
  - **Android Studio** : Arctic Fox 이상
  - **gradle plugin** : 3.2.0 버전 이상
  - **JDK** : 11 이상 **(반드시 JAVA_HOME 설정이 필요)**
- AppSuit module 환경
  - **AppSuit Plugin** : 24.1.1 버전 이상
  - **cloud-natv** : 24.0.7 버전 이상
- React Native 개발 환경 **(React Native 프로젝트의 경우)**
  - **React Native** : 0.64 이상
    - bundle 파일 암호화 지원을 위해 Hermes 활성화 필요
    - react-native-code-push를 사용하는 프로젝트의 경우, bundle 파일 암호화를 지원하지 않습니다.

b. 적용 과정

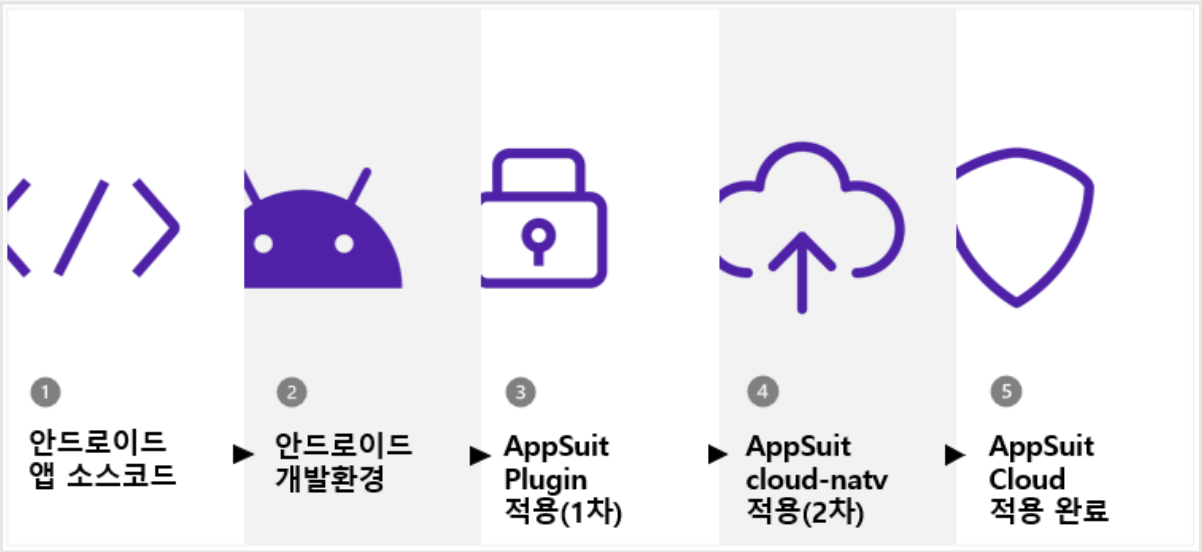

그림 1. AppSuit Cloud 적용 과정

- AppSuit Cloud 솔루션은 [그림 1]과 같이 2단계에 걸쳐 적용됩니다.
  1. **AppSuit Plugin(1차)** 적용은 Android Studio에서 이루어지며 소스코드 난독화 및 암호화 등의 기능이 적용됩니다.
  2. **AppSuit cloud-natv(2차)** 적용은 AppSuit Cloud 빌드 서버에서 이루어지며 위협 탐지 기능(루팅 탐지, 에뮬 탐지, 위변조 탐지 등)이 적용됩니다.

c. 적용 시 필요한 파일

1. **AppSuit Plugin**
   - AppSuit Cloud 1차 적용을 위한 파일입니다.
2. **AppSuit cloud-natv**
   - AppSuit Cloud 2차 적용을 위한 모듈입니다.
     스틸리언 내부 서버를 사용하는 경우 해당 파일은 제공되지 않습니다.
3. **rule.pro**
   - AppSuit Cloud 적용 시 기본적으로 필요한 예외처리 옵션이 작성된 파일입니다.
4. **AppSuit Cloud Android 적용 가이드**

2. AppSuit Cloud 적용 방법

a. AppSuit Plugin 파일 추가

1. AppSuit를 적용하고자 하는 프로젝트 내의 최상단 디렉토리에 **appsuit** 라는 이름의 디렉토리를 생성합니다.
2. **AppSuit Plugin 파일**(.jar)을 **appsuit** 디렉토리에 복사합니다.

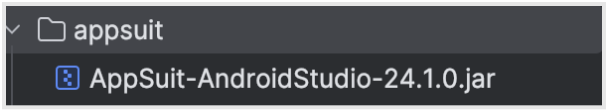

그림 2. AppSuit Plugin 파일 적용 예시

b. AppSuit rule 파일 추가

1. appsuit 디렉토리 내 **rule.pro 파일**을 생성 후, 해당 파일에 아래 기본 예외처리 옵션을 복사합니다.

```
-logout_std
-no_remove_logging #배포 시 제외

# android standard
-keep class android.** { *; }
-keep class androidx.** { *; }

-keepbare class android.** { *; }
-keepbare class androidx.** { *; }

-keep class org.apache.** { *; }
-keepstrings class org.apache.** { *; }

# kotlin library
-keep class kotlin.** { *; }
-keep class kotlinx.** { *; }

# firebase rule
-keep class com.google.** { *; }
-keepreflect class com.google.** { *; }
-keepstrings class com.google.** { *; }
-keepflow class com.google.** { *; }

# third-party
```

```
-keep class javax.** { *; }
-keep class okhttp3.** { *; }
-keepreflect class okhttp3.** { *; }

-keep class okio.** { *; }
-keep class retrofit2.** { *; }
-keep class io.reactivex.** { *; }
-keep class dagger.** { *; }
-keep class org.xmlpull.** { *; }
-keep class butterknife.** { *; }

-keep class com.facebook.** { *; }
-keepreflect class com.facebook.** { *; }
-keep class android.support.v4.app.** { *; }
-keepbare class android.support.v4.app.** { *; }

# except layout, component
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class * extends android.app.backup.BackupAgentHelper
-keep public class * extends android.preference.Preference
-keep public class * extends android.widget.TextView

-keepclassmembers class * { @android.webkit.JavascriptInterface <methods>;
}

-keepclasseswithmembernames class * {
native <methods>;
}

-keepclassmembers class * extends android.support.v7.app.AppCompatActivity
{
public void *(android.view.View);
}

-keepclassmembers class * extends android.app.Activity {
public void *(android.view.View);
}
```

```
-keepclassmembers enum * {
    values(...);
    valueOf(...);
}

-keepreflect class a.AppSuitDexLoader { *; }
-keepreflect class a.AppSuitDexLoader$* { *; }
-keepstrings class a.AppSuitDexLoader { *; }
-keepstrings class a.AppSuitDexLoader$* { *; }

-reflect_flow_target_all

-keepreflect class **.Dagger* { *; }

-keepreflect class **$** { public final invoke(...); }
-keepreflect class **$** { @androidx.compose.runtime.Composable <methods>;
}
-keepreflect class **.** { @androidx.compose.runtime.Composable <methods>;
}

-use_api_desugar
-use_d8
-sync_lib_proguard_rules
```

그림 3. AppSuit rule.pro 파일 기본 예외처리 옵션

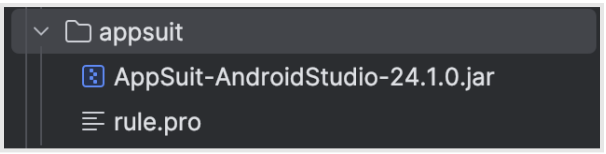

그림 4. AppSuit rule.pro 파일 생성

c. (app)build.gradle 설정

- app 모듈의 *build.gradle* 파일에 아래의 내용을 추가합니다.
  ※ 항목 ①에서 ⑥에 해당하는 값은 사용하시는 환경에 맞추어 변경하시기 바랍니다.
  - 항목 ①에서 ⑥에 대한 자세한 설명은 [2-e. build.gradle 설정 관련 설명] 부분 참고 부탁드립니다.

```
apply plugin: 'appsuit'

buildscript {
  repositories {
    flatDir dirs: '../appsuit'
  }
dependencies {
  classpath 'com.stealien.appsuit.gradle:[① AppSuit Plugin 파일명]'
  }
}

appsuit {
  serverOptions {
    host = '[② AppSuit Cloud 빌드 서버 URL]/api/v1'
    id '[③ ID]'
    pw '[PW]'
    options = [④ 2차 빌드 option]
    cloudBuild true
    enableServerBuild [⑤ boolean]
    processLibrary false
    }

    rulePath '../appsuit/rule.pro'
    dumpPath '../appsuit/dump.txt'
    buildTypes { ⑥
      release {
        enabled true
      }
      debug {
        enabled false
      }
    }
}
```

그림 5. (app)build.gradle 설정 예시

d. (app)build.gradle.kts 설정

※ 해당 gradle 설정의 경우 build.gradle.kts 파일을 사용하는 프로젝트에서만 진행합니다.

- project 모듈의 *build.gradle.kts* 파일에 아래의 내용을 추가합니다.
  ※ 항목 ①에 해당하는 값은 사용하시는 환경에 맞추어 변경하시기 바랍니다.
  - 항목 ①에 대한 자세한 설명은 [2-e. build.gradle 설정 관련 설명] 부분 참고 부탁드립니다.

```
buildscript {
    repositories {
        ...
        flatDir {
            dirs("appsuit")
        }
    }
    dependencies {
        ...
        classpath("com.stealien.appsuit.gradle:[① AppSuit Plugin 파일명]")
    }
}
```

그림 6. (project)build.gradle.kts 설정 예시

- AppSuit Plugin은 android application 플러그인을 필요로 합니다. project 모듈의 *build.gradle.kts* 파일에 android application 플러그인을 추가합니다.
  ※ 항목 ①에 해당하는 값은 사용하시는 환경에 맞추어 변경하시기 바랍니다.
- 현재 프로젝트에 사용중인 Android Gradle Plugin과 동일하게 설정합니다.

```
plugins{
    ...
    id("com.android.application") version [① Android Gradle Plugin
Version] apply false
    또는
    alias(libs.plugins.android.application) apply false
}
```

그림 7. (project)build.gradle.kts 설정 예시 - android application 플러그인 설정

- app 모듈의 *build.gradle.kts* 파일에 아래의 내용을 추가합니다.
  ※ 항목 ②에서 ⑥에 해당하는 값은 사용하시는 환경에 맞추어 변경하시기 바랍니다.
  - 항목 ②에서 ⑥에 대한 자세한 설명은 [2-e. build.gradle 설정 관련 설명] 부분 참고 부탁드립니다.

```
plugins {
    ...
    id("appsuit")
}
appsuit {
    serverOptions {
        host = "[② AppSuit Cloud 빌드 서버 URL]/api/v1"
        id = "[③ ID]"
        pw = "[PW]"
        options = listOf(
               [④ 2차 빌드 option]
        )
        cloudBuild = true
        enableServerBuild = [⑤ boolean]
        processLibrary = false
    }

    rulePath = "../appsuit/rule.pro"
    dumpPath = "../appsuit/dump.txt"
    buildTypes.create("release"){ ⑥
        enabled = true
    }
    buildTypes.create("debug"){
        enabled = false
    }
}
```

그림 8. (app)build.gradle.kts 설정 예시

e. React Native 프로젝트의 (app)build.gradle 설정

- React Native 프로젝트의 android 디렉토리 내, app 모듈의 *build.gradle* 파일에 아래의 내용을 추가합니다.
  ※ 항목 ①에서 ⑥에 해당하는 값은 사용하시는 환경에 맞추어 변경하시기 바랍니다.
  - 항목 ①에서 ⑥에 대한 자세한 설명은 [2-e. build.gradle 설정 관련 설명] 부분 참고 부탁드립니다.

```
buildscript {
  repositories {
    flatDir dirs: '../appsuit'
  }
dependencies {
  classpath 'com.stealien.appsuit.gradle:[① AppSuit Plugin 파일명]'
  }
} // 이 부분까지는 파일의 최상단에 위치하도록 합니다.

apply plugin: 'appsuit' // 이 부분부터는 파일의 최하단에 위치하도록 합니다.

appsuit {
  serverOptions {
    host = '[② AppSuit Cloud 빌드 서버 URL]/api/v1'
    id '[③ ID]'
    pw '[PW]'
    options = [④ 2차 빌드 option]
    cloudBuild true
    enableServerBuild [⑤ boolean]
    processLibrary false
    }

    rulePath '../appsuit/rule.pro'
    dumpPath '../appsuit/dump.txt'
    buildTypes { ⑥
      release {
        enabled true
      }
      debug {
        enabled false
      }
    }
}
```

그림 9. (app)build.gradle 설정 예시

f. React Native 프로젝트의 MainApplication.java 설정

- **encrypt_rn_bundle 옵션 사용 시에만 설정합니다.**
- **React Native 버전 0.72 이하로 생성한 프로젝트에 해당합니다.**
- 아래의 스니펫을 참조하여, mReactNativeHost의 DefaultReactNativeHost(this) 생성자 내에 getJSBundleFile() 함수를 오버라이딩하는 코드를 추가합니다.
- **debug, release 외의 커스텀 variant를 별도로 생성하여 사용하는 프로젝트의 경우, 해당 함수 내에서 React Native에서 debuggableVariants로 설정된 variant에 대한 예외 처리가 별도로 필요합니다.**
    - BuildConfig.BUILD_TYPE 스트링을 이용하여 해당하는 variant 여부를 확인할 수 있습니다.

```
  private final ReactNativeHost mReactNativeHost =
      new DefaultReactNativeHost(this) {
          // 아래의 함수를 추가합니다.
          @Override
          protected String getJSBundleFile() {
              if (BuildConfig.DEBUG) return null;
              else return getApplicationContext().getFilesDir() +
"/appsuit2/" + "index.android.bundle";
              // 이때 번들 파일의 이름은 bundleAssetName을 통해 설정한 번들 이름과
동일하게 설정합니다.
          }

          ...
```

그림 10. MainApplication.java 설정 예시

g. React Native 프로젝트의 MainApplication.kt 설정

- **encrypt_rn_bundle 옵션 사용 시에만 설정합니다.**
- **React Native 버전 0.73 이상으로 생성한 프로젝트에 해당합니다.**
- 아래의 스니펫을 참조하여, reactNativeHost의 DefaultReactNativeHost(this) 생성자 내에 getJSBundleFile() 함수를 오버라이딩하는 코드를 추가합니다.
- **debug, release 외의 커스텀 variant를 별도로 생성하여 사용하는 프로젝트의 경우, 해당 함수 내에서 React Native에서 debuggableVariants로 설정된 variant에 대한 예외 처리가 별도로 필요합니다.**
    - BuildConfig.BUILD_TYPE 스트링을 이용하여 해당하는 variant 여부를 확인할 수 있습니다.

```
  override val reactNativeHost: ReactNativeHost =
      object : DefaultReactNativeHost(this) {
        // 아래의 함수를 추가합니다.
        override fun getJSBundleFile(): String? {
          if (BuildConfig.DEBUG) return null
          else return applicationContext.filesDir.absolutePath +
"/appsuit2/" + "index.android.bundle";
          // 이때 번들 파일의 이름은 bundleAssetName을 통해 설정한 번들 이름과
동일하게 설정합니다.
        }
```

그림 11. MainApplication.kt 설정 예시

h. React Native 프로젝트의 ReactInstanceManager 설정 (Java)

- **encrypt_rn_bundle 옵션 사용 시에만 설정합니다.**
- react-native init 커맨드를 이용하여 생성된 프로젝트가 아닌, 기존에 존재하는 안드로이드 프로젝트에 React Native를 추가한 경우에 해당합니다.
- 아래의 스니펫을 참조하여, mReactInstanceManager 생성 및 설정 부분을 수정합니다.
  - 해당 변수의 이름은 구현에 따라 다를 수 있으며, ReactRootView를 생성하는 코드에서 사용하는 ReactInstanceManager 변수입니다.
- **debug, release 외의 커스텀 variant를 별도로 생성하여 사용하는 프로젝트의 경우, React Native에서 debuggableVariants로 설정된 variant에 대한 예외 처리가 별도로 필요합니다.**
  - BuildConfig.BUILD_TYPE 스트링을 이용하여 해당하는 variant 여부를 확인할 수 있습니다.

```
    // 기존에 사용하시던 mReactInstanceManager와 동일하게 설정하되,
    // setBundleAssetName() 함수 호출을 제외합니다.
    ReactInstanceManagerBuilder builder = ReactInstanceManager.builder()
            .setApplication(getApplication())
            .setCurrentActivity(this)
            .setJSMainModulePath("index")
            .addPackages(packages)
            .setUseDeveloperSupport(BuildConfig.DEBUG)
            .setInitialLifecycleState(LifecycleState.RESUMED);

    // 번들 파일의 이름은 bundleAssetName을 통해 설정한 번들 이름과 동일하게
설정합니다.
    if (BuildConfig.DEBUG) {
      builder = builder.setBundleAssetName("index.android.bundle");
    } else {
      builder = builder.setJSBundleFile(
              getApplicationContext().getFilesDir() + "/appsuit2/" +
"index.android.bundle");
    }

    mReactInstanceManager = builder.build();
```

그림 12. .ReactInstanceManager 설정 예시

i. React Native 프로젝트의 ReactInstanceManager 설정 (Kotlin)

- **encrypt_rn_bundle 옵션 사용 시에만 설정합니다.**
- react-native init 커맨드를 이용하여 생성된 프로젝트가 아닌, 기존에 존재하는 안드로이드 프로젝트에 React Native를 추가한 경우에 해당합니다.
- 아래의 스니펫을 참조하여, reactInstanceManager 생성 및 설정 부분을 수정합니다.
  - 해당 변수의 이름은 구현에 따라 다를 수 있으며, ReactRootView를 생성하는 코드에서 사용하는 ReactInstanceManager 변수입니다.
- **debug, release 외의 커스텀 variant를 별도로 생성하여 사용하는 프로젝트의 경우, React Native에서 debuggableVariants로 설정된 variant에 대한 예외 처리가 별도로 필요합니다.**
  - BuildConfig.BUILD_TYPE 스트링을 이용하여 해당하는 variant 여부를 확인할 수 있습니다.

```
        // 기존에 사용하시던 reactInstanceManager와 동일하게 설정하되,
        // setBundleAssetName() 함수 호출을 제외합니다.
        var builder = ReactInstanceManager.builder()
            .setApplication(application)
            .setCurrentActivity(this)
            .setJSMainModulePath("index")
            .addPackages(packages)
            .setUseDeveloperSupport(BuildConfig.DEBUG)
            .setInitialLifecycleState(LifecycleState.RESUMED)

        // 번들 파일의 이름은 bundleAssetName을 통해 설정한 번들 이름과 동일하게
설정합니다.
        if (BuildConfig.DEBUG) {
            builder = builder.setBundleAssetName("index.android.bundle")
        } else {
            builder = builder.setJSBundleFile(
                applicationContext.filesDir.absolutePath
                        + "/appsuit2/" + "index.android.bundle"
            )
        }

        reactInstanceManager = builder.build()
```

그림 13. .ReactInstanceManager 설정 예시

j. build.gradle 설정 관련 설명

- **① AppSuit Plugin 파일명**
  - 설명 : **AppSuit Plugin 파일 이름**과 동일하게 지정합니다.
  - 적용 방법

```
classpath 'com.stealien.appsuit.gradle:[AppSuit Plugin 파일명]'
```

  - 예시

```
classpath 'com.stealien.appsuit.gradle:AppSuit-AndroidStudio(24.1.1)'
```

  - **주의사항 : AppSuit Plugin 확장자 .jar은 제외하여 입력합니다.**

- **② AppSuit Cloud 빌드 서버 URL**
  - 설명 : **AppSuit Cloud** 빌드 서버 주소를 지정합니다.
  - 적용 방법

```
host = '[AppSuit Cloud 빌드 서버 URL]/api/v1'
```

  - 예시

```
host = 'https://cloud.appsu.it/api/v1'
```

  - **주의사항 : 서버 주소 뒤 /api/v1을 반드시 입력해야 합니다.**

- **③ ID / PW**
  - 설명 : **AppSuit Cloud** 빌드 서버에 접근 및 사용하기 위한 계정과 암호를 지정합니다.
  - 적용 방법

```
id '[ID]'
pw '[PW]'
```

  - 예시

```
id 'test'
pw 'Test123!'
```

  - **주의사항 : 해당 프로젝트를 빌드 하기 위한 라이센스 등록을 진행한 계정 정보를 입력합니다.**
- **④ 2차 빌드 option**
  - 설명 : **AppSuit Cloud** 빌드 서버에 전달되는 옵션입니다.
    AppSuit cloud-natv(2차) 적용 시 사용되며, 해당 옵션에 대한 설명은 [부록-D] 참고 부탁드립니다.
  - 적용 방법
    - *build.gradle* 의 경우

```
options = [2차 빌드 option]
```

      - 예시

```
options = ['check_root', 'check_emul', 'check_integrity']
```

    - *build.gradle.kts* 의 경우

```
options = listOf([2차 빌드 option])
```

      - 예시

```
options = listOf("check_root", "check_emul", "check_integrity")
```

- **⑤ enableServerBuild Flag**
  - 설명 : **AppSuit cloud-natv(2차)** 적용 여부를 결정합니다.
  - 적용 방법

    ```
    enableServerBuild [true / false]
    ```

  - 예시

    ```
    enableServerBuild true
    ```

  - **주의사항 : false로 설정 시 AppSuit cloud-natv(2차)가 적용되지 않습니다.**
- **⑥ buildTypes**
  - 설명 : Build Variants 별 AppSuit Cloud 적용 여부 지정합니다.
    release, debug 등 빌드 타입을 지정할 수 있습니다.
  - 적용 방법
    - *build.gradle* 의 경우

      ```
      buildTypes {
        [Build Variants] {
          enabled [true / false]
        }
      }
      ```

    - 예시

      ```
      buildTypes {
        release { enabled true }
        debug { enabled false }
      }
      ```

- *build.gradle.kts* 의 경우

```
buildTypes.create("[Build Variants]"){
    enabled = [true / false]
}
```

  - 예시

```
buildTypes.create("release"){
    enabled = true
}
buildTypes.create("debug"){
    enabled = false
}
```

- **주의사항 : 별도로 지정하지 않으면 false로 설정됩니다.**

3. 빌드 및 결과 확인

a. AppSuit 설정 후 빌드

1. Android Studio에서 **Build** 메뉴를 클릭합니다.
2. **Generate Signed Bundle / APK** 메뉴를 클릭합니다.
3. **Build Type**을 선택합니다.
4. **Finish** 버튼을 클릭합니다.

b. 빌드 결과 확인

- 다음과 같이 정상적으로 AppSuit Cloud 가 빌드 된 것을 확인합니다.

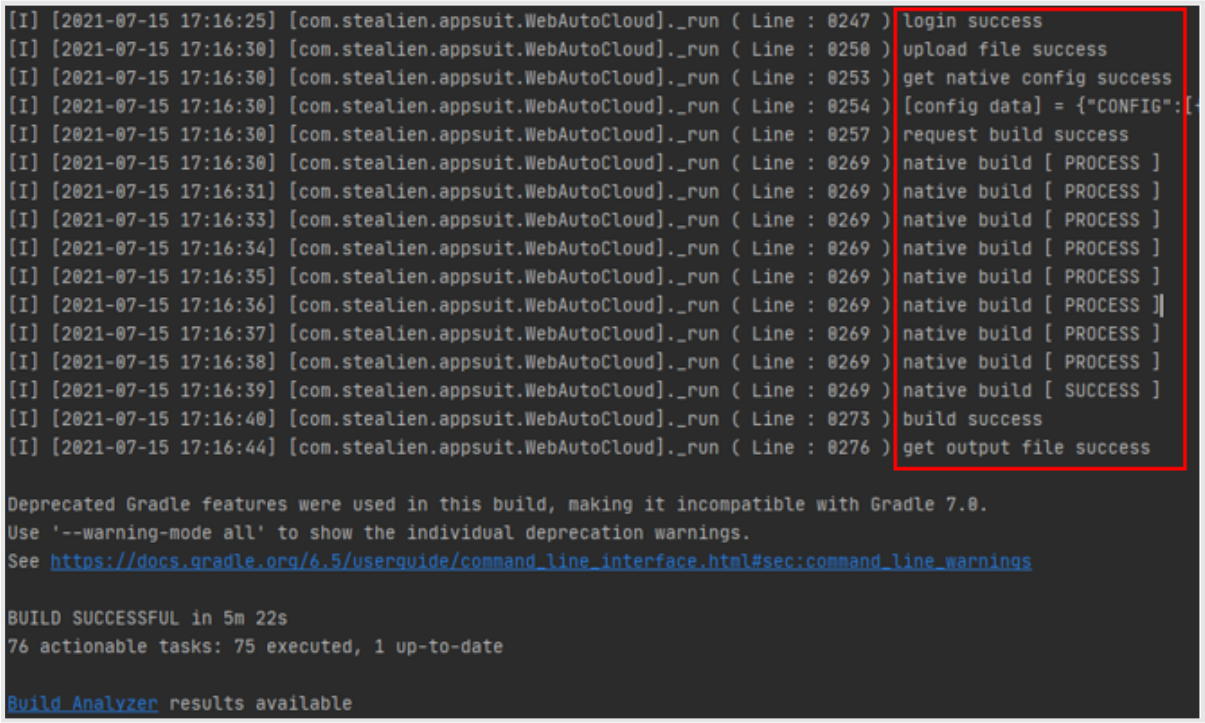

그림 14. AppSuit Cloud 적용 확인 예시

c. AppSuit 적용 앱 실행

1. 지정한 앱 추출 경로(AppSuit 적용 전과 동일한 경로)에 앱이 생성된 것을 확인합니다.
2. 해당 앱을 단말기에 설치합니다.
3. 설치한 앱이 정상적으로 실행되는지 확인합니다.

4. AppSuit Cloud 검증 방법

※ 입력하는 명령어의 경우 환경 변수 및 PATH 설정 여부에 따라 다를 수 있다는 점 참고 부탁드립니다.

a. AppSuit Cloud 솔루션 적용 검증

- **검증 시나리오**

1. 해당 APK / AAB의 내부를 확인합니다.
2. [APK / AAB의 내부] - assets – appsuit 디렉토리 내 *momo* 파일이 존재하는 경우 AppSuit Plugin 적용(1차)된 aab 임을 확인할 수 있습니다.

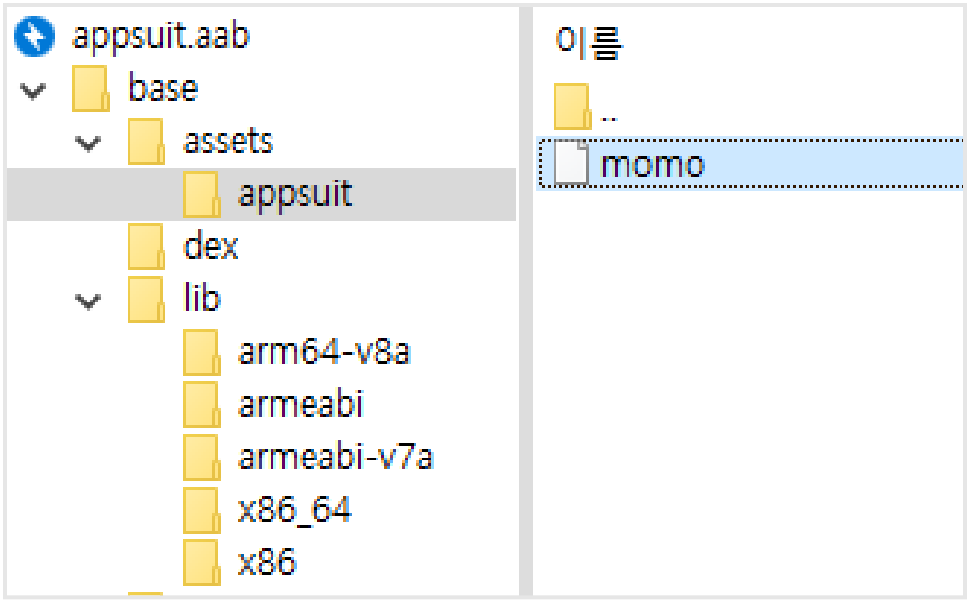

그림 15. appSuit 디렉토리 확인

3. [APK / AAB의 내부] - lib - arch 디렉토리 내 *libAppSuit.so* 파일이 존재할 경우 AppSuit cloud-natv 적용(2차)된 aab 임을 확인할 수 있습니다.

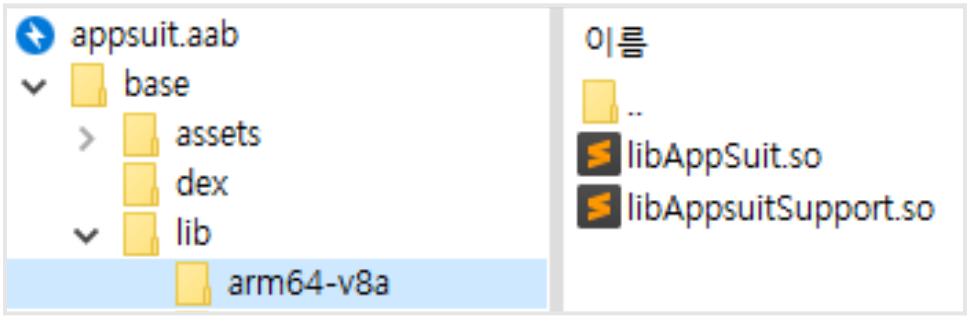

그림 16. arch 디렉토리 확인

b. 난독화 여부 검증

- **검증 환경**
  - 파일 분석용 PC
    - jadx-gui 프로그램이 필요합니다.
      - jadx-gui의 경우 APK를 분석하기 위한 디컴파일러이며,
        이는 https://github.com/skylot/jadx/releases 에서 다운로드 가능합니다.
  - AppSuit Cloud를 적용한 앱 (APK / AAB)
  - AppSuit Cloud를 적용하지 않은 앱 (APK / AAB)

- **검증 시나리오**

1. jadx-gui로 AppSuit Cloud가 적용된 앱과 적용되지 않은 앱을 각각 실행합니다.

```
public PodcastCategoryEntryDao_Impl(RoomDatabase __db) {
    this.__db = __db;
    this.__insertionAdapterOfPodcastCategoryEntry = new EntityInsertionAdapter<PodcastCategoryEntry>(__db) { // from class: com.example
        @Override // androidx.room.SharedSQLiteStatement
        public String createQuery() {
            return "INSERT OR REPLACE INTO `podcast_category_entries` (`id`,`podcast_uri`,`category_id`) VALUES (nullif(?, 0),?,?)";
        }

        @Override // androidx.room.EntityInsertionAdapter
        public void bind(SupportSQLiteStatement stmt, PodcastCategoryEntry value) {
            stmt.bindLong(R.styleable.ActivityNavigator, value.getId());
            if (value.getPodcastUri() == null) {
                stmt.bindNull(R.styleable.ActivityRule);
            } else {
                stmt.bindString(R.styleable.ActivityRule, value.getPodcastUri());
            }
            stmt.bindLong(R.styleable.AnimatedStateListDrawableCompat, value.getCategoryId());
        }
    };
```

그림 17. AppSuit Cloud 적용 전 난독화 여부 확인

```
public kex(RoomDatabase __db) {
    this.aue = __db;
    this.key = new EntityInsertionAdapter<ihy>(__db) { // from class: kex.1
        public String createQuery() {
            return ady.aes(new byte[]{102, 56, -110, -62, 125, 34, -31, -56, 125, 86, -109, -62, ByteCompanionObject.MAX_VALUE, 58, ByteCompanionObject.MIN_VALUE, -60
        }

        /* renamed from: ama */
        public void bind(SupportSQLiteStatement stmt, ihy value) {
            char c = 1;
            char c2 = 0;
            int i = Integer.parseInt(ady.ael(new byte[]{4}, 2096734298, -1583633172, false)) > 0 ? 1 : 0;
            long bhh = value.bhh();
            int i2 = hps.hqc;
            int i3 = 2;
            Object[] objArr = new Object[Integer.parseInt(ady.aet(1419082492, new byte[]{-13}, -367621721, -2136846512, -1705524883, false)) > 1 ? 2 : 1];
            objArr[Integer.parseInt(ady.aet(1415748262, new byte[]{54}, 788728015, 1659753475, -1332449677, false)) > 1 ? (char) 1 : (char) 0] = Integer.valueOf(i);
            objArr[Integer.parseInt(ady.ael(new byte[]{4}, 2096734298, -1583633172, false)) > 0 ? (char) 1 : (char) 0] = Long.valueOf(bhh);
            hps.gcz(stmt, i2, objArr);
            if (value.ecx() == null) {
                if (Integer.parseInt(ady.aet(1419082492, new byte[]{-13}, -367621721, -2136846512, -1705524883, false)) > 3) {
                    i3 = 3;
                }
                int i4 = hps.hwr;
                Object[] objArr2 = new Object[Integer.parseInt(ady.ael(new byte[]{4}, 2096734298, -1583633172, false)) > 0 ? 1 : 0];
                if (Integer.parseInt(ady.aet(1415748262, new byte[]{54}, 788728015, 1659753475, -1332449677, false)) <= 1) {
                    c = 0;
                }
                objArr2[c] = Integer.valueOf(i3);
                hps.gcz(stmt, i4, objArr2);
                return;
            }
            int i5 = Integer.parseInt(ady.aet(1419082492, new byte[]{-13}, -367621721, -2136846512, -1705524883, false)) > 3 ? 3 : 2;
            String ecx = value.ecx();
            int i6 = hps.hwd;
            if (Integer.parseInt(ady.aet(1419082492, new byte[]{-13}, -367621721, -2136846512, -1705524883, false)) > 3) {
                i3 = 3;
            }
            Object[] objArr3 = new Object[i3];
            if (Integer.parseInt(ady.aet(1415748262, new byte[]{54}, 788728015, 1659753475, -1332449677, false)) > 1) {
                c2 = 1;
            }
            objArr3[c2] = Integer.valueOf(i5);
            objArr3[1] = ecx;
            hps.gcz(stmt, i6, objArr3);
        }
    };
```

그림 18. AppSuit Cloud 적용 후 난독화 여부 확인

c. 루팅 탐지 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'check_root',
            'detect_magisk'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 정상 디바이스, 루팅 디바이스, Magisk 디바이스
  - ADB가 설치된 분석용 PC
    - ADB의 경우 PC와 디바이스 간에 통신을 할 수 있는 명령어 도구이며, 이는 Android Studio에서 SDK 설치 시 함께 다운로드 됩니다.
      [Android SDK Path]/platform-tools 에 있으니 참고 부탁드립니다.
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오(For. 정상 디바이스)**
  1. AppSuit Cloud 솔루션이 적용된 앱을 설치합니다.

     ```
     host$ adb install [APK_AAB_FILE_PATH]
     ```

  2. 앱을 실행하여 정상적으로 실행되는 것을 확인합니다.
  3. background에 살아있는 프로세스가 없도록 앱을 완전히 종료합니다.
  4. ADB를 통해 디바이스 Shell에 접속하여, '/sdcard/APPSUIT_ROOTING_TEST' 경로의 파일을 생성합니다.

     ```
     host$ adb shell

     device$ touch /sdcard/APPSUIT_ROOTING_TEST
     ```

  5. 앱을 재실행 하여 앱이 종료되는 것을 확인합니다.
  6. 확인 후 생성한 *APPSUIT_ROOTING_TEST* 파일을 반드시 삭제합니다.

     ```
     host$ adb shell

     device$ rm /sdcard/APPSUIT_ROOTING_TEST
     ```

- **검증 시나리오(For. 루팅 디바이스)**
  1. AppSuit Cloud 솔루션이 적용된 앱을 설치합니다.

     ```
     host$ adb install [APK_AAB_FILE_PATH]
     ```

  2. 앱을 실행하여 앱이 종료되는 것을 확인합니다.

- **검증 시나리오(For. magisk 디바이스)**
  1. AppSuit Cloud가 적용된 앱을 설치합니다.

     ```
     host$ adb install [APK_AAB_FILE_PATH]
     ```

  2. magisk manager앱에서 magisk 자체 보호 기법을 전부 적용합니다. (Zygisk, DenyList 등)
  3. 앱을 실행 하여 앱이 종료되는 것을 확인합니다.

e. 에뮬레이터 탐지 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'check_emul'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 정상 디바이스
  - ADB가 설치된 분석용 PC
    - NOX, bluestack, memu 등의 에뮬레이팅 프로그램이 필요합니다.
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오**
  1. AppSuit Cloud가 적용된 앱을 정상 디바이스에 설치합니다.
  2. 앱을 실행하여 정상적으로 실행되는 것을 확인합니다.
  3. 동일한 앱을 에뮬레이터 프로그램에 설치합니다.
  4. 앱을 실행 하여 앱이 종료되는 것을 확인합니다.

f. 위변조 탐지 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'check_integrity'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 정상 디바이스
  - ADB가 설치된 분석용 PC
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오(For. AAB(split_apks) - DEX 위변조)**
  1. AppSuit Cloud가 적용된 앱을 정상 디바이스에 설치합니다.
     - aab를 마켓과 동일한 방식으로 설치하기 위해서는 bundletool이 필요하며,
       이는 https://github.com/google/bundletool/releases 에서 다운로드 가능합니다.
  2. 앱을 실행하여 정상적으로 실행되는 것을 확인합니다.
  3. ADB 명령어로 AppSuit Cloud가 적용된 앱의 패키지명을 획득합니다.

```
host$ adb shell pm list packages
    > ...........
    > package:com.dp.app_shell
```

```
    > ..........
```

4. ADB 명령어로 앱의 패키지명을 통해 설치된 split apk들의 path를 획득합니다.

```
host$ adb shell pm path [PACKAGE_NAME]
    > package:/data/app/~~OU8U8gQH2mA-5iUt93g--
Q==/com.dp.app_shell-3ECA3cd2ou-X6HDWFPvaGA==/base.apk
    > package:/data/app/~~OU8U8gQH2mA-5iUt93g--
Q==/com.dp.app_shell-3ECA3cd2ou-
X6HDWFPvaGA==/split_config.arm64_v8a.apk
    > package:/data/app/~~OU8U8gQH2mA-5iUt93g--
Q==/com.dp.app_shell-3ECA3cd2ou-X6HDWFPvaGA==/split_config.en.apk
    > package:/data/app/~~OU8U8gQH2mA-5iUt93g--
Q==/com.dp.app_shell-3ECA3cd2ou-
X6HDWFPvaGA==/split_config.xxhdpi.apk
```

5. 디바이스에 설치된 split apk들 추출합니다.

```
host$ adb pull /data/app/~~OU8U8gQH2mA-5iUt93g--
Q==/com.dp.app_shell-3ECA3cd2ou-X6HDWFPvaGA==/base.apk
[WORKING_DIR]
host$ adb pull .........
host$ adb pull .........
.........
```

6. 추출한 base.apk를 *apktool*로 디컴파일합니다.
    - apktool의 경우 APK를 리버스 엔지니어링하기 위한 도구이며,
      이는 https://ibotpeaches.github.io/Apktool/ 에서 다운로드 가능합니다.

```
host$ java -jar apktool.jar d base.apk -o decom
```

7. 디컴파일된 디렉토리 내부에 존재하는 .smali 확장자를 갖는 파일 수정합니다.
    - 코드 영역에 *nop*를 삽입하거나, 줄 바꿈 등을 수행합니다.
8. *apktool*을 사용해 변경된 정보를 갖는 APK를 빌드합니다.

```
host$ java -jar apktool.jar b decom -o modi.apk
```

9. 위변조 완료된 apk를 포함해서 추출한 모든 apk들을 *zipalign* 과 *apksigner* 로 재서명합니다.
    - [Android SDK Path]/build-tools 에 있으니 참고 부탁드립니다.

```
host$ zipalign -p 4 modi.apk modi_aligned.apk
host$ apksigner sign --ks [SIGNING_JKS_FILE] --ks-key-alias [ALIAS]
--ks-pass pass:[PASSWORD] --key-pass pass:[PASSWORD] base.apk

# 위변조하지 않은 다른 split apk들을 재서명
host$ zipalign -p 4 split_config.arm64_v8a.apk
split_config_aligned.arm64_v8a.apk
host$ apksigner sign --ks [SIGNING_JKS_FILE] --ks-key-alias [ALIAS]
--ks-pass pass:[PASSWORD] --key-pass pass:[PASSWORD]
split_config.arm64_v8a.apk

host$ zipalign -p 4 split_config.en.apk split_config_aligned.en.apk
host$ apksigner sign --ks [SIGNING_JKS_FILE] --ks-key-alias [ALIAS]
--ks-pass pass:[PASSWORD] --key-pass pass:[PASSWORD]
split_config.en.apk

host$ zipalign -p 4 split_config.xxhdpi.apk
split_config_aligned.xxhdpi.apk
host$ apksigner sign --ks [SIGNING_JKS_FILE] --ks-key-alias [ALIAS]
--ks-pass pass:[PASSWORD] --key-pass pass:[PASSWORD]
split_config.xxhdpi.apk
```

10. 재서명된 apk들을 디바이스에 설치합니다.

```
# 앞서 재서명한 모든 apk들을 인자로 넣어서 한번에 설치합니다.
host$ adb install-multiple modi_aligned.apk
split_config_aligned.arm64_v8a.apk .....
```

11. 앱을 재실행 하여 앱이 종료되는 것을 확인합니다.

g. 디버거 탐지 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'check_debugging'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 루팅 디바이스
    - 루팅 디바이스가 없는 경우 NOX로 대체 가능합니다.
  - ADB가 설치된 분석용 PC
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오(Using. NOX)**

1. 분석용 pc의 ADB를 NOX와 연결합니다.
2. AppSuit Cloud가 적용된 앱을 설치합니다.

```
# Using. NOX
host$ adb connect 127.0.0.1:62001 # nox 와 연결하는 명령어입니다
host$ adb install [APK_AAB_FILE_PATH]
```

3. 앱을 실행하여 정상적으로 실행되는 것을 확인합니다.
4. ADB를 통해 디바이스 Shell에 접속합니다.
5. 설치한 앱에 해당하는 프로세스들을 확인합니다.

```
host$ adb shell

device$ ps
    > ........
    > .. u0_a45   5036 1873
    > .. u0_a45   5053 5036
    > ........
```

6. strace 명령어를 통해 디버깅 시작합니다.
7. strace 명령어 사용 직후, 앱이 종료되는 것을 확인합니다.

```
device$ strace -p 5036
    > strace: Process 5036 attached
    > ........
    > +++ killed by SIGSEGV +++
```

- **검증 시나리오(For. Frida 탐지)**
  1. AppSuit Cloud가 적용된 앱을 설치합니다.

     ```
     host$ adb install [APK_AAB_FILE_PATH]
     ```

  2. 앱을 실행하여 정상적으로 실행되는 것을 확인합니다.
  3. background에 살아있는 프로세스가 없도록 앱을 완전히 종료합니다.
  4. 디바이스 arch에 맞는 frida-server를 설치합니다.
     - https://github.com/frida/frida/releases 링크에서 설치 가능합니다.
  5. 다운받은 frida-server 파일을 디바이스 내부 /data/local/tmp/ 경로에 복사합니다.

     ```
     host$ adb push [FRIDA_SERVER_PATH] /data/local/tmp
     ```

  6. ADB를 통해 디바이스 Shell에 접속합니다.
  7. frida-server에 실행권한을 부여합니다.
  8. frida-server를 실행합니다.

     ```
     host$ adb shell

     device$ cd /data/local/tmp/
     device$ chmod a+x [FRIDA_SERVER_FILE]
     device$ ./[FRIDA_SERVER_FILE]
     ```

  9. 앱을 재실행 하여 앱이 종료되는 것을 확인합니다.

h. 디버깅 방지 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'prevent_debugging'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 루팅 디바이스
    - 루팅 디바이스가 없는 경우 NOX로 대체 가능합니다.
  - ADB가 설치된 분석용 PC
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오(Using. NOX)**

1. 분석용 pc의 ADB를 NOX와 연결합니다.
2. AppSuit Cloud가 적용된 앱을 설치합니다.

```
# Using. NOX
host$ adb connect 127.0.0.1:62001 # nox 와 연결하는 명령어입니다
host$ adb install [APK_AAB_FILE_PATH]
```

3. 앱을 실행하여 정상적으로 실행되는 것을 확인합니다.
4. ADB를 통해 디바이스 Shell에 접속합니다.
5. 설치한 앱에 해당하는 프로세스들을 확인합니다.

```
host$ adb shell

device$ ps
    > ........
    > .. u0_a45   5036 1873
    > .. u0_a45   5053 5036
    > ........
```

6. 메인 프로세스의 tracerPid가 child process의 pid와 동일한지 확인합니다.

```
device$ cat /proc/5036/status
    > ........
    > .. TracerPid:      5053
    > ........
```

7. strace 명령어를 통해 ptrace가 선점되었는지 다시 한 번 확인합니다.
    - **해당 방식의 경우 strace가 내장되어있지 않은 디바이스에서는 확인 불가능합니다.**
8. child process가 ptrace를 선점하고 있어 strace가 정상적으로 동작하지 않는 것을 확인합니다.

```
device$ strace -p 5036
    > ptrace(PTRACE_ATTACH, ...): Operation not permitted
```

- **검증 시나리오(For. 루팅 디바이스)**

1. AppSuit Cloud가 적용된 앱을 설치합니다.

```
host$ adb install [APK_AAB_FILE_PATH]
```

2. 앱을 실행하여 정상적으로 실행되는 것을 확인합니다.
3. ADB를 통해 디바이스 Shell에 접속합니다.
4. 설치한 앱에 해당하는 프로세스들 확인합니다.

```
host$ adb shell

device$ ps
    > ........
    > .. u0_a45   5036 1873
    > .. u0_a45   5053 5036
    > ........
```

5. 메인 프로세스의 tracerPid가 child process의 pid와 동일한지 확인합니다.

```
device$ cat /proc/5036/status
    > ........
    > .. TracerPid:      5053
    > ........
```

i. ADB 방지 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'prevent_adb'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 정상 디바이스
  - ADB가 설치된 분석용 PC
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오**

1. AppSuit Cloud가 적용된 앱을 설치합니다.

```
host$ adb install [APK_AAB_FILE_PATH]
```

2. 디바이스에서 USB 디버깅을 완전히 비활성화합니다.
   - *adb kill-server* 커맨드를 실행합니다.
   - USB 케이블을 연결 해제합니다.
   - 개발자 옵션 및 USB 디버깅 옵션을 비활성화합니다.

```
host$ adb kill-server

... USB 케이블을 연결 해제합니다. ...
... 개발자 옵션 및 USB 디버깅 옵션을 비활성화합니다....
```

3. 앱을 실행하여 정상적으로 실행되는 것을 확인합니다.
4. 백그라운드 프로세스가 남아있지 않도록 앱을 완전히 종료합니다.
5. 개발자 옵션 및 USB 디버깅 옵션을 활성화한 다음, ADB를 통해 디바이스 Shell에 접속합니다.

```
host$ adb shell
```

6. 앱을 재실행 하여 앱이 종료되는 것을 확인합니다.

j. SO 암호화 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'encrypt_so'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - ELF 파일분석을 위한 PC
    - 솔루션과 개발환경 이외에 추가적인 분석 도구가 필요합니다.
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오(Using. 010editor)**
  1. AppSuit Cloud가 적용된 앱을 unzip합니다.
     - *apktool*로 패키지를 풀어도 무관합니다.
  2. 솔루션이 적용된 임의의 so library 파일 추출하여 검사를 진행합니다.
     - encrypt_so 옵션의 경우 *libAppSuit.so*는 암호화 대상이 아님으로 해당 파일을 제외한 다른 파일을 선택해야 합니다.
  3. [something.so] 파일을 010 editor로 open합니다.
     - 010 editor의 경우 16 진수 편집기이며,
       이는 https://www.sweetscape.com/010editor/ 에서 다운로드 가능합니다.

4. ELF template으로 run합니다.
5. parsing된 결과 탭에서 section이 3개만 존재하는 것을 확인합니다.

k. 솔루션 코드 자체 암호화 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'self_protect'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - ELF 파일분석을 위한 PC
    - 솔루션과 개발환경 이외에 추가적인 분석 도구가 필요합니다.
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오(Using. 010editor)**
  1. AppSuit Cloud가 적용된 앱을 unzip합니다.
     - *apktool*로 패키지를 풀어도 무관합니다.
  2. lib 디렉토리 내부에 존재하는 *libAppSuit.so* 파일을 대상으로 검사 진행합니다.
  3. *libAppSuit.so* 파일을 010 editor로 open합니다.
     - 010 editor의 경우 16 진수 편집기이며,
       이는 https://www.sweetscape.com/010editor/ 에서 다운로드 가능합니다.
  4. ELF template으로 run합니다.
  5. parsing된 결과 탭에서 section이 3개만 존재하는 것을 확인합니다.

I. 디컴파일 방지 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'prevent_decompile'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 파일 분석용 PC
    - jadx-gui 프로그램이 필요합니다.
      - jadx-gui의 경우 Java 디컴파일러이며,
        이는 https://github.com/skylot/jadx/releases 에서 다운로드 가능합니다.
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오(Using. jadx-gui)**
  1. jadx-gui로 AppSuit Cloud가 적용된 앱을 실행합니다.
  2. 디컴파일 결과에 아래의 이미지와 같은 ERROR가 발생하는지를 확인합니다.
     - UNKNWON INSTRUCTION류의 에러가 발생합니다.

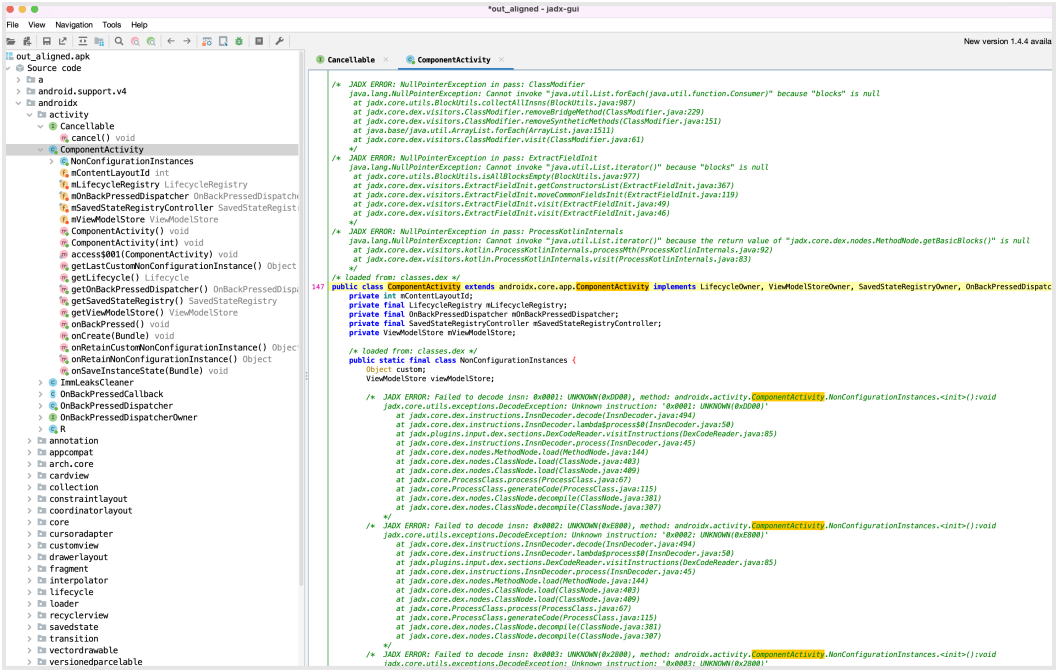

그림 19. 디컴파일 실패

m. 메모리 위변조 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'check_mem_scanner'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 루팅 디바이스
    - 루팅 디바이스가 없는 경우 루팅 권한을 사용할 수 있는 에뮬레이터로 대체 가능합니다.
  - ADB와 Frida가 설치된 분석용 PC
    - Frida는 분석 대상 앱의 메모리 주소를 조작해 다양한 기능을 제공하는 분석 도구입니다.
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)

- **검증 시나리오**

1. AppSuit Cloud가 적용된 앱을 설치합니다.

```
host$ adb install [APK_AAB_FILE_PATH]
```

2. 디바이스 arch에 맞으며, 분석용 pc에 설치된 frida 버전과 동일한 frida-server를 설치합니다.
    - https://github.com/frida/frida/releases 링크에서 설치 가능합니다.
3. 다운받은 frida-server 파일을 디바이스 내부 /data/local/tmp/ 경로에 복사합니다.

```
host$ adb push [FRIDA_SERVER_PATH] /data/local/tmp
```

4. ADB를 통해 디바이스 Shell에 접속합니다.
5. frida-server에 실행권한을 부여합니다.
6. frida-server를 실행합니다.

```
host$ adb shell

device$ cd /data/local/tmp/
device$ chmod a+x [FRIDA_SERVER_FILE]
device$ ./[FRIDA_SERVER_FILE] &
```

7. 앞서 설치한 검증 대상 앱을 실행합니다.
8. 디바이스 Shell에서 실행된 앱의 pid를 가져옵니다.

```
device$ ps -ef | grep [packageName]
    > .. u0_a45   5036 1873   [packageName]
    > .. u0_a45   5053 5036   [packageName]
```

9. 분석용 pc에서 frida를 사용해 device에서 실행중인 앱의 프로세스에 attach 합니다.

```
host$ frida -U -p 5036
```

10. 앱이 종료되는 것을 확인합니다.

n. Flutter 암호화 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - (app)build.gradle에 아래 옵션을 추가합니다.

```
appsuit {
    serverOptions {
        host = ' ... '
        id ' ... '
        pw ' ... '

        options = [
            'encrypt_flutter'
        ]
        cloudBuild true
        enableServerBuild true
}
```

- **검증 환경**
  - 파일 분석용 PC
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)
  - AppSuit Cloud를 적용하지 않은 앱 (APK / AAB)

- **검증 시나리오**
  1. AppSuit Cloud를 적용하지 않은 앱과 AppSuit Cloud가 적용된 앱을 각각 unzip합니다.
     - *apktool*로 패키지를 풀어도 무관합니다.
  2. lib - arch 디렉토리 내 libapp.so 파일 존재 여부를 비교합니다.
     - AppSuit Cloud의 Flutter 암호화 기능 적용 이후에는 libapp.so 파일이 숨겨져 APK 내에 노출되지 않습니다.

o. React Native .bundle 암호화 검증

- **빌드**
  - 안드로이드 프로젝트에 AppSuit Cloud 솔루션을 적용합니다.
  - AppSuit rule.pro 파일에 아래의 옵션을 추가합니다.

```
-encrypt_rn_bundle
```

- **검증 환경**
  - 파일 분석용 PC
  - 상기 표기된 [빌드] 에 따라 빌드된 앱 (APK / AAB)
  - AppSuit Cloud를 적용하지 않은 앱 (APK / AAB)

- **검증 시나리오**
  1. AppSuit Cloud를 적용하지 않은 앱과 AppSuit Cloud가 적용된 앱을 각각 unzip합니다.
     - *apktool*로 패키지를 풀어도 무관합니다.
  2. assets - appsuit2 디렉토리 내의 .bundle 파일의 존재 여부를 비교합니다.
     - React Native .bundle 암호화가 적용되지 않은 앱의 경우, assets 디렉토리 내에 .bundle 파일이 존재합니다.
     - .bundle 파일의 이름은 별도 설정이 없는 경우 *index.android.bundle*이며, 프로젝트 설정에 따라 달라질 수 있습니다.

부록

A. AppSuit Plugin(1차) 난독화 적용 옵션

- **logout_std**
  - 설명 : Gradle Console에 AppSuit에 사용되는 옵션과 각종 로그를 표시하는 옵션입니다.
  - 옵션 적용 방법 : -logout_std
- **no_remove_logging**
  - 설명 : Log.e, Log.i 등과 같은 디버깅 정보를 출력해주는 메소드를 제거하는 옵션입니다.
  - 옵션 적용 방법 : -no_remove_logging
  - **주의사항 : 해당 옵션은 배포 시 제외해야하는 옵션입니다.**
- **maxHeap**
  - 설명 : 앱의 크기가 커 메모리 부족으로 인한 문제 발생 시, Heap의 크기를 늘리기 위해 사용되는 옵션입니다.
  - 적용 예시 : -maxHeap 4096 (단위 mb)
- **logout**
  - 설명 : Gradle Console에 출력되는 내용을 사용자가 지정한 텍스트 파일에 저장하는 옵션입니다.
  - 옵션 적용 방법 : -logout D:\buildlog.txt
  - **주의사항 : 텍스트 파일을 해당 경로에 미리 생성해야합니다.**
- **use_d8**
  - 설명 : d8 컴파일러 호환을 위한 옵션입니다.
  - 옵션 적용 방법 : -use_d8
- **sourcecompat**
  - 설명 : 자바 최신 버전과의 호환성을 위한 옵션입니다.
  - 옵션 적용 방법 : -sourcecompat 1.8
  - **주의사항 : 스틸리언 엔지니어의 요청이 아닌 경우 해당 옵션은 추가하지 않습니다.**
- **targetcompat**
  - 설명 : 자바 최신 버전과의 호환성을 위한 옵션입니다.
  - 옵션 적용 방법 : -targetcompat 1.8
  - **주의사항 : 스틸리언 엔지니어의 요청이 아닌 경우 해당 옵션은 추가하지 않습니다.**
- **use_complex_class_name**
  - 설명 : 사전에 프로가드 및 난독화가 적용되어 있는 경우 AppSuit에서 선점하여 사용하는 a.a 클래스를 a.xlftm으로 변경하는 옵션입니다.
  - 옵션 적용 방법 : -use_complex_class_name
- **no_remove_sourcefile_attr**

    - 설명 : AppSuit 적용 시 삭제되는 디버깅 정보(ex. StackTrace)를 삭제하지 않도록하는 옵션입니다.
    - 옵션 적용 방법 : -no_remove_sourcefile_attr
    - **주의사항 : 해당 옵션 적용 시 보안이 취약해지므로 StackTrace와 같은 디버깅 정보를 사용하는 경우에만 사용해야 합니다.**
- **use_api_desugar**
    - 설명 : java desugar를 지원하는 옵션입니다.
    - 옵션 적용 방법 : -use_api_desugar
    - **주의사항 : 난독화 과정에 desugar 로직이 포함되어 빌드 시 속도저하가 발생할 수 있습니다.**
      desugaring 기능을 사용하는 경우에만 사용 가능합니다.
- **merge_lib_proguard_rules**
    - 설명 : 라이브러리들에 포함되어 있는 proguard 룰들을 병합하는 옵션입니다.
      rule.pro 경로에 merged_rules.pro 파일을 생성합니다.
    - 옵션 적용 방법 : -merge_lib_proguard_rules
    - **주의사항 : build.gradle에 minifyEnabled가 true 로 설정되어야 합니다.**
- **sync_lib_proguard_rules**
    - 설명 : 라이브러리들에 포함되어 있는 proguard 룰들을 병합하여 빌드하는 옵션입니다.
      rule.pro 경로에 merged_rules.pro 파일을 생성합니다.
      merge_lib_proguard_rules 를 자동으로 활성화 합니다.
    - 옵션 적용 방법 : -sync_lib_proguard_rules
    - **주의사항 : build.gradle에 minifyEnabled가 true 로 설정되어야 합니다.**
      appsuit 디렉토리 내 rule.pro 파일에 -dumpprofile proguard 옵션이 추가되어야 합니다.
- **obfuscate_with_proguard**
    - 설명 : proguard를 사용하여 식별자 난독화를 수행합니다.
      proguard 산출물을 기반으로 하는 서드파티 도구 사용시에 꼭 필요한 경우에만 활성화 합니다.
      rule.pro 경로에 merged_rules.pro 파일을 생성합니다.
      merge_lib_proguard_rules 를 자동으로 활성화 합니다.
    - 옵션 적용 방법 : -obfuscate_with_proguard
    - **주의사항 : build.gradle에 minifyEnabled가 true 로 설정되어야 합니다.**
      appsuit 디렉토리 내 rule.pro 파일에 -dumpprofile proguard 옵션이 추가되어야 합니다.
      식별자 난독화 예외 처리가 필요한 경우, Proguard 및 r8 룰 대신 AppSuit 룰에 해당 파일을 추가해야합니다.+ Proguard 및 r8 룰에 난독화 예외 처리가 된 경우, AppSuit 룰에 무관하게 해당 예외 처리가 적용되는 점 주의 바랍니다.
- **encrypt_rn_bundle**
    - 설명 : React Native 앱에서 사용하는 .bundle 파일을 암호화하는 기능입니다.
    - 옵션 적용 방법 : -encrypt_rn_bundle

- **주의사항 : 옵션 사용 시 [2-e ~ 2-i] 항목을 참조하여, .bundle 로딩에 관련된 코드를 수정해야 합니다.**

B. AppSuit Plugin(1차) 난독화 예외처리 옵션

- **keep**
  - 설명 : 클래스 및 메소드의 이름을 난독화하지 않는 옵션입니다.
  - 적용 예시
    - 예시1 : -keep class android.support.** { *; }
      → *_android.support* 이하의 클래스 및 필드, 메소드의 이름을 난독화 하지 않습니다.
    - 예시2 : -keep class * extends android.app.Activity { *; }
      → *android.app.Activity*를 상속받는 클래스와 그 클래스에 있는 필드, 메소드들의 이름을 난독화하지 않습니다.
    - 예시3 : -keep class com.github.** { public *; }
      → *com.github* 이하의 모든 클래스 중 접근제어지시자가 *public*인 필드, 메소드만 이름 난독화하지 않습니다.
- **keepstrings**
  - 설명 : 클래스에 있는 문자열을 암호화하지 않는 옵션입니다.
  - 적용 예시
    - 예시 : -keepstrings class okhttp3.** { *; }
      → *okhttp3* 이하의 모든 클래스에 있는 문자열을 암호화하지 않습니다.
- **keepbare**
  - 설명 : 클래스를 암호화하지 않는 옵션입니다.
  - 적용 예시
    - 예시 : -keepbare class com.stealien.test.MainActivity { *; }
      → *com.stealien.test* 패키지의 *MainActivity*를 \암호화 하지 않습니다.
- **keepreflect**
  - 설명 : API 숨김 기능을 적용하지 않는 옵션입니다.
  - 적용 예시
    - 예시 : -keepreflect class com.google.** { *; }
      → *com.google* 패키지 내의 모든 클래스에 대해 API 숨김을 진행하지 않습니다.
- **keepflow**
  - 설명 : 코드 제어 흐름을 변경하지 않는 옵션입니다.
  - 적용 예시
    - 예시 : -keepflow class com.retrofit2.** { *; }
      → *com.retrofit2* 이하의 모든 클래스의 코드 제어 흐름을 변경하지 않습니다.

- **allowappsuit**
  - 설명 : 난독화 대상이 아닌 클래스를 난독화 합니다.
  - 적용 예시
    - 예시 : -allowappsuit class com.retrofit2.** { *; }
      → *com.retrofit2* 를 난독화 합니다.
  - **주의사항 : processLibrary 옵션이 false 일때만 동작합니다.**
- **forceencrypt**
  - 설명 : 해당 클래스를 암호화하는 옵션입니다.
  - 적용 예시
    - 예시 : -forceencrypt class com.example.** { *; }
      → *com.example* 이하의 모든 클래스를 암호화합니다.
  - **주의사항 : keepbare 옵션보다 우선순위가 낮습니다.**
    - 예시 : -forceencrypt class com.a.** { *; }
      -keepbare class com.a.b { *; }
      → *com.a.b* 클래스를 제외한 *com.a* 내의 모든 클래스를 암호화합니다.

C. AppSuit에서 지원하는 proguard 예외처리 옵션 목록

- **keepnames**
- **keepclasseswithmembers**
- **keepclasseswithmembernames**
- **keepclassmembernames**
- **keepclassmembers**
- **if**

D. AppSuit cloud-natv(2차) 적용 옵션

- **check_root**
  - 설명 : 루팅 탐지 기능 옵션입니다.
  - 상세 설명 : 단말이 루팅 되었는지 검사합니다.
- **detect_magisk**
  - 설명 : magisk 탐지 기능 옵션입니다.
  - 상세 설명 : 단말이 magisk로 루팅되었는지 검사합니다.
  - **참고 사항 : check_root 활성화 시 사용 가능합니다.**
- **check_integrity**
  - 설명 : 위변조 탐지 기능 옵션입니다.
  - 상세 설명 : apk/aab 내부 파일들(.dex, .so, etc.)이 위조/변조되었는지 검사합니다.
- **check_emul**
  - 설명 : 에뮬레이터 탐지 기능 옵션입니다.
  - 상세 설명 : 단말이 에뮬레이터(실제 단말이 아닌 가상화 프로그램)인지 검사합니다.
- **check_debugging**
  - 설명 : 디버거 탐지 기능 옵션입니다.
  - 상세 설명 : 디버거(gdb, frida, etc.)의 존재/사용 여부를 탐지합니다.
  - **참고 사항 : Frida 탐지 기능 사용을 위해서는 AndroidManifest.xml에 아래 Permission을 추가해야합니다.**

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

- **prevent_debugging**
  - 설명 : 디버깅 방지 기능 옵션입니다.
  - 상세 설명 : 디버거가 사용하는 자원을 선점해 디버깅을 하기 어렵게 만듭니다.
- **prevent_adb**
  - 설명 : *adb* 방지 기능 옵션입니다.
  - 상세 설명 : *adb*를 사용한 디버깅을 방지합니다.
- **encrypt_so**
  - 설명 : so library 암호화 기능 옵션입니다.
  - 상세 설명 : 라이브러리(lib/*.so) 파일들을 암호화합니다.

- **self_protect**
  - 설명 : AppSuit 솔루션 자체 보호 기능 옵션입니다.
  - 상세 설명 : AppSuit 솔루션 코드가 포함된 파일(*libAppSuit.so*)을 암호화합니다.
- **prevent_decompile**
  - 설명 : 디컴파일 방지 기능 옵션입니다.
  - 상세 설명 : jadx, dex2jar 디컴파일러를 통해 소스 코드를 복원하지 못하게 합니다.
  - **주의 사항 : cloud-natv v23.0.2 미만의 모듈을 사용하시는 경우, 해당 기능은 Jetpack Compose와 같이 사용하실 수 없습니다.**
- **check_mem_scanner**
  - 설명 : 메모리 해킹 방지 옵션입니다.
  - 상세 설명 : frida와 같이 메모리 위변조 기반의 분석 및 해킹 도구들이 메모리를 변조하였는지 검사합니다.
- **encrypt_flutter**
  - 설명 : Flutter 앱 전용 암호화 옵션입니다.
  - 상세 설명 : Flutter 앱 내 dart로 작성된 코드를 암호화합니다.
  - **참고 사항 : 암호화 데이터는 압축률이 낮아, 해당 기능 사용 시 앱 크기가 커질 수 있습니다.**

E. AppSuit cloud-natv(2차) 예외처리 옵션

※ 해당 옵션들은 appsuit 디렉토리 내 rule.pro 파일에 작성되어야 합니다.

- **exclude_resources**
  - 설명 : check_integrity 옵션이 활성화 되었을 때, 위변조 여부를 검사하지 않아도되는 파일을 지정하는 옵션입니다.
    주로 동적으로 업데이트되는 파일들을 예외처리하는데 사용됩니다. (정규식으로 사용 가능합니다.)
  - 옵션 적용 방법

```
-exclude_resources {
    META-INF/**
    AndroidManifest.xml
    [추가 예외처리할 파일]
}
```

  - **주의사항 : META-INF/** 과 AndroidManifest.xml은 항상 기입해주셔야 합니다.**
    무분별하게 wildcard(*)를 사용하는 경우, 보안성이 매우 떨어질 수 있습니다.

- **except_so_list**
  - 설명 : encrypt_so 옵션이 활성화 되었을 때, 암호화를 하지 않아도 되는 so 파일을 지정하는 옵션입니다.
    일반적으로 타 보안 솔루션과 충돌이 일어날 때, 이를 핸들링하기위한 방법의 일환으로 사용됩니다.
  - 옵션 적용 방법

```
-except_so_list {
    [libname.so]
}
```

  - 예시

```
-except_so_list {
    libExample1.so
    libExample2.so
}
```

→ *_libExample1.so* 와 *libExample2.so*를 암호화 하지 않습니다.

- **주의사항 : 기존 premium 빌드 서버 옵션과 상이합니다.**
  따라서 해당 옵션을 사용하던 기존 고객사의 경우, 수정이 필요합니다.

F. AppSuit Open Page

- (주)스틸리언에서는 솔루션의 최신화 및 안정적인 지원을 위해 Open Page 를 제공하고 있습니다.
  AppSuit Cloud 적용 시 필요한 파일 및 FAQ 등 은 아래 스틸리언 Open Page 참고 부탁드립니다.

```
https://appsuit.notion.site/AppSuit-Premium-5eafc8bde42246a7ae8542e45fb1ccd7
```

- 지원 환경
  - AppSuit Cloud 를 적용하기 위해 필요한 환경에 대한 정보가 작성되어 있습니다.
- ReleaseNote
  - AppSuit Cloud 적용 시 필요한 AppSuit Plugin (1차) 과 cloud-natv (2차) 에 대한 정보가 작성되어 있습니다.
- Code Snippets & Guides
  - AppSuit Cloud 사용 시 함께 사용 가능한 제품에 대한 정보와 가이드가 작성되어 있습니다.
- Trouble Shooting
  - FAQ에 작성되어 있지 않는 이슈 발생 시 스틸리언 엔지니어에게 전달해주실 정보가 작성되어 있습니다.
- FAQ
  - AppSuit Cloud 적용 시 발생하는 이슈에 대한 해결방안이 작성되어 있습니다.

(주) 스틸리언

TEL : 02-2088-4835
E-mail : se@stealien.com
주소 : 서울특별시 용산구 원효로90길 11, 더프라임타워 업무동 12층
홈페이지 : https://www.stealien.com/main