App bundle doesn't install any native libraries?

+1 1 Hotlists (4) Unmark Duplicate

Comments (15) Dependencies Duplicates (0) Blocking (0) Resources (5)

Duplicate of 147768067 Bug P2 + Add Hotlist Reviewed (L1)

STATUS UPDATE No update yet. Edit

Reporter	
Type	Bu
Priority	P2
Severity	S2
Status	D
Access	De
Assignee	
Verifier	--
Collaborators	
CC	ad cc cf. iu. ni. ...
AOSP ID	--
Blocking Release	--
Release Status	--
Found In	--
Targeted To	--
Verified In	--
In Prod	

Show 1 addition

DESCRIPTION cc...@gmail.com created issue #1 Jan 20, 2020 08:17PM

Gradle version: 5.4.1
Android Plugin Version: 3.5.3
Module Compile Sdk Version: 29
Module Build Tools Version: 29.0.2
Android SDK Tools version: 26.1.1

I'm trying to migrate to app bundles instead of APK, unfortunately the built bundle, which contains all necessary native libraries, doesn't seem to let Android Studio or Play Store install them !

I've tried the answer from this: https://stackoverflow.com/questions/57576202/android-release-app-bundle-build-missing-native-lib-so-files
Even though I don't think it makes a lot of sense and actually answers the issue, it most certainly didn't help in any ways.

I tried launching it from Android Studio on various devices (Pixel 3 XL Android 10, Nexus 9 Android 7), the library is never installed.

I then published an internal test on Play Store, and while Play Console recognizes the various native libraries, it didn't install anything.

I then tried another suggestions to add those lines in build.gradle, and debugging from Android Studio didn't work any better, however the built/published bundle on Play Store seemed to work. But only once. I published an update and yet again, no native libraries are installed!?

```
bundle {
    language {
        enableSplit = false
    }
    density {
        enableSplit = true
    }
    abi {
        enableSplit = true
    }
}
```

Just a quick note about how I built the 32 vs 64 bits libraries: 32 bits uses SDK 19 otherwise run_pie workaround will not work, and for 64 bits I have to use compile SDK 23. Quite a while ago I was using a single module just fine until 64 bits support was removed from SDK 19, or something similar.

Is there a way to make a single module, compiling with SDK 19 the 32 bits version and SQK 23 the 64 bits?

Can someone please help me figure out what's wrong and how to fix this? I'd like to use bundles to avoid having to constantly confirm the warning and publish the obfuscation file separately. It has become a pain to publish any app on the console these days.

See attached screenshot of the .aab content and Play Console analysis.

sm_as.png
51 KB View Download

sm_pc.png
57 KB View Download

✓ Links (5) Hide all

Links (5)

"I've tried the answer from this: https://stackoverflow.com/questions/57576202/android-release-app-bundle-build-missing-native-lib-so-files"	cc...@ #1
"It seems I'm not the only one: https://stackoverflow.com/questions/58753615/native-libraries-not-installed-in-release-build"	cc...@ #2
"There is some documentation on how to load native libraries here: https://developer.android.com/training/articles/perf-jni#native-libraries"	le...@ #7
" ...me explanation of what we do with uncompressed native libraries here: https://developer.android.com/topic/performance/reduce-apk-size#extract-false"	le...@ #7
"It's in the NDK header file android/dlext.h, and we have Doxygen auto-generated docs for that: https://developer.android.com/ndk/reference/group/libdl ."	rp...@ #13

COMMENTS All comments Oldest first

cc...@gmail.com <cc...@gmail.com> #2 Jan 21, 2020 12:19AM

It seems I'm not the only one: https://stackoverflow.com/questions/58753615/native-libraries-not-installed-in-release-build

cc...@gmail.com <cc...@gmail.com> #3 Jan 21, 2020 09:16PM

Interestingly, checked on a Nexus 9 (Android 7.1.1) and a Pixel 3 (Android 10), the APK containing the lib is there, however it's not installed !

I tried to set extractNativeLibs="false" (in application tag of manifest), however it didn't work.

Once I set "android.bundle.enableUncompressedNativeLibs=false" in gradle.properties, it started to work as designed.

However the first flag should have done that already.

cc...@gmail.com <cc...@gmail.com> #4 Jan 23, 2020 04:28AM

I meant I tried setting extractNativeLibs="true" in manifest, didn't work.

je...@google.com <je...@google.com> Jan 23, 2020 05:22AM

Assigned to le...@google.com.

le...@google.com <le...@google.com> #5 Jan 23, 2020 05:37AM

Status: Duplicate of 147768067

This is an intended behaviour, please see the explanation in the bug I marked this one a duplicate of.
If you think this is a mistake, feel free to reopen or ask for clarifications.

cc...@gmail.com <cc...@gmail.com> #6 Jan 23, 2020 06:22AM

As long as I can make it work, this is fine with me. However I'd like to point out documentation is very scarce on the subject and unclear. I search for information for quite some time until I could find some stack overflow mentioning those options.

Furthermore I'd like to point out that it would have taken me a lot more time to figure this out if my device was not rooted, because it wouldn't have allowed me to check app's folder content and realize the APK was there.

le...@google.com <le...@google.com> #7 Jan 23, 2020 06:42AM

Thank you for the feedback.

There is some documentation on how to load native libraries here: https://developer.android.com/training/articles/perf-jni#native-libraries
And some explanation of what we do with uncompressed native libraries here: https://developer.android.com/topic/performance/reduce-apk-size#extract-false

But I suspect the issue is that when you or any SDK you depend on haven't followed these recommendations in the first place, I can understand it's hard to find what has gone wrong. Troubleshooting has more often been done through this issue tracker or third-party websites such as stackoverflow (I believe several posts have covered this topic already and been answered, it's unfortunate you couldn't find them).

Note that the `android.bundle.enableUncompressedNativeLibs=false` property is not recommended since it makes your app slower to install and bigger on users' devices.

cc...@gmail.com <cc...@gmail.com> #8 Jan 23, 2020 06:10PM

Thanks for the links. In extract false link, it says to add extract=false, however it is the default (right?). Then setting it to true has no effect, at least that's what I noticed.

FWIW, I actually need access to the .so file, which I made into an executable so that I can start it once and connect it to my app using standard input/output and later communicate with it without having to start it every time. Did this because I noticed a 100ms overhead starting any shell command.

I suppose I could create a small binary (or native lib) to load the .so file directly from the APK, however I'll have to extract it manually as it's not possible to specify which native library is to be extracted or not, right? Or maybe I could manually extract that tiny native lib and then it would load the "big" one.

Is there an "easy way" in C (I'm not using C++ for size considerations) of loading a native lib from APK ? I'll have to do some research as I suppose the code must exists to do just that when the native lib is loaded with System.loadLibrary().

le...@google.com <le...@google.com> #9 Jan 24, 2020 12:58AM

Re "extract=false", it only applies to APKs, not App Bundles. For App Bundles, Play is responsible for generating the APKs, and the APKs it generates may have this attribute set to true or false depending on the SDK versions of the devices the generated APKs will be served to.

You don't need to create a library/binary to load the .so file from the APK. As I mentioned in the duped bug, using standard Android APIs such as System.loadLibrary("libfoo") to load the file "foo.so" is sufficient and this will work regardless if the .so file is extracted or not.

To do this in C, I believe this can be done using "dlopen" and it should also work transparently.

Hope that helps,

cc...@gmail.com <cc...@gmail.com> #10 Jan 24, 2020 01:15AM

Ah! That's why extract=false never worked. No wonder. That was not quite clear to me when I read documentation.

Anyway, I'm already using System.loadLibrary().

I'll try to use dlopen(), but doesn't it take a full path ? What would be the expected path, something like /data/app/<pkg>/the_lib.apk or /data/app/<pkg>/the_lib.apk/libthe_lib.so ?

I already use it to load some system library and without full path it always failed. Maybe things have changed since I tested this.

... and much thanks for your inputs.

le...@google.com <le...@google.com> #11 Jan 24, 2020 01:20AM

@Christopher, could you please comment on the best way to load a .so file in C regardless of whether it's been extracted at installation or mmaped inside the APK? Is there any documentation on this?

cf...@google.com <cf...@google.com> #12 Jan 30, 2020 05:16AM

To answer comment#11, I'm not sure I'm best to ask about this. However, there is no one method to load a .so from native code (C/C++ etc), for an extracted shared library it would be dlopen. For a shared library that is not extracted, you have to use android_dlopen_ext.

I'm not sure about documentation. Ryan, do we have any documentation for the android_dlopen_ext call? I'm assuming it can be used from app code.

rp...@google.com <rp...@google.com> #13 Jan 30, 2020 10:44AM

> For a shared library that is not extracted, you have to use android_dlopen_ext.

I don't think this is the case? dlopen can search a ZIP file for a library. It accepts a special syntax: `<zip file>!/<zip entry>`.

Typically, an app can call one of:

- `System.loadLibrary("foo")`, or
- `dlopen("libfoo.so", ...)`

e.g.: `System.loadLibrary("native-lib")` might turn into a call to `android_dlopen_ext("/data/app/com.example.minimal-iaVP2RGv-Rm1dTfwndtcFA==/base.apk!/lib/arm64-v8a/libnative-lib.so", ...)`. (In later releases, the ART runtime uses android_dlopen_ext instead of dlopen because it needs to configure linker namespaces. That's not something an app should be doing, though.)

dlopen doesn't require an absolute path. If the path argument has no slashes in it (e.g. `"libfoo.so"`), then:

- dlopen first searches already-loaded libraries for one with a matching DT_SONAME (-Wl,--soname) string.
- Then dlopen searches for the library on the library search path. This search path might differ depending on which library called dlopen. It can include directories within APKs.

For an app, the search path is configured by the runtime, though, so if you're spawning a new process, then you'll need some other way to find your libraries (e.g. `LD_LIBRARY_PATH`, absolute paths, `DT_RUNPATH`).

IIUC, apps can consist of multiple APKs. Maybe that complicates things.

> Ryan, do we have any documentation for the android_dlopen_ext call? I'm assuming it can be used from app code.

It's in the NDK header file `android/dlext.h`, and we have Doxygen auto-generated docs for that: https://developer.android.com/ndk/reference/group/libdl.

cc...@gmail.com <cc...@gmail.com> #14 Mar 31, 2020 08:56AM

Thanks for the details, I'm testing android_dlopen_ext using the full path.

I have a couple of questions regarding implementation.

To make it simple I need to run from shell one shared library, which in turn will load the original library using android_dlopen_ext() call as detailed above.

=> Is there a way to specify a library to be extracted automatically, while others will remain in the APK?

Currently I use android.bundle.enableUncompressedNativeLibs=false (AND android:extractNativeLibs="true", left over from testing) and it works, but it's not possible to specify which library to extract and which to keep in APK file.

=> Will this flag continue to be supported in the future?

Thanks for your time.

le...@google.com <le...@google.com> #15 Mar 31, 2020 09:27PM

No, it's not possible to specify which native libraries are extracted and which aren't.

We will continue to support this gradle property as long as the alternative is not viable for developers.

Maybe Christopher or Ryan can answer whether there is any viable way to run an .so file from the shell if it's not extracted from the APK.