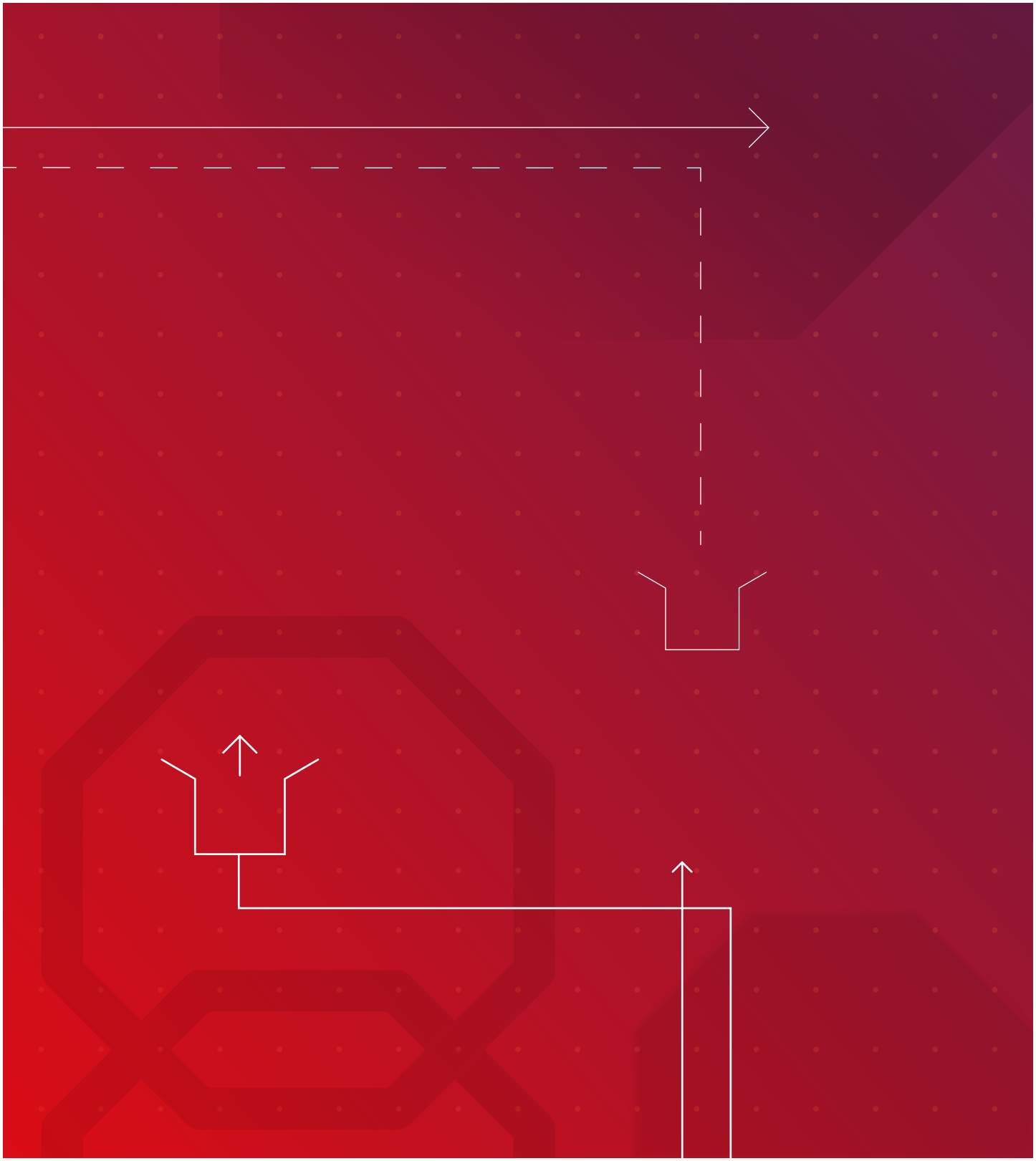
Avira | OEM

1 November 2022

Anti-malware SDK v.3.2

for Android (MAVAPI)



Contents

1 Introduction....4

2 Package Content....4
2.1 Examples....4
2.2 Essential information regarding size of binaries and data....5
2.3 Third-party libraries....6

3 Upgrading from version 2.x to 3.x....7

4 Integrating MavapiLibrary....8
4.1 Description....8
4.2 Interface....9
4.3 Configuration....10
4.4 Usage....10
4.5 Notes and known limitations....10

5 Integrating LocalScanner....11
5.1 Description....11
5.2 Interface....11
5.3 Configuration....13
5.4 Usage....16
5.5 Notes and known limitations....17

6 Integrating ProtectionCloud....18
6.1 Description....18
6.2 Interface....19
6.3 Configuration....21
6.4 Usage....22
6.5 Notes and known limitations....23

7 Integrating AVKCCert....24
7.1 Description....24
7.2 Interface....24
7.3 Configuration....24
7.4 Usage....25
7.5 Notes and known limitations....25

8 Integrating Updater....25
8.1 Description....25
8.2 Interface....26
8.3 Configuration....26
8.4 Usage....28
8.5 Notes and known limitations....28

9 Android compatibility....28

9.1 Android 5 to 9.... 28
9.2 Android 10.... 29
9.3 Android 11.... 30
9.4 Android 12.... 31

10 Licensing.... 31

11 Contact information.... 32
11.1 Support services.... 32
11.2 Contact.... 32

1 Introduction

This document is an easy introduction to Anti-malware SDK for Android (aka MAVAPI), its interface, and to a step by step integration tutorial.

For accessibility reasons, the tutorial uses `Android Studio` for setting up a simple application based on MAVAPI.

Note: If you are currently using a MAVAPI version 2.x and plan to migrate to MAVAPI 3.x, please see chapter Upgrading from version 2.x to 3.x

The SDK library has the following components:

- MavapiLibrary
- LocalScanner
- ProtectionCloud - cloud hash check
- Updater
- AVKCCert - whitelist certificate check

2 Package Content

Path	Information
./bin/mavapi-*.aar	MAVAPI modules (release version), for ARM and x86
./docs/Anti-malware SDK for Android (MAVAPI).pdf	The current document
./docs/Change Log Anti-malware SDK for Android (MAVAPI).pdf	Describes the changes between versions
./docs/html/*	Documentation generated for the MAVAPI interface
./examples/*	Multiple examples showing how MAVAPI can be integrated
./legal/*	License files for third-party libraries

2.1 Examples

The package contains multiple projects, split into 3 types: basic, advanced and kotlin. The scope of these projects is a better understanding on how to use Mavapi.

All projects have the same Mavapi AAR library integrated into them by using a relative path `../../../../bin/mavapi.aar`. Therefore, if a project or the *mavapi.aar* file is moved to a different path, the library path needs to be updated.

The applications can be opened with Android Studio. You should only:

- Add the key file *hbedv.key* to `./<example>/app/src/main/assets/antivirus/`
- Add to the `./<example>/local.properties` file:
  - `LocalScannerProductCode` with the product code received, required for using LocalScanner, e.g.:

    `LocalScannerProductCode=12345` — numeric, lenght might vary
  - `ProtectionCloudAPIKey` with the API key received, required for using ProtectionCloud, e.g.:

    `ProtectionCloudAPIKey=01234567890abcdef01234567890abcdef01234567890abcdef01234567890ab` — hex, lenght must be 64

The example is now ready to run.

Example path	Description
basic/MavapiBasicScan	Example on how to integrate Mavapi and scan some APKs. Set to scan the first 10 APKs found in PackageManager list. (Single thread scan)
basic/MavapiConfiguration	Example on how to set different configurations. (Single thread scan)
basic/MavapiBasicWithMaven	Example on how to set a Local Maven Repository for the Mavapi Library
advanced/MavapiScannerCallbacks	Example on how to use LocalScannerCallbacks and LocalScannerCallbackData. Set to scan the first 10 APKs found in PackageManager list. (Single thread scan)
advanced/MavapiDirectoryScan	Example on how to scan recursively a directory. Set to scan external storage. (Single thread scan)
advanced/MavapiMultiThreadedScan	Example of multi-threaded usage of Mavapi. Set to scan the first 10 APKs found in PackageManager list. (Multithread scan)
advanced/MavapiCloudHashCheck	Example on how to use ProtectionCloud to check all APKs found in PackageManger list. (Single thread scan)
advanced/MavapiCloudHashCheckDirectory	Example on how to use ProtectionCloud to check all APKs found in storage. (Single thread scan)
advanced/MavapiAdvancedScan	Example on how to use both, ProtectionCloud and LocalScanner. (Single thread scan)
advanced/MavapiPerformanceTest	Example on how to test the product performance with different scenarios. (Single thread scan)
kotlin/KotlinBasicExample	Example on how to integrate Mavapi and scan with LocalScanner.
kotlin/KotlinAdvancedExample	Example on how to integrate Mavapi and scan with ProtectionCloud and LocalScanner.

Note: For logging the examples, the ProtectionCloud result "Unknown" gets mapped to "Inconclusive". In other words, the log's term "Inconclusive" stands for "Unknown".

2.2 Essential information regarding size of binaries and data

- Engine binary files = ~1.3 MB (1,267 KB on **arm64-v8**, 1,302 KB on **armeabi-v7a**, 1,342 KB on **x86**, 1,535 KB on **x86-64**)
- Engine data files = 1.8 KB (for **arm64-v8**, **armeabi-v7a**, **x86** and **x86-64**)
- VDF data files = 4.5 MB (for **arm64-v8**, **armeabi-v7a**, **x86** and **x86-64**)
- Mavapi binary files = ~462 KB (530 KB on **arm64-v8**, 465 KB on **armeabi-v7a**, 482 KB on **x86**, 461 KB on **x86-64**)

Note: With time, there will be changes to the binary and data files. Thus, the sizes will vary.

These specified sizes refer to:

- Mavapi v3.0.0
- Engine v8.3.62.232
- VDF v8.15.118.8

Mavapi binary files — currently consist of:

- libantivirus.so
- libmavapi.so
- libmaven.so

VDF data files — currently consist of:

- +30 files of format xba00000.vdf
- aevdf.dat
- (optional) local0000.vdf — file created on the device resulted from merging the 30+ files in order to speed up the load process

Engine data files:

- aeset.dat — (for each arm64-v8a, armeabi-v7a, x86_64 and x86)

Engine binary files:

- libaecore.so
- libaehelp.so
- libaepack.so
- libaevdf.so
- libaedroid.so
- libaemobile.so
- libaescn.so

Note: With time, there will be changes to the number or name of the files, thus it is recommended not to use the hardcoded name or number.

2.3 Third-party libraries

Mavapi:

Library	Current Version	License type
Retrofit	2.9.0	Apache License 2.0
BigDigits	2.1	Custom
MD5	October 09 2009	Public Domain
SHA2	February 02 2007	Custom
apktool-lib	1.4.4-5	Apache License 2.0
ZipAlign	Customized version	Apache License 2.0
Timber	5.0.1	Apache License 2.0

Engine:

Library	Current Version	License type
bigdigits	2.1	N/A
bsdiff	4.3	N/A
bzip2	1.0.8	N/A
distorm64	1.7.30	N/A
esprima	Customized version	N/A
html_tidy	Customized version	N/A
innosetup	Customized version	N/A
md5	N/A	N/A
memmem	N/A	N/A
mt19937ar	N/A	N/A
nsis_bzip2	N/A	N/A
nsis_zlib	N/A	N/A

Library	Current Version	License type
pstdint	0.1.16.0	N/A
sha2	N/A	N/A
stdintw32	N/A	N/A
strnstr	N/A	N/A
tinf	1.2.1	N/A
tropicssl	Customized version	N/A
unarj	Customized version	N/A
unrar	Customized version	N/A
uthash	2.1.0	N/A
uudecode	N/A	N/A
zlib	1.2.12	N/A

3 Upgrading from version 2.x to 3.x

Mavapi 3.0 comes with incompatible backward changes.

- Internal and external dependencies changed
  - External dependencies:
    - Updated to AndroidX, thus all dependencies are updated to the latest version supported (must still support Android 5)
    - The source code was update to Kotlin, resulting in a new dependency list:

```
implementation 'androidx.core:core-ktx:1.3.2'
implementation 'androidx.room:room-runtime:2.4.2'
kapt           'androidx.room:room-compiler:2.4.2'
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
implementation 'com.squareup.okhttp3:okhttp:4.9.3'
implementation "org.jetbrains.kotlin:kotlin-stdlib:1.4.30"
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.2'
```

  - Internal dependencies:
    - Each component (e.g: `ProtectionCloud`, `LocalScanner`, `Updater`, `AVKCCert`, etc) has its own controller (singleton) and in order to be used, the controller must be initialized successfully, otherwise all calls will fail
    - All controllers are initialized and managed by the library controller `MavapiLibController`
- Most class names changed:
- `MavapiScanner` renamed to `LocalScanner`
  - All classes related to `LocalScanner` were renamed accordingly, e.g.:
    - `MavapiReturnCode` renamed to `LocalScannerErrorCode`
- `MavapiAPC` -> `ProtectionCloud`
  - All classes related to `LocalScanner` were renamed accordingly, e.g.:
    - `Result` -> `ProtectionCloudDetection`
- `Mavapi` -> `MavapiLibController`
  - `MavapiCallbackData` -> `LocalScannerCallbackData`
  - `MavapiScanner.ScannerListener` -> `LocalScannerCallback`

  - `MavapiMalwareInfo` -> `LocalScannerMalwareInfo`

- Package structure changed
  - `APCPackageObserver` no longer exists. The user must create its own observer and use `MavapiLibController.updateLocalCache` to apply the changes.
  - `MavapiConfig` was split for each component:
    - `LocalScannerConfig`
    - `UpdaterConfig`
    - `ProtectionCloudConfig`
    - `AVKCCertConfig`
  - All classes related to a component are placed in the same package (e.g.: for LocalScanner there is the `com.avira.mavapi.localScanner` package).
  - Extract `AVKCCert` to a public component, it can be used as a plug-in, and is no longer used by default by ProtectionCloud. The user must use `Initializer.attachPlugin` to decide what components should use this plug-in (currently, only supported by ProtectionCloud).

4 Integrating MavapiLibrary

4.1 Description

`MavapiLibrary` refers to the whole Mavapi AAR library as well as to the head class of the SDK named `MavapiLibController`.

As mentioned in the introduction, the library contains multiple components/modules. Each component has a controller. The controller can be initialized and retrieved through `MavapiLibController`.

The controllers are singletons, thus they will return the same object on each call, except for cases when the component is re-initialized. Resulting in uninitializing the old object and creating a new one.

Integrating

- Either add the MAVAPI AAR module to the project using the following **Project Structure** settings:
  - Add **New Module** -> Select **Import .JAR/AAR PACKAGE** -> Set path to `mavapi-arm.aar` in File Name
  - Select **app**-> Select **Dependencies** -> **Add** -> **Module Dependency** -> Select *mavapi*
- Or manually add file *mavapi-arm.aar* to the project and add the path to it in `build.gradle`:

```
// ...
dependencies {
 // ...
    implementation fileTree(dir: '<path_to_aar_dir>/', include: ['mavapi-arm.aar'])
 // ...
}
```

- Or even use a Local Maven Repository (follow the Basic Example with Maven Repository):

```
// ...
dependencies {
    // ...
        implementation 'com.avira:mavapi-maven:1.0.0'
    // ...
}
```

Dependencies

- build.gradle:

```
dependencies {
    // ....
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.5.1'
}
```

Additional dependencies will be required for using the other components.

Components

- LocalScanner — used for scanning files localy
- ProtectionCloud — used for checking file hashes with cloud server
- AVKCCert — used for checking APK files certificates and integrity
- Updater — used for updating components (e.g.: virus signatures for engine, whitelist for AVKCCert)

4.2 Interface

`MavapiLibController` interface:

- `initialize(ctx: Context): Initializer`

  Method used to start initializing Mavapi library and its components, as well as to setup the plugins. A default configuration will be set the first time for `MavapiConfig` if one is not given.

- `getLocalScannerController(): LocalScannerController`

  Returns controller for `LocalScanner`. The value returned is singleton and remains unchanged until the component is re-initialized using `MavapiLibController.initialize`

- `getProtectionCloudController(): ProtectionCloudController`

  Returns controller for `ProtectionCloud`. The value returned is singleton and remains unchanged until the component is re-initialized using `MavapiLibController.initialize`

- `getAvkccertController(): AVKCCertController`

  Returns controller for `AVKCCert`. The value returned is singleton and remains unchanged until the component is re-initialized using `MavapiLibController.initialize`

- `getUpdaterController(): UpdaterController`

  Returns controller for `Updater`. The value returned is singleton and remains unchanged until the component is re-initialized using `MavapiLibController.initialize`

- `updateLocalCache(context: Context, packageName: String, action: String)`

  Updates the local cache according to the package name and action given

- `getProductVersion(): String`

  Gets the product version

- `setVerbosity(level: Int)`

  Sets the log verbosity. Does not require any pre-initialization.

`Initializer` interface:

- `add(builder: Builder): Initializer`

  Adds the builder from the configuration for a specific component.

- `attachPlugin(targetCtrlName: String, pluginCtrlName: String): Initializer`

  Attaches the plugin to a specific component.

- `build()`

  Finishes the setup by initializing the specified components and attaches the plugins

4.3 Configuration

The class for configuring `MavapiLibrary` is `MavapiConfig`. The configuration method is based on a builder design pattern, meaning that in order to generate a configuration object, `MavapiConfig.Builder` must be used to set the preferred configuration and to call afterward `.build()` to generate the configuration object.

`MavapiConfig` interface supports the following:

- AssetsPath:

- **Default**: %app_dir%/bin/antivirus/
- **Getter**: MavapiConfig.assertPath
- **Setter**: MavapiConfig.Builder.setAssetsPath()
- **Description**: Path where to extract the assets files

4.4 Usage

Note: Make sure that `MavapiLibrary` is integrated properly into the project. Review Integrating MavapiLibrary - Description

Initialize `MavapiLibrary`:

```
import com.avira.mavapi.*

// ...
fun initializeMavapiLib(applicationContext: Context) {
 // ...
    MavapiLibController.initialize(applicationContext)
     .add(MavapiConfig.Builder(applicationContext))
     .build()

 // ...
}
```

4.5 Notes and known limitations

Mavapi AAR contains:

- 3 AAR files — **mavapi-arm.aar** with support for ARM arch, **mavapi-x86.aar** with support for x86 arch and **mavapi-full.aar** with support for ARM an x86 arch
- **mava binary files** — **libantivirus.so** and **libmavapi.so** compiled specifically for **armeabi-v7a**, **arm64-v8a**, **x86** and **x86_64**, are copied automatically in the native library directory on Android devices when the application is installed.
- **engine binary files** — required by `LocalScanner`, a set of binary files compiled specifically for **armeabi-v7a**, **arm64-v8a**, **x86** and **x86_64**. These files are within the AAR file corresponding to its arch, are copied automatically in the native library directory on Android devices when the application is installed.
- **engine data files** — required by `LocalScanner`, are ***.dat** files for each set of engine binary files. These files can be found in each AAR in **assets/antivirus/<platform>/**, and will be extracted on the device when `MavapiLibrary` is initialized.

Requires:

- **VDF** files (aka virus signatures) — required by `LocalScanner` for engine, multiple files (usually around 30) that are downloaded at runtime and are required by engine.

- ***.key** license key — required by engine, can be added to **assets/antivirus/<platform>/** in order to be extracted automatically (during `MavapiLibrary` initialization) on device.
- **product code** — required by engine
- **API key code** — required by `ProtectionCloud`

Device limitations:

- min Android 5 (API 21)

Note:

- For applications that uses Android Gradle plugin 3.6.0 or higher, if minSdkVersion is higher then 23, then **extractNativeLibs** option must be set to **true** within **AndroidManifest.xml** in order to allow the binary files to be extracted. Source: https://developer.android.com/guide/topics/manifest/application-element#extractNativeLibs

5 Integrating LocalScanner

5.1 Description

`LocalScanner` is the component used for scanning files locally. It requires a set of virus signatures (aka VDF).

Technical information:

- Requires:
  - engine binary files — a set of ***.so** files compiled specifically for that platform (arm64-v8a, x86-64). Should be found within the AAR.
  - engine data files — ***.dat** file specific for that platform. Should be found within the AAR.
  - VDF files — a set ~30 ***.vdf** files that can be either downloaded through `Updater` or copied manually on to device.
  - key file — a ***.key** file
  - product code — a code associated with that key file
- Available update rate: approx. 1-2 update/day

Once the requirements are met, `LocalScanner` can be used multi-threaded. To scan in parallel, multiple scanning instances must be created and 1 for each thread must be used.

Applications should not use more scanning instances than the number of cores present in a device. More threads will use more memory and also, if actively used for longer periods of time may generate high load on the CPU, which may cause the device to throttle the CPU frequency. High memory usage on an Android device may result in the OS terminating applications automatically. A good example for an application would be to have one instance for on-demand scans and another instance for the on-install scans.

5.2 Interface

`LocalScannerController` interface:

- `getInitializationStatus(): InitStatus`

  Gets the initialization status of the controller. If failed, calling the rest of the methods will return an error.

- `createInstance(): LocalScanner`

  Gets instance for scanning. First instance created will also:

  - load and check the engine and virus signatures
  - load libmavapi.so
  - load key file

  Once loaded, it will update information regarding the version, date and expiration.

- `clearInstances()`

  Will clear and destroy the instances created (using `createInstance()`) until that point. Using destroyed instances will result in an error or unknown behavior.

- `getUpdateModule(): Module`

  Returns the update module used by `UpdaterController` to update the virus signatures.

- `unloadLibrary(): Boolean`

  Unloads `libmavapi.so` library to free more memory. Once `createInstance` is called, it will automatically load back the library.

- `engineVersion: String`

  Variable containing the engine version. Is set after the first instance is created or an update is performed.

- `vdfVersion: String`

  Variable containing the vdf version. Is set after the first instance is created or an update is performed.

- `vdfSignatureDate: String`

  Variable containing the vdf signature date. Is set after the first instance is created or an update is performed.

- `keyExpirationDate: Date`

  Variable containing the key expiration date. Is set after the first instance is created or an update is performed.

- `stopAll`

  Stops scanning on all instances.

- `removeInstance:(LocalScanner)`

  Removes a local scanner instance. After destroying all the instances, it will automatically unload the engine and virus signatures (this will help free the memory).

- `getInstancesCount(): Int`

  Returns the number of Local Scanner instances created.

`LocalScanner` interface:

- `getInitStatus(): InitStatus`

  Gets the initialization status. In case the controller was not initialized successfully, as result this instance will not be initialized successfully either.

- `getInitErrorCode(): LocalScannerErrorCodes`

  Gets the error code of the initialization result.

- `scan(path: String): LocalScannerDetectionResult`

Scans a file, should be called from the same thread as object creation (e.g. the same thread where `LocalScannerController.createInstance` was called). The call is blocking until the scan process is finished.

- `stop(): LocalScannerErrorCodes`

  Stops the scan as soon as possible, should be called from a different thread.

Notes:

- The scan will not stop immediately, only after the scan of the current file is finished.

- The callback will still be called to report the results.

- This function is most useful when scanning an archive containing multiple files.

- `setScanCallback(cb: LocalScannerCallback)`

  Sets callbacks for the current instance. Callbacks are called during scan only.

- `setUserCallbackData(data: Any)`

  Sets the user callback data for current instance.

`LocalScannerDetectionResult` interface:

- `errorCode: LocalScannerErrorCodes` — error code in case **failed to scan**, e.g.: no VDFs found, error with engine files, etc.
- `malwareInfo: ArrayList<LocalScannerMalwareInfo>` — scan result

`LocalScannerMalwareInfo` interface:

- `name: String` — malware name
- `type: String` — malware type
- `message: String` — details about the malware

`LocalScannerCallback` interface:

- `onScanComplete(callbackData: LocalScannerCallbackData)`

  Will always be called at the end of the scan.

- `onScanError(callbackData: LocalScannerCallbackData)`

  Might be called multiple times during the scan, e.g. when encountering encrypted files.

5.3 Configuration

The class for configuring `LocalScanner` is `LocalScannerConfig`. The configuration method is based on a builder design pattern, meaning that in order to generate a configuration object, `LocalScannerConfig.Builder` must be used to set the preferred configuration and to call afterward `.build()` to generate the configuration object.

`LocalScannerConfig` interface supports the following:

- EnginePath

- **Default**: %native library directory%
- **Getter**: LocalScannerConfig.enginePath
- **Setter**: N/A
- **Description**: Path to where the engine library files are stored.

- EngineDataPath

- **Default**: %app_dir%/bin/antivirus/
- **Getter**: LocalScannerConfig.engineDataPath
- **Setter**: LocalScannerConfig.Builder.setEngineDataPath
- **Description**: Path to the engine data directory where the engine *.dat files can be found.

- VdfPath

- **Default**: %app_dir%/bin/antivirus/
- **Getter**: LocalScannerConfig.vdfPath
- **Setter**: LocalScannerConfig.Builder.setVdfPath
- **Description**: Path to the VDF directory where the *.vdf files can be found and stored.

- KeyPath

- **Default**: %app_dir%/bin/antivirus/
- **Getter**: LocalScannerConfig.keyPath
- **Setter**: LocalScannerConfig.Builder.setKeyPath
- **Description**: Path to the key directory where the *.key file can be found.

- ArchiveMaxRecursion

- **Default**: 5
- **Getter**: LocalScannerConfig.archiveMaxRecursion
- **Setter**: LocalScannerConfig.Builder.setArchiveMaxRecursion
- **Description**: The maximum allowed recursion within an archive. The higher the archives recursion level, the more open files will be used during the archive processing. A value of "0" means the maximum allowed value (INT64_MAX).

- ArchiveMaxRation

- **Default**: 250
- **Getter**: LocalScannerConfig.archiveMaxRation
- **Setter**: LocalScannerConfig.Builder.setArchiveMaxRation
- **Description**: The maximum allowed decompressing-ratio within an archive. A value of "0" means the maximum allowed value (INT64_MAX).

- ArchiveMaxSize

- **Default**: 1073741824 (1GByte)
- **Getter**: LocalScannerConfig.archiveMaxSize
- **Setter**: LocalScannerConfig.Builder.setArchiveMaxSize
- **Description**: The maximum allowed size (in bytes) for any file within an archive. A value of "0" means the maximum allowed value (INT64_MAX).

- ArchiveMaxCount

- **Default**: 0
- **Getter**: LocalScannerConfig.archiveMaxCount

- **Setter**: LocalScannerConfig.Builder.setArchiveMaxCount
- **Description**: The maximum allowed number of files within an archive. A value of "0" means the maximum allowed value (INT64_MAX).

- ScanMode

- **Default**: SMART
- **Getter**: LocalScannerConfig.scanMode
- **Setter**: LocalScannerConfig.Builder.setScanMode
- **Description**: Scanning method. Available options are: SMART — Smart Extensions scan mode. The files scanned for malware are chosen by MAVAPI. The choice is made based on the files content. This is the recommended setting. ALL — All scan mode. Files are scanned for malware, no matter their content or extension.

- DetectAppl

- **Default**: false
- **Getter**: LocalScannerConfig.detectAppl
- **Setter**: LocalScannerConfig.Builder.setDetectAppl
- **Description**: Activates detection of applications from uncertain origin or which might be hazardous to use.

- DetectSpr

- **Default**: false
- **Getter**: LocalScannerConfig.detectSpr
- **Setter**: LocalScannerConfig.Builder.setDetectSpr
- **Description**: Activates the detection of programs that violate the private domain (Security Privacy Risk). This concerns software that may be able to compromise the security of the device, initiate unwanted program activities, damage the privacy or spy on the user's behavior and could therefore be unwanted.

- DetectPfs

- **Default**: false
- **Getter**: LocalScannerConfig.detectPfs
- **Setter**: LocalScannerConfig.Builder.setDetectPfs
- **Description**: Activates the detection of fraudulent software, also known as "scareware" or "rogueware" that pretends the device is infected by viruses or malware. The term "PFS" (Possible Fake Software) describes a software that usually requires a fee but has no functionality or that installs other suspicious components.

- DetectAdware

- **Default**: false
- **Getter**: LocalScannerConfig.detectAdware
- **Setter**: LocalScannerConfig.Builder.setDetectAdware
- **Description**: Detections in this category display excessive advertisements.

- DetectAdspy

- **Default**: true
- **Getter**: LocalScannerConfig.detectAdspy
- **Setter**: LocalScannerConfig.Builder.setDetectAdspy
- **Description**: Activates detection of software that displays advertising pop-ups or sends user-specific data to third parties without the user's consent and might therefore be unwanted. ADSPY denotes adware or spyware.

- DetectPua

- **Default**: true
- **Getter**: LocalScannerConfig.detectPua
- **Setter**: LocalScannerConfig.Builder.setDetectPua
- **Description**: Activates the detection of Potentially Unwanted Applications. These are hidden applications, unknowingly downloaded alongside legitimate apps which clutter the device with ads, hijack user's browser, slow down the device – and frequently collect data on what the user clicks.

- ProductCode

- **Default**: N/A
- **Getter**: LocalScannerConfig.productCode
- **Setter**: LocalScannerConfig.Builder.setProductCode
- **Description**: Product code

Note: Only if the value is valid, will it be set, otherwise an error will be logged and the value is ignored.

5.4 Usage

Note: Make sure that `MavapiLibrary` is integrated properly into the project. Review Integrating MavapiLibrary - Description

First, it requires initialization:

```
import com.avira.mavapi.*

// ...
fun initializeMavapiLib(applicationContext: Context) {
 // ...
    MavapiLibController.initialize(applicationContext)
     .add(MavapiConfig.Builder(applicationContext))
     .add(LocalScannerConfig.Builder(applicationContext)
               .setDetectAdspy(true)
                    .setDetectAdware(true)
                    .setDetectAppl(true)
                    .setDetectPfs(true)
                    .setDetectPua(true)
                    .setDetectSpr(true)
                    .setProductCode(<LocalScannerProductCode>))
     .build()

 // ...
}
```

Check if the initialization was successful:

```
// store controller in order to avoid calling
 'MavapiLibController.getLocalScannerController()' over and over for the same result
val localScannerController = MavapiLibController.getLocalScannerController()
// check initialization
if (localScannerController.getInitializationStatus() == InitStatus.FAILED) {
 // ... localScanner cannot be used, check logs
}
```

Then create instance for scanning:

```
val scannerInstance = localScannerController.createInstance()
// check if there was an error
if (scannerInstance.getInitStatus() == InitStatus.FAILED) {
    // ... localScanner cannot be used, check logs and 'scannerInstance.getInitErrorCode()'
}
```

Start scanning:

```
// optionally, callbacks can be set
scannerInstance.setScanCallback(object : LocalScannerCallback {
 override fun onScanComplete(callbackData: LocalScannerCallbackData) { /* ... do something
 with the result */ }
    override fun onScanError(callbackData: LocalScannerCallbackData) { /* ... do something
 with the error */ }
})
// perfom the scans
for (filePath in list) {
 val result = scannerInstance.scan(filePath)
    // ... do something with the result
}
```

Once scanning everything that was planned is finished, destroy the instances:

```
// before this, make sure that all other instances have finished their tasks
localScannerController.clearInstances()
```

5.5 Notes and known limitations

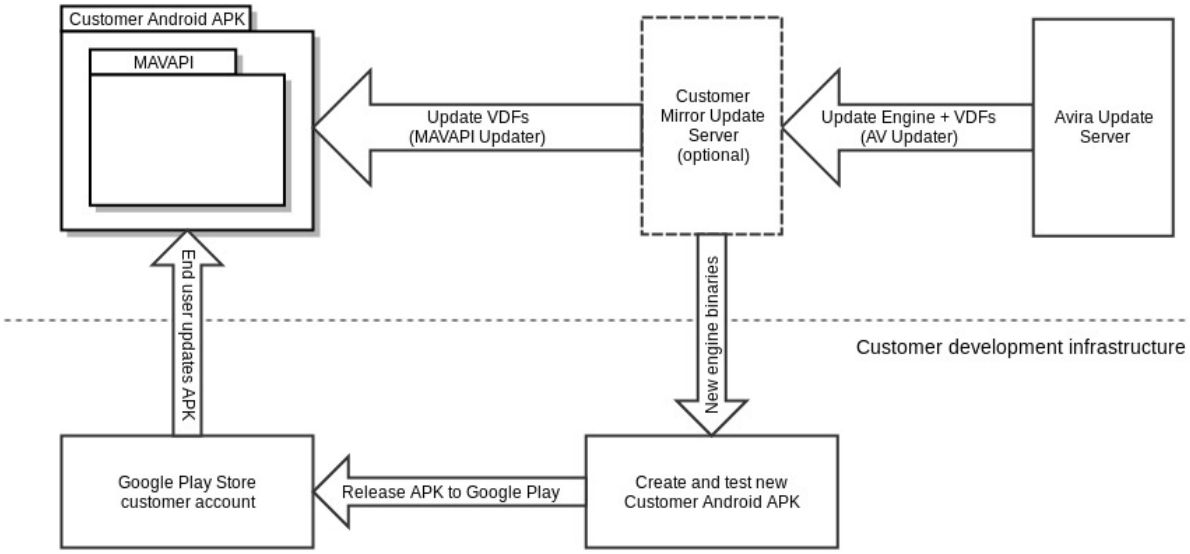
All Mavapi and engine library files are signed and this sign is validated at runtime.

Android Studio will try to truncate the library files thus removing that signature. Make sure not to strip the files by adding them to *doNotStrip* option within **build.gradle**:

```
packagingOptions {
    doNotStrip = [
            "**/libaecore.so",
            "**/libaedroid.so",
            "**/libaeexp.so",
            "**/libaehelp.so",
            "**/libaemobile.so",
            "**/libaepack.so",
            "**/libaescn.so",
            "**/libaevdf.so",
            "**/libmaven.so",
    ]
}
```

Engine files might change from version to version, thus make sure to check them or skip all ***.so** files from being stripped.

According to Google's policies, applications are not allowed to download any executable files from outside Google Play Store. That is why engine files must be updated together with the customer APK. The workflow of updating the file:



Note: To update `LocalScanner`, all scanning instances must be destroyed, otherwise, the update will fail.

6 Integrating ProtectionCloud

6.1 Description

`ProtectionCloud` is a component used for checking installed APK hashes (as well as other information about that file) with Avira Protection Cloud server. Both, the information send as well as the response are stored locally in a cache to speed up and reduce the process.

For stored APKs, `ProtectionCloud` will not store the data in cache and it will send only partial information (limitation from parsing the APK).

Information that it is send:

- OS information, e.g. language, arch, type, version, model (nothing about the user)
- `MavapiLibrary` components versions, e.g. engine version, vdf versions, etc.
- APK infromation: e.g. size, certificates, version, parent, installer, sdk, etc.

For full check, the application might require additional permissions added to **AndroidManifest.xml**:

- `android.permission.QUERY_ALL_PACKAGES` — to view all installed applications (needed for install APKs check)
- `android.permission.MANAGE_EXTERNAL_STORAGE` — to access files on external storage (if needed for stored APKs check)

To use this component, the following dependencies are required in **build.gradle**:

```
dependencies {
    implementation "com.squareup.retrofit2:retrofit:2.9.0"
    implementation "com.squareup.retrofit2:converter-gson:2.9.0"
    implementation "androidx.room:room-runtime:2.4.2"
    kapt "androidx.room:room-compiler:2.4.2"
    implementation "com.squareup.okhttp3:okhttp:4.9.3"
}
```

Additionally, `ProtectionCloud` supports the following plug-ins:

- `AVKCCert` — filters packages before checking the cache and Avira Protection Cloud server

6.2 Interface

`ProtectionCloudController` interface:

- `getInitializationStatus(): InitStatus`

  Gets the initialization status of the controller. If failed, calling the rest of the methods will return an error.

- `getProtectionCloud(): ProtectionCloud`

  Gets instance for ProtectionCloud

`ProtectionCloud` interface:

- `getInitStatus(): InitStatus`

  Gets the initialization status. In case the controller was not initialized successfully, as result this instance will not be initialized successfully either.

- `getInitErrorCode(): ProtectionCloudErrorCodes`

  Gets the error code of the initialization result.

- `syncCache(callback: ProtectionCloudSyncCallback)`

  Synchronizes the current state of the applications with the local cache.

  Notes:

  – The call must not be run in the UI thread.

  – The call will be time consuming.

  – Call this function after initializing and whenever the cache might be out of sync. For example, if the service receiving signals regarding changes on the system, was down.

- `queryInstalledAPKsSync(packageNames: List<String>): ProtectionCloudDetectionResult`

  Query Avira Protection Cloud about information on a list of packages.

  Return data as ProtectionCloudDetectionResult.

  `ProtectionCloudResult.Failure` can contain `ProtectionCloudErrorCodes.OK`, `ProtectionCloudErrorCodes.NOT_INITIALIZED`, `ProtectionCloudErrorCodes.CACHE_NOT_SYNCHRONIZED`.

  Iterates through *packageNames* and checks the ones found in local cache with Avira Protection Cloud. The results will be added to the local cache to avoid future checks of the same packages.

  Note: The flow is to have the local cache always up to date, and check either specific packages or all.

Important: To keep the local cache up to date with the changes on the system (installed, updated or removed applications) use `MavapiLibController.updateLocalCache`.

- `queryInstalledAPKsSync(packageNames: List<String>, callback: ProtectionCloudQueryCallback?)`

  Query Avira Protection Cloud about information on a list of packages.

  Return data in ProtectionCloudQueryCallback.

  Protection Cloud errors will be returned in the ProtectionCloudQueryCallback.onError.

  ProtectionCloudQueryCallback.onComplete can contain `ProtectionCloudErrorCodes.OK`, `ProtectionCloudErrorCodes.NOT_INITIALIZED`, `ProtectionCloudErrorCodes.CACHE_NOT_SYNCHRONIZED`.

  Iterates through *packageNames* and checks the ones found in local cache with Avira Protection Cloud. The results will be added to the local cache to avoid future checks of the same packages.

  Note: The flow is to have the local cache always up to date, and check either specific packages or all.

  Important: To keep the local cache up to date with the changes on the system (installed, updated or removed applications) use `MavapiLibController.updateLocalCache`.

- `queryStoredAPKs(apkPaths: List<String>): ProtectionCloudDetectionResult`

  Query Avira Protection Cloud about information on a list of apk files.

  Return data as ProtectionCloudDetectionResult

  `ProtectionCloudResult.Failure` can contain `ProtectionCloudErrorCodes.OK`, `ProtectionCloudErrorCodes.NOT_INITIALIZED`, `ProtectionCloudErrorCodes.CACHE_NOT_SYNCHRONIZED`.

  Note: Unlike the `queryInstalledAPKsSync` this will not use or update the local cache with the results.

- `queryStoredAPKs(apkPaths: List<String>, callback: ProtectionCloudQueryCallback)`

  Query Avira Protection Cloud about information on a list of apk files.

  Return data in ProtectionCloudQueryCallback

  Protection Cloud errors will be returned in the ProtectionCloudQueryCallback.onError

  ProtectionCloudQueryCallback.onComplete can contain `ProtectionCloudErrorCodes.OK`, `ProtectionCloudErrorCodes.NOT_INITIALIZED`, `ProtectionCloudErrorCodes.CACHE_NOT_SYNCHRONIZED`

  Note: Unlike the `queryInstalledAPKsSync` this will not use or update the local cache with the results.

`ProtectionCloudDetectionResult`:

- `Failure`:
  - `errorCode: ProtectionCloudErrorCodes` — error code in case the check failed
- `Detections`:
  - `detections: Map<String, ProtectionCloudDetection>` — detection results for each package

`ProtectionCloudDetection` interface:

- `pkgName: String` — package name
- `errorStatus: ProtectionCloudErrorCodes` — scan error status for package
- `detectionStatus: DetectionStatus` — check status: 0 (unknown), 1 (clean), 2 >= (infected), -1 <= (failed)
- `detectionName: String` — details about the result if it was infected
- `sourceDetection: SourceDetection`

`SourceDetection`:

- Source of detection results. The scope is to identify the module that came up with the results.

`DetectionStatus`:

- Enum with integer value to describe the status of the detection. Subject to future changes.

6.3 Configuration

The class for configuring `ProtectionCloud` is `ProtectionCloudConfig`. The configuration method is based on a builder design pattern, meaning that in order to generate a configuration object, `ProtectionCloudConfig.Builder` must be used to set the preferred configuration and to call afterward `.build()` to generate the configuration object.

`ProtectionCloudConfig` interface supports the following:

- ApiKey:

- **Default**: ""
- **Getter**: ProtectionCloudConfig.apiKey
- **Setter**: ProtectionCloudConfig.Builder.setApiKey()
- **Description**: Avira Protection Cloud API Key, needed in order to check hashes.

- Proxy:

- **Default**: null
- **Getter**: ProtectionCloudConfig.proxy
- **Setter**: ProtectionCloudConfig.Builder.setProxy()
- **Description**: Proxy for hash check.

- ApcUrl:

- **Default**: { "https://query-api.eu1.apc.avira.com", "" }
- **Getter**: ProtectionCloudConfig.apcUrl
- **Setter**: ProtectionCloudConfig.Builder.setApcUrl()
- **Description**: Avira Protection Cloud URL together with the country code from where the application is used.

- Reserved:

- **Default**: N/A
- **Getter**: ProtectionCloudConfig.reserved
- **Setter**: ProtectionCloudConfig.Builder.setReserved()

- **Description**: Additional information, optionally supplied by the integrator, maximum 64 characters (truncated otherwise).

- SyncCacheThreads:

- **Default**: 1
- **Getter**: ProtectionCloudConfig.syncCacheThreads
- **Setter**: ProtectionCloudConfig.Builder.setSyncCacheThreads()
- **Description**: Set the maximum number of threads used to synchronize local cache. Increasing the number of threads for synchronizing the local cache can improve performance. Be aware that the performance improvement is not linear with the number of threads.

  For example in our tests (Pixel 4a, Android 12, 300 application installed), incrementing the number of threads, we saw the following progression:

```
1 thread  -> 19s
2 threads -> 13s
3 threads -> 12s
4 threads -> 10.5s
5 threads -> 10.0s
```

  We strongly recommend to consider the hardware your app is targeting, and make a decision based on your own testing results.

- ConnectTimeout:

- **Default**: 3
- **Getter**: ProtectionCloudConfig.connectTimeout
- **Setter**: ProtectionCloudConfig.Builder.setConnectTimeout()
- **Description**: Number of seconds until it throws timeout because the connection was not established. A value of 0 means no timeout, otherwise values must be between 0 and 2147483.

- ReadTimeout:

- **Default**: 30
- **Getter**: ProtectionCloudConfig.readTimeout
- **Setter**: ProtectionCloudConfig.Builder.setReadTimeout()
- **Description**: Number of seconds until it throws timeout because there was no information received from server. A value of 0 means no timeout, otherwise values must be between 0 and 2147483.

6.4 Usage

Note: Make sure that `MavapiLibrary` is integrated properly into the project. Review Integrating MavapiLibrary - Description.

First, it requires initialization:

```
import com.avira.mavapi.*

// ...
fun initializeMavapiLib(applicationContext: Context) {
 // ...
    MavapiLibController.initialize(applicationContext)
     .add(MavapiConfig.Builder(applicationContext))
     .add(ProtectionCloudConfig.Builder(applicationContext)
              .setApiKey(<ProtectionCloudAPIKey>)
     .build()
 // ...
}
```

Check if the initialization was successful:

```
// store controller in order to avoid calling
 'MavapiLibController.getProtectionCloudController()' over and over for the same result
val protectionCloudController = MavapiLibController.getProtectionCloudController()
// check initialization
if (protectionCloudController.getInitializationStatus() == InitStatus.FAILED) {
 // ... ProtectionCloud cannot be used, check logs
}
```

Then get the instance:

```
val protectionCloud = protectionCloudController.getProtectionCloud()
// check if there was an error
if (protectionCloud.getInitStatus() == InitStatus.FAILED) {
    // ... protectionCloud cannot be used, check logs and
 'protectionCloud.getInitErrorCode()'
}
```

Start scanning:

```
// pass empty list in case we want to check all packages, or pass a list with the specific
 packages we want to check
val results = protectionCloud.queryInstalledAPKsSync(emptyList())
// check if there was an error
if (results.errorCode != ProtectionCloudErrorCodes.OK) {
    // ... do something with the error
} else {
    // ... do something with the results
}
```

6.5 Notes and known limitations

`ProtectionCloud` uses local cache as a map of the system status. Meaning, that if an application is installed and it's not added to the cache, on query, the package will not be checked with Avira Protection Cloud server.

To keep the cache up to date, intercept broadcast messages related to install/uninstall applications (or other similar method) and use `MavapiLibController.updateLocalCache` to update the local cache.

Note: Adding new applications or updating the information within the cache will be time consuming. In order to speed up the performance when checking them or making a query, all the necessary data is stored in cache.

`ProtectionCloud` **CANNOT** be run multi-threaded and it is not required to be. The check function can receive a list of packages to be checked. This list, together with additional information is sent in a batch to the cloud for checking. When the batch is received by the cloud, all the hashes are checked in parallel. The batch size is set to the maximum value supported to get the best performance from the cloud check. Thus, adding multi-threading support to it will not guarantee that the processing will be faster.

7 Integrating AVKCCert

7.1 Description

`AVKCCert` (Avira Known Clean Certificates) is a component used for filtering checks based on APK zip check and certificates checks. This component uses a clean whitelist certificate list generated by Avira to verify if the APK is trusted or not.

Technical information:

- Required files: avkccert.db
- Available update rate: approx. 1 update/day

`AVKCCert` is a plugin that can be used by:

- `ProtectionCloud` — filters packages before checking the cache and Avira Protection Cloud server

7.2 Interface

`AVKCCertController` interface:

- `getInitializationStatus(): InitStatus`

  Gets the initialization status of the controller. If failed, calling the rest of the methods will return an error.

- `getAVKCCert(): ProtectionCloud`

  Gets the instance for `AVKCCert`

- `getUpdateModule(): Module`

  Returns update module used by `UpdaterController` to update the virus signatures.

- `avkccertVersion: String`

Variable containing the AVKCCert version. It is set after it is initialized successfully and the required file is present on the device.

`AVKCCert` interface:

- `tryLoadWhiteList(path: String): Boolean`

  Try load the whitelist form specific path. If content is valid, it will store in memory, otherwise the current list will remain unchanged.

7.3 Configuration

The class for configuring `AVKCCert` is `AVKCCertConfig`. The configuration method is based on a builder design pattern, meaning that in order to generate a configuration object, `AVKCCertConfig.Builder` must be used to set the preferred configuration and to call afterward `.build()` to generate the configuration object.

`AVKCCertConfig` interface supports the following:

- WhitelistPath

- **Default**: %app_dir%/bin/antivirus/
- **Getter**: AVKCCertConfig.whitelistPath
- **Setter**: AVKCCertConfig.Builder.setWhitelistPath
- **Description**: Path to where the whitelist is stored

7.4 Usage

Notes: Make sure that `MavapiLibrary` is integrated properly into the project. Review Integrating MavapiLibrary - Description.

First, it requires initialization:

```
import com.avira.mavapi.*

// ...
fun initializeMavapiLib(applicationContext: Context) {
 // ...
    MavapiLibController.initialize(applicationContext)
     .add(MavapiConfig.Builder(applicationContext))
     .add(AVKCCertConfig.Builder(applicationContext))
     .add(ProtectionCloudConfig.Builder(applicationContext)
              .setApiKey(<ProtectionCloudAPIKey>))
     .attachPlugin(ProtectionCloudController::class.java.simpleName,
 AVKCCertConfig::class.java.simpleName)
     .build()

 // ...
}
```

Check if the initialization was successful:

```
// store controller in order to avoid calling 'MavapiLibController.getAVKCCertController()'
 over and over for the same result
val avkccertController = MavapiLibController.getAVKCCertController()
// check AVKCCert initialization
if (avkccertController.getInitializationStatus() == InitStatus.FAILED) {
 // ... AVKCCert cannot be used, check logs
    // this fail will NOT affect ProtectionCloud in any way, the plugin will just not be
 attached
}

// store controller in order to avoid calling
 'MavapiLibController.getProtectionCloudController()' over and over for the same result
val protectionCloudController = MavapiLibController.getProtectionCloudController()
// check ProtectionCloud initialization
if (protectionCloudController.getInitializationStatus() == InitStatus.FAILED) {
 // ... ProtectionCloud cannot be used, check logs
}
```

At this point, if there are no errors, `ProtectionCloud` should be able to use `AVKCCert` to filter the hash checks.

7.5 Notes and known limitations

Mavapi AAR library will not come with a pre-set clean whitelist certificate list. Either use `Updater` to download a fresh clean whitelist certificate list or add manually a set of clean whitelisted certificates list (not recommended).

8 Integrating Updater

8.1 Description

The `Updater` component is used to update other components. Each updatable component must have a `Module` .

When the controller of an updatable component is initialized successfully, it will automatically register that module (internally), and will be updated when `UpdaterController.updateAllComponents()` is called. Alternatively, `UpdaterController.updateComponent(module)` can be called to update a specific module.

To use this component, the following dependencies are required in **build.gradle**:

```
dependencies {
    implementation "com.squareup.okhttp3:okhttp:4.9.3"
}
```

Updatable components:

- `LocalScanner`
- `AVKCCert`

8.2 Interface

`UpdaterController` interface:

- `getInitializationStatus(): InitStatus`

  Gets the initialization status of the controller. If failed, calling the rest of the methods will return error.

- `updateAllComponents(): Map<String, UpdaterResult>`

  Updates all successful initialized components that have `Module`

- `updateComponent(module: Module): UpdaterResult`

  Updates specific module.

- `attemptToAbort(): AbortResult`

  Attempts to stop the download. If the download hasn't started, it will remember the decision and stop the moment it tries to download the files. If the download finished, the call will return an error.

  The function can return:

  – `AbortResult.ABORT_SUCCESSFULLY` — The Updater received the abort signal and will stop almost immediately returning the error `UpdaterResult.ERROR_USER_ABORT`.

  – `AbortResult.ABORT_IN_PROGRESS` — The Updater just started and is testing the scanner. This part of the process cannot be stopped and it might take a few (milli)seconds for the Updater to stop. It does not differ much from `AbortResult.ABORT_SUCCESSFULLY`, only that it will take a bit longer. The updater will return `UpdaterResult.ERROR_USER_ABORT`.

  – `AbortResult.ABORT_FAILED` — The Updater is in the latest stage of updating where it makes no sense to stop the process anymore (because the downloaded files and the scanner are being tested, and because of cleaning up). In this case, the updater will **NOT** return `UpdaterResult.ERROR_USER_ABORT`.

  How can it be used?

  – The return value can be ignored and just wait for the result of the Updater. The Updater will stop as soon as possible.

  – Use the result (in case of `AbortResult.ABORT_SUCCESSFULLY` and `AbortResult.ABORT_IN_PROGRESS`) to stop waiting for the Updater (assuming that it is running in a separated thread) and note that there was no successful update and that later a new try is needed. If received `AbortResult.ABORT_FAILED` then the user should wait for the result.

- `getRemoteVersionForModule(module: Module): UpdaterResultOf<String>`

  Get remote version from update servers for required module.

8.3 Configuration

The class for configuring `UpdaterCloud` is `UpdaterConfig`. The configuration method is based on a builder design pattern, meaning that in order to generate a configuration object,

`UpdaterConfig.Builder` must be used to set the preferred configuration and to call afterward `.build()` to generate the configuration object.

`UpdaterConfig` interface supports the following:

- DownloadPath

- **Default**: %app_dir%/bin/antivirus/
- **Getter**: UpdaterConfig.downloadPath
- **Setter**: UpdaterConfig.Builder.setDownloadPath()
- **Description**: Path where to download files for update (such as .info files)

- UpdateServers

- **Default**: "https://oem.avira-update.com/update/"
- **Getter**: UpdaterConfig.updateServers
- **Setter**: UpdaterConfig.Builder.setUpdateServers()
- **Description**: Update server adress.

Note: For efficiency reasons, control, and better user experience, it is recommended to setup and use a mirror update server for managing the frequent VDF (Virus Definitions Files) updates. For information on how to setup mirror update servers, please contact our international sales engineers or our OEM support.

- RandomizeUpdateServerList

- **Default**: True
- **Getter**: UpdaterConfig.randomizeUpdateServerList
- **Setter**: UpdaterConfig.Builder.setRandomizeUpdateServerList()
- **Description**: Enables the randomization of update server list (trying to balance the load on the update servers).

- ConnectTimeout

- **Default**: 60
- **Getter**: UpdaterConfig.connectTimeout
- **Setter**: UpdaterConfig.Builder.setConnectTimeout()
- **Description**: Connection to update server timeout. A value of 0 means no timeout, otherwise values must be between 0 and 2147483.

- ReadTimeout

- **Default**: 300
- **Getter**: UpdaterConfig.readTimeout
- **Setter**: UpdaterConfig.Builder.setReadTimeout()
- **Description**: Reads the timeout for update server calls. A value of 0 means no timeout, otherwise values must be between 0 and 2147483.

- Proxy

- **Default**: N/A

- **Getter**: UpdaterConfig.proxyHost, UpdaterConfig.proxyPort
- **Setter**: UpdaterConfig.Builder.setProxy()
- **Description**: Proxy for the updater connection.

8.4 Usage

Note: Make sure that `MavapiLibrary` is integrated properly into the project. Review Integrating MavapiLibrary - Description.

First, it requires initialization:

```
import com.avira.mavapi.*

// ...
fun initializeMavapiLib(applicationContext: Context) {
 // ...
    MavapiLibController.initialize(applicationContext)
     .add(MavapiConfig.Builder(applicationContext))
     // ... also add some updatable components to be initialized
     .add(UpdaterConfig.Builder(applicationContext))
     .build()
 // ...
}
```

Check if the initialization was successful:

```
// store controller in order to avoid calling 'MavapiLibController.getUpdaterController()'
 over and over for the same result
val updaterController = MavapiLibController.getUpdaterController()
// check initialization
if (updaterController.getInitializationStatus() == InitStatus.FAILED) {
 // ... Updater cannot be used, check logs
}
```

Run update process:

```
val result = updaterController.updateAllComponents()
// ... do something with the results
```

8.5 Notes and known limitations

If no updatable component is initialized, then `Updater.updateAllComponents` will return empty results.

`Updater` **CANNOT** and **SHOULD NOT** be run multi-threaded (e.g. to update in parallel multiple modules). It is already optimized for best downloading time and speed. Running multi-threaded, might affect performance and result in unknown behavior.

9 Android compatibility

9.1 Android 5 to 9

In AndroidManifest.xml:

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

Ensure that the user has granted permission to the application:

```
@Override
public void onCreate(Bundle savedInstanceState)
{
 // ...
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE) !=
 PackageManager.PERMISSION_GRANTED)
    {
        ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.READ_EXTERNAL_STORAGE}, <requesting code>);
    }
 // ...
}
//...

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[]
 grantResults)
{
 if (requestCode == <requesting code>)
    {
  if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE) ==
 PackageManager.PERMISSION_GRANTED)
  {
   // Permission granted
  } else {
   // Permission denied
  }
 }
}
```

9.2 Android 10

Starting with Android 10, some changes where made to how the external storage is managed. Thus, it will require the additional flag `requestLegacyExternalStorage` set for the application.

In AndroidManifest.xml:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<application
 ...
    android:requestLegacyExternalStorage="true"
    ...>
```

Ensure that the user has granted permission to the application:

```
@Override
public void onCreate(Bundle savedInstanceState)
{
 // ...
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE) !=
 PackageManager.PERMISSION_GRANTED)
    {
        ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.READ_EXTERNAL_STORAGE}, <requesting code>);
    }
 // ...
}
//...

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[]
 grantResults)
{
 if (requestCode == <requesting code>)
    {
  if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE) ==
 PackageManager.PERMISSION_GRANTED)
  {
   // Permission granted
  } else {
   // Permission denied
  }
 }
}
```

9.3 Android 11

With Android 11:

- The `requestLegacyExternalStorage` becomes deprecated
- A new type of permission category is added: **All files access**. Which will need **MANAGE_EXTERNAL_STORAGE** permission. This is needed to access all files on storage, except for private application files.
- Choose **QUERY_ALL_PACKAGES** to view all installed applications

In AndroidManifest.xml:

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

Ensure that the user has granted permission to the application:

```
@Override
public void onCreate(Bundle savedInstanceState)
{
 // ...
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE) !=
 PackageManager.PERMISSION_GRANTED)
    {
        ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.READ_EXTERNAL_STORAGE}, <requesting code>);
    } else {
     checkExternalStorageManager();
    }
 // ...
}
//...

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[]
 grantResults)
{
 if (requestCode == <requesting code>)
    {
  if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE) ==
 PackageManager.PERMISSION_GRANTED)
  {
   // Permission granted
   checkExternalStorageManager();
  } else {
   // Permission denied
  }
 }
}

private void checkExternalStorageManager() {
 if (Build.VERSION.SDK_INT < 30 || Environment.isExternalStorageManager()) {
  return;
    }
    // ...
    Intent intent = new Intent(ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION);
    startActivity(intent);
    // ...
}
```

9.4 Android 12

Starting with Mavapi 3.1, the library is compatible with Android 12.

10 Licensing

Your company will receive one dedicated *.key* file, which can be delivered to all your customers together with your own application's key. Please note, that you always have to deliver the *.key* file, otherwise MAVAPI will not scan.

This key is unique for you as a company. It relates (among other things) to the expiration date and the product ID that you will receive from Avira (in the same time with the new key).

Warning: You are responsible for integrating, distributing or exchanging the key in your application, which integrates MAVAPI. Your application must send the product ID to MAVAPI. MAVAPI processes the key to initialize the engine. If the product ID and the key do not match, the product does not scan.

The key you receive from Avira represents a license that is valid for an agreed amount of time. A license can be renewed or blacklisted whenever necessary. Practically, it is up to you if your customers will even see that they have two licenses. For them, the MAVAPI license key can be just another file included in the package.

Of course, your product must make sure that the key does not expire, otherwise it will not scan anymore.

When the key expires, the product will scan using its existing signatures. If you update the engine or signatures and the expiration date in the key is older than the date in the engine or the signature database, MAVAPI will not scan.

11 Contact information

11.1 Support services

During evaluation, integration, and live use

If you are evaluating or starting to integrate Avira's technology into your solution, or if your integration is finalized and you are going to release your solution to your customers, the Integration Support engineers will answer your technical questions — from planning the architecture of the integration, to detailed code-related routines and live use.

To contact the OEM support team for technical issues, mailto:oemsupport@avira.com

Partner Portal

For our OEM customers we also provide a login to our Partner Portal which includes all the latest news and information about Avira's technology, SDK downloads, and documentation: OEM Partner Portal

11.2 Contact

Address

Avira Operations GmbH

Kaplaneiweg 1

D-88069 Tettnang

Germany

Internet

You can find further information about us and our products on the website: https://oem.avira.com

Europe
Middle East, Africa

Avira
Kaplaneiweg 1
88069 Tettnang, Germany
Tel: +49 7542 5000

Americas

Avira, inc
c/o WeWork, 75 E Santa Clara Street
Suite 600, 6th floor San José
CA 95113 United States

Asia/Pacific and China

Avira Pte Ltd
50 Raffles Place
32-01 Singapore Land Tower
Singapore 048623

Japan

Avira GK
8F Shin-Kokusai Bldg
3-4-1, Marunouchi Chiyoda-ku
Tokyo 100-0005, Japan