PROMON

Shield Gradle plugin for Android

Version 2.1.0

Table of Contents

1. Introduction 1
2. Installation and setup 1
    2.1. Include dependency on plugin file 1
    2.2. Apply plugin 2
    2.3. Configuration 3
    2.4. Test run 5
    2.5. Output 5
3. How it works 6
4. Requirements 6
5. Obfuscation and mapping.txt 7
6. Configuration 7
    6.1. Plugin specific options (`pluginOptions`) 9
    6.2. Variants 9
    6.3. Examples 12
        6.3.1. Basic example 12
        6.3.2. Complex example 12
        6.3.3. Disable automatically sending proguard-mapping to Shielder 14
7. Known issues 14
8. Changelog 15

1. Introduction

Shield Gradle plugin for Android is a plugin for *Gradle* that will automatically apply shield to the app that is being built.

The plugin supports both standard groovy-style `build.gradle` and kotlin-style `build.gradle.kts`. All examples listed in this documentation will be shown in both groovy and kotlin.

Both `.apk` and `.aab` (app bundles) are supported and will be automatically shielded when the plugin is enabled.

2. Installation and setup

To start using Shield Gradle plugin for Android a few steps are necessary.

- Include the shield-gradle-plugin-2.1.0.jar as a dependency into gradle
- apply the plugin to the project
- finally, configure the plugin

The Shield Gradle plugin for Android is distributed as a .jar file.

2.1. Include dependency on plugin file

Include the plugin as dependency within the `buildscript` container (typically found in the root project's `build.gradle`). The path should be an absolute path that points to the provided `shield-gradle-plugin-2.1.0.jar` file.

Example 1. Include dependency in root project `buildscript` (root project `./build.gradle`)

Groovy (`./build.gradle`)

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    dependencies {
        classpath files('path/to/shield-gradle-plugin-2.1.0.jar') ①
    }
}

plugins {
    id 'com.android.application' version "7.4.1" apply false ②
    ...
}
```

Kotlin (`./build.gradle.kts`)

```
// Top-level build file where you can add configuration options common to all subprojects/modules.
buildscript {
    dependencies {
        classpath(files("path/to/shield-gradle-plugin-2.1.0.jar")) ①
    }
}

plugins {
    id("com.android.application") version "7.4.1" apply false ②
    ...
}
```

① Path should refer to the downloaded `shield-gradle-plugin-2.1.0.jar` file

② The Android Gradle plugin 4.2.0 or higher is required.

This version of the Shield Gradle plugin for Android has been tested against android gradle plugin version 4.2.*, 7.0.*, 7.1.*, 7.2.*, 7.3.*, 7.4.*

2.2. Apply plugin

In app module's `build.gradle` apply the plugin with the plugin enclosure

Example 2. Apply plugin in app project build file (`./app/build.gradle[.kts]`)

Groovy (`./app/build.gradle`)

```
plugins {
    id 'com.android.application'
    // ... other plugins
    id 'no.promon.shield.shieldplugin'
}

android {
    // ...
}
```

Kotlin (`./app/build.gradle.kts`)

```
plugins {
    id("com.android.application")
    // ... other plugins
    id("no.promon.shield.shieldplugin")
}

android {
    // ...
}
```

2.3. Configuration

Setup basic configuration. The plugin creates the new `shieldConfig` container which can be used to configure Shield.

Example 3. Configure Shield Gradle plugin for Android (`./app/build.gradle[.kts]`)

Groovy (`./app/build.gradle`)

```
plugins {
    id 'com.android.application'
    // ... other plugins
    id 'no.promon.shield.shieldplugin'
}

android {
    // ...
}

shieldConfig {
    shielderJar "/path/to/Shielder.jar"
    config "/path/to/config.xml"
}
```

Kotlin (`./app/build.gradle.kts`)

```
plugins {
    id("com.android.application")
    // ... other plugins
    id("no.promon.shield.shieldplugin")
}

android {
    // ...
}

shieldConfig {
    shielderJar = file("/path/to/Shielder.jar") ①
    config = file("/path/to/config.xml") ①
}
```

① Kotlin's properties does not permit overloaded setters (with different type than the property itself). Thus, it will only accept a `File` object

For the plugin to work at least the `shielderJar` and `config` options must be specified. This should be put into the app module's `build.gradle` (in a typical project found at `app/build.gradle`).

2.4. Test run

Try building the application.

The plugin will inject Shield into the application in one of the final gradle steps. This should look something like the following in the gradle console window in android studio:

```
Executing tasks: [:app:assembleDebug]

> Task :app:preBuild UP-TO-DATE
> Task :app:preDebugBuild UP-TO-DATE
> Task :app:compileDebugAidl NO-SOURCE
[...]
> Task :app:mergeProjectDexDebug
> Task :app:packageDebug

> Task :app:shielderApkDebug
About to run shielder with args: ...

 * Shielding app: MyApplication/app/build/outputs/apk/debug/app-debug.apk
 * Decoding app code - using promon engine
 * Preparing Shield SDK
[...]
 * Writing shield data
 * Encoding app - using promon engine
 * Signing file: shielded-app-debug.apk
 * Cleaning up
 * Success. New app: MyApplication/app/build/outputs/shield/debug/apk/shielded-app-
debug.apk
> Task :app:assembleDebug

BUILD SUCCESSFUL in 34s
27 actionable tasks: 17 executed, 10 up-to-date
```

2.5. Output

Shielder will generate new `.apk` or `.aab` file that by default are placed in `/app/build/outputs/shield/`

> **WARNING** This is different from earlier Shield Gradle plugin for Android versions, which replaced the existing apk file.

The output location can be changed using the pluginOption `outputLocation`

Example 4. Set output location of artifacts (`./app/build.gradle[.kts]`)

Groovy (`./app/build.gradle`)

```
shieldConfig {
    // ...
    pluginOptions {
        outputLocation project.layout.buildDirectory.dir("outputs/shield/") ①
    }
}
```

Kotlin (`./app/build.gradle.kts`)

```
shieldConfig {
    // ...
    pluginOptions {
        outputLocation = project.layout.buildDirectory.dir("outputs/shield/") ①
    }
}
```

① Must be an `org.gradle.api.file.Directory`. See Plugin specific options (`pluginOptions`) for more information.

3. How it works

Shield Gradle plugin for Android creates a task that will run as the last step when building an application. This task analyses the gradle configuration and then configures and runs `Shielder.jar` that wraps the application with Shield.

Internally, this is using the artifact transformation api provided in the android gradle plugin. Example usage of this api can be found at https://github.com/android/gradle-recipes

The task runs after the PackageApplication task, and works on the compiled `.apk` or `.aab` file. The plugin detects and supplies the compiled app `.apk` or `.aab` file automatically, and it does not need to be moved or provided.

The plugin respects flavors and buildTypes, and attempts to use signing information provided there.

Shield Gradle plugin for Android also considers the `splits` option, and will wrap each split-output accordingly. If abi-split is specified, shielder will be instructed to include only the relevant architecture. If splits isn't specified, then the plugin attempts to consider the `abiFilter` option, if specified.

4. Requirements

Shield Gradle plugin for Android requires `com.android.tools.build:gradle:4.2.0` or higher.

5. Obfuscation and mapping.txt

Crashlytics and similar crash-reporting tools need mapping.txt to de-obfuscate stack traces. Crashlytics will automatically upload mapping.txt from proguard for this through its gradle plugin. Unfortunately, Crashlytics and some similar tools does not allow manual upload of these files.

If both proguard and shield obfuscation is enabled then Shield Gradle plugin for Android will provide Shielder with the proguard's mapping.txt which Shielder then translates into a combined mapping.txt. Shield Gradle plugin for Android will then attempt to override the proguard's mapping.txt with this combined mapping before Crashlytics uploads this file to its servers.

CAUTION Crashlytics specifically checks the proguard tasks, and will not run if proguard is not enabled. To persuade Crashlytics plugin to upload shield obfuscated mappings, proguard **must** be enabled (`minifyEnabled true`) even if only shield obfuscation is desired.

Crashlytics stores and uploads the mapping file through its two tasks `:app:crashlyticsStoreDeobs<flavor>`, `:app:crashlyticsUploadDeobs<flavor>`. Shield Gradle plugin for Android will attempt to detect these tasks, and replace the proguard mapping.txt file before these run.

TIP It's possible to enable/run Proguard without using any of its features.

app/build.gradle

```
release {
    minifyEnabled true
    proguardFiles 'proguard-rules.pro'
}
```

proguard-rules.pro

```
 # Disable everything
-dontshrink
-dontoptimize
-dontpreverify

-keep class ** {
    *;
}
-dontwarn **
-dontnote **
```

6. Configuration

Shield Gradle plugin for Android is configured in your app module's `build.gradle` file using the

`shieldConfig` container. This container should be placed in the top level, or within the `android` container.

Within `shieldConfig` container there are two types of options- `options` which are parameters that will be passed directly to shielder, and `pluginOptions` which are options specifically for the gradle plugin.

Example 5. Configure Shield Gradle plugin for Android (`./app/build.gradle[.kts]`)

Groovy (`./app/build.gradle`)

```
plugins {
    id 'com.android.application'
    // ... other plugins
    id 'no.promon.shield.shieldplugin'
}

android {
    // ...
}

shieldConfig {
    pluginOptions {
        option "shielderJar", "/path/to/Shielder.jar"
    }
    options {
        option "config", "/path/to/config.xml"
    }
}
```

Kotlin (`./app/build.gradle.kts`)

```
plugins {
    id("com.android.application")
    // ... other plugins
    id("no.promon.shield.shieldplugin")
}

android {
    // ...
}

shieldConfig {
    pluginOptions {
        option("shielderJar", "/path/to/Shielder.jar")
    }
    options {
        option("config", "/path/to/config.xml")
    }
}
```

All the specified `options` will be passed through directly to Shielder, and follows all the same formats as if invoked directly as a command.

TIP All the available options are specified in the documentation for Shielder, and can also be found by running `java -jar Shielder.jar`.

6.1. Plugin specific options (`pluginOptions`)

Command	Description	Example
`shielderJar`	Specify the path to the `Shielder.jar` file supplied by Promon	`shielderJar "home/myName/Downloads/shield/wrap/Shielder.jar"`
`enabled`	Enable/disable Shield wrapping of this specific build	`enabled false`
`autoSigning`	Enable/disable plugin from retrieving and passing along signing information to shielder.	`autoSigning false`
`autoProfile`	Enable/disable plugin from automatically setting profile == debug when building debug builds	`autoProfile false`
`autoMapping`	Enable/disable plugin from retrieving and passing along mapping.txt to shielder (as --proguard-mapping option)	`autoMapping false`
`autoAbiFilters`	Enable/disable plugin from retrieving and passing along abi-filters	`autoAbiFilters false`
`outputLocation`	The location where the plugin will put artifacts. Please note this requires a `org.gradle.api.file.Directory` value. Use `project.layout.buildDirectory.dir(String)` or `project.layout.projectDirectory.dir(String)` to get an appropriate value get an appropriate value.	`outputLocation project.layout.buildDirectory.dir("outputs/shield/")`

6.2. Variants

Variant-specific configurations can be specified like so:

Groovy

```
android {
    // ....
    flavorDimensions "version"
    productFlavors { ①
        demo {
            dimension "version"
        }
        full {
            dimension "version"
        }
        premium {
            dimension "version"
        }
    }
}

shieldConfig {
    shielderJar "/path/to/Shielder.jar"
    config "/path/to/config.xml"
    enabled true
    options { ②
        option("no-arch-armeabi-7a")
    }
    pluginOptions "debug", {  ②
        enabled false ③
    }
    options "demoRelease", {  ②
        option("no-arch-mips")
    }
    options "premium", {  ②
        option("config", "/path/to/different/config.xml")
    }
}
```

Kotlin

```
android {
    // ....
    flavorDimensions.add("version")
    productFlavors { ①
        create("demo") {
            dimension = "version"
        }
        create("full") {
            dimension = "version"
        }
        create("premium") {
            dimension = "version"
        }
    }
}

shieldConfig {
    shielderJar = file("/path/to/Shielder.jar")
    config = file("/path/to/config.xml")
    enabled = true
    options { ②
        option("no-arch-armeabi-7a")
    }
    pluginOptions("debug") {  ②
        enabled = false ③
    }
    options("demoRelease") {  ②
        option("no-arch-mips")
    }
    options("premium") {  ②
        option("config", "/path/to/different/config.xml")
    }
}
```

① Specify flavors (`productFlavors`) and build types as usual in `android` container.

② `options` and `pluginOptions` both support variants.

- Specifying a variant is optional.
- If nothing is specified then "default" is assumed, which is active across all variants.
- If specified, then the options specified therein will only activate on corresponding variants. Any combination of flavor names and build types are accepted.

③ `enabled false` can be used to disable shield wrapping for some variants. In this case it means that the plugin will not run shielder when/if building debug

TIP Even if your project doesn't specifically use flavors, the default buildTypes will be available (`debug` and `release` by default). These and any custom buildType can be used in the variant filter, regardless or in combination with flavors.

6.3. Examples

6.3.1. Basic example

```
java -jar Shielder.jar app-release.apk --storepass 123456 --digestalg SHA256 --profile
release --keyname testAlias --sigalg SHA256withRSA --keystore testKeystore.jks
--keypass 123456 --config config.xml --output app-release.apk
```

is roughly equivalent to the following shieldConfig configuration

Example 6. Project build.gradle (`app/build.gradle[.kts]`)

Groovy

```
shieldConfig {
    shielderJar "/path/to/Shielder.jar"
    config "/path/to/config.xml"
    options {
        option("digestalg", "SHA256")
        option("sigalg", "SHA256withRSA")
    }
}
```

Kotlin

```
shieldConfig {
    shielderJar = file("/path/to/Shielder.jar")
    config = file("/path/to/config.xml")
    options {
        option("digestalg", "SHA256")
        option("sigalg", "SHA256withRSA")
    }
}
```

TIP If gradle is configured to sign the application then the plugin will include that signing information automatically and forward it to Shielder.

6.3.2. Complex example

Groovy

```
shieldConfig {
    shielderJar "/path/to/Shielder.jar"
    rules "/path/to/rules-file.cfg" ①
    options "debug", {
        option("config", "/path/to/debug-config.xml") ②
        option("arch", "x86") ③
    }
    options "release", {
        option("config", "/path/to/release-config.xml")  ②
        option("rules", "/path/to/release-rules-file.cfg")  ①
        option("obfuscate", "on") ④
    }
}
```

Kotlin

```
shieldConfig {
    shielderJar = file("/path/to/Shielder.jar")
    rules = file("/path/to/rules-file.cfg") ①
    options("debug") {
        option("config", "/path/to/debug-config.xml")  ②
        option("arch", "x86") ③
    }
    options("release") {
        option("config", "/path/to/release-config.xml")  ②
        option("rules", "/path/to/release-rules-file.cfg")  ①
        option("obfuscate", "on") ④
    }
}
```

① Setting `shieldConfig.rules` is equivalent to `shieldConfig.options.option("rules")`. In this example, the value of `"/path/to/release-rules-file.cfg"` will be used if building with the `release` buildType, and with `"/path/to/rules-file.cfg"` as value in all other cases (including debug, and any other custom buildType)

② `shieldConfig.config` and `shieldConfig.options.option("config")` are equivalent. However, `shieldConfig.options` allow you to define an applicable variant - here "debug" and "release" -, allowing you to specify option values that only applies for corresponding flavors and build types. Here it is used to provide a "debug config" for when building a debug build (debug buildType of the app), and a "release config" for the release build of the app.

③ Only add x86 native Shield libraries onto the app when shielding when building debug version of the app.

④ Enables obfuscation of the app, only when building release

6.3.3. Disable automatically sending proguard-mapping to Shielder

Groovy

```
shieldConfig {
    shielderJar "/path/to/Shielder.jar"
    config "/path/to/config.xml"
    pluginOptions "release", { ①
        autoMapping = false  ②
    }
}
```

Kotlin

```
shieldConfig {
    shielderJar = file("/path/to/Shielder.jar")
    config = file("/path/to/config.xml")
    pluginOptions("release") { ①
        autoMapping = false ②
    }
}
```

① `pluginOptions` just like `options` can specify a variant filter to apply those options only to the specified variant. In this case, "release" is specified and so autoMapping is set to false only for the "release" buildType.

② Setting autoMapping to false means the Shield Gradle plugin for Android will *not* automatically resolve the proguard mappings.txt file and pass it along to Shielder.

7. Known issues

- Some earlier versions of Android Studio (specifically those using AGP version 4.1.*) have an issue where using the green "play" button to build the app and deploy to device will install the wrong apk to device. It's still shielding properly, however Android Studio simply looks at the initial artifact instead of the last one coming out of the transformations. This has been fixed in more recent versions of AS.
- This plugin uses some APIs from AGP that are marked as @Incubating in AGP version 4.1.* and 4.2.*. This plugin implements a compatibility layer that will select functionality tailor-made for each version to ensure broad compatibility. This compatibility layer can be seen in the log/console window as

```
Shielder plugin configured with compatibility layer: class
no.promon.shield.gradleplugin.wrappers.v70.CompatibilityCheck
```

8. Changelog

v2.1.0

- [SGP-39] Support for AGP 7.3.x + Gradle 7.4. With that AGP version an intermediate, signed app bundle gets a file name like `app/build/intermediates/bundle/signBundle/out` which Shielder Gradle Plugin now handles.
- [SGP-40] Fix handling of output option like `option("output", "my-app.aab")`.
- [SGP-41] Remove support for AGP 4.1+.
- [SGP-42] Add support for AGP 7.4.x + Gradle 7.5
- [SGP-44] Add support for specifying the name/path to the output mapping.txt through `option("output-mapping", "my-mapping.txt")`.

v2.0.1

- Fix handling of arguments with spaces.

v2.0.0

- Internal testing and building improvements.

v2.0-beta02

Various QOL improvements and fixes.

- [SGP-27] Fixed issue related to missing dimension on flavor
- [SGP-29] Prevent tasks from registering if plugin is disabled - fixes artifact system in android gradle plugin from getting confused
- [SGP-31] Fixed an issue that would bundle in Kotlin stdlib. Removed usage of kotlin-reflect.

v2.0-beta01

Plugin rewritten in kotlin.

- Added support for kotlin build scripts (`build.gradle.kts`).
- Added support for app bundles.
- Added support for new transformation API from AGP.