packageDebugBundle fails for lib/armeabi-v7a/ffmpeg file from aar dependency

Assigned Bug P3 + Reviewed (L1)

STATUS UPDATE No update yet. Edit

DESCRIPTION sa...@gmail.com created issue #1

```
Build: AI-193.5233.102.40.6137316, 202001151802,

AI-193.5233.102.40.6137316, JRE 1.8.0_212-release-1586-b4-5784211x64 JetBrains s.r.o, OS Linux(amd64) v5.3.0-42-generic, screens 1920x1080, 1680x1050

AS: 4.0 Canary 9; Kotlin plugin: 1.3.70-release-Studio4.0-1; Android Gradle Plugin: 4.0.0-beta03; Gradle: 6.1.1; NDK: from local.properties: 21.1.6273396-beta2, latest from SDK: (not found); LLDB: from SDK: 3.10.2, from PATH: 3.10.2
```

Android Gradle Plugin: __4.0.0-beta03__

Gradle: __6.1.1__

Project URL: [GitHub:FFmpeg-Android#masterQ](https://github.com/alexcohn/FFmpeg-Android/tree/masterQ)

What went wrong:

```
Execution failed for task ':sample:packageDebugBundle'.
> A failure occurred while executing com.android.build.gradle.internal.tasks.Workers$ActionFacade
   > File 'root/lib/armeabi-v7a/ffmpeg' uses reserved file or directory name 'lib'.
```

Analysis:
This is not a regression: same behavior happens on earlier, non-beta versions, too.

Some while ago, the restriction on `lib***.so` pattern for native binaries has been lifted, for good. E.g., we can package shell scripts, as shown on [developer.android.com](https://developer.andro
Same project builds APK flawlessly, which runs as expected even on R emulator.

Workaround:
The old workaround, i.e. renaming `ffmpeg` --> `lib..ffmpeg..so` still works and the AAB is generated as expected.

✓ Mentioned issues (4) ✓ Links (17)

Mentioned issues (4)

P3 [Android Q Beta] Apps can no longer execute binaries in their home directory (execute_no_trans) "https://issuetracker.google.com/128554619"

P2 Issue creating app bundle when jar dependency contains native libraries "Thanks, Pierre! Yes, that clarifies it :). I filed Issue 153606709 to track my investigation."

P2 Error when building app bundle "File 'root/lib/x86_64-MacOSX-gpp/jni/libjunixsocket-native-2.0.4.jnilib' uses reserved file or directory name 'lib'." "https://issuetracker.google.com/129457542"

P2 Loosen restrictions on files accepted by extractNativeLibs=true "https://issuetracker.google.com/160129591"

Links (17)

"Project URL: [GitHub:FFmpeg-Android#masterQ](https://github.com/alexcohn/FFmpeg-Android/tree/masterQ)"
"Some while ago, the restriction on `lib***.so` pattern for native binaries has been lifted, for good. E.g., we can package shell scripts, as shown on [developer.android.com](https://developer.android.c
"Some while ago, the restriction on `lib***.so` pattern for native binaries has been lifted, for good. E.g., we can package shell scripts, as shown on [developer.android.com](https://developer.android.c
"This error is coming from bundletool: https://github.com/google/bundletool/blob/a325c37e7d415944f20ea7ed65c92f6282abf6e4/src/main/java/com/android/tools/build/bundletool/validation/Bu
" ...fraid this battle is already lost. Unfortunately, even in current master , the 'wrong' names are filtered out. I was wrong that the restriction had been lifted: actually, in Nov'2016 a change was mad

See all related links

COMMENTS

cm...@google.com <cm...@google.com>

Assigned to an...@google.com.

je...@google.com <je...@google.com>

Reassigned to sp...@google.com.

sp...@google.com <sp...@google.com> #2

Reassigned to le...@google.com.

This error is coming from bundletool: https://github.com/google/bundletool/blob/a325c37e7d415944f20ea7ed65c92f6282abf6e4/src/main/java/com/android/tools/build/bundletool/valida

Based on Comment#1's link, it looks like bundletool should support this case.

Message last modified on Apr 7, 2020 06:29AM

---

le...@google.com <le...@google.com> #3

Reassigned to sp...@google.com.

The native library is not packaged correctly by AGP. Native libraries should be placed in the directory `lib/` not `root/lib/`.

Bundletool is stricter at validating the correct files are set in known directories, i.e. we ensure that the "lib/" directory only contains .so files. Having this validation is important because we fir which eventually get sent to end-user devices, so catching these things early is important.

I will add support for allowing files named `wrap.sh`, but the file `ffmpeg` will still remain disallowed.

@Scott, can you work on getting the native libraries packaged correctly when the library is a JAR and not an AAR?

---

sa...@gmail.com <sa...@gmail.com> #4

There is a very good reason for developers to put some things in `lib/`, especially for API 29: this is the only place that will be extracted on the target device with executable permissions, see *their home directory*. Do you prefer the ugly workaround that renames the executable?

The AGP is not to blame for `root/lib/`: the file is bundled in **resources** directory, not in **jniLibs**, as demonstrated in the `wrap.sh` example. Files from **jniLibs** are filtered for known pattern "l

NB: Actually, this attempt was irrelevant, because on older devices (e.g. API 24) the installer filters out all unexpected files, including `wrap.sh`.

Message last modified on Jun 26, 2020 04:25AM

---

sp...@google.com <sp...@google.com> #5

Reassigned to le...@google.com.

> @Scott, can you work on getting the native libraries packaged correctly when the library is a JAR and not an AAR?

Pierre, I don't follow what you're asking of me. Can you please elaborate?

---

le...@google.com <le...@google.com> #6

Reassigned to sp...@google.com.

Sorry for the brevity. Let me try to describe what I know of how Android App Bundles (AABs) are built in Gradle.

The `PackageBundleTask.kt` task calls bundletool to build the AAB (through the `BuildBundleCommand`). One of the argument is a list of modules which are effectively ZIP files with the files

In each of these "module zips" (and eventually in the AAB as well), the files are organized as such:

- Dex files are in the directory `dex/`
- Resources are in the directory `res/`
- Assets are in the directory `assets/`
- Native libraries are in the directory `lib/`
- Other files that would normally end up in the root of the APK (e.g. java resources) are in the directory `root/`.

Bundletool has some validation to ensure that all files under `lib/` are native libraries, and we currently verify this by simply checking the extension of the file to be `.so` (FYI, I've just submitt

All the files in the `root/` directory (including subdirectories) end up at the root of the APK, e.g. a file named `root/foo/LICENSE` in the module zip would end up as a file named `foo/LICENSI` named `root/lib/` or `root/dex/` etc.

The problem raised in this bug is because one of the module zip has as file under `root/lib/` which is forbidden. These module zips are built by AGP (I think in `PerModuleBundleTask.kt`).

What I've noticed is that when an app has a dependency on an AAR, then native libraries from that AAR under `lib/` are correctly packaged under `lib/` in the module zips. However, when an are packaged under `root/lib/`. If these are effectively native libraries, then AGP should consider packaging them under `lib/` instead.

I hope that clarifies it :)

---

sp...@google.com <sp...@google.com> #7

Reassigned to le...@google.com.

Thanks, Pierre! Yes, that clarifies it :). I filed Issue 153606709 to track my investigation.

Re: comment #3, I agree that it makes sense to constrain what can go in the "lib/" directory to prevent junk getting sent to end users, but should we allow devs to specify other files to include

---

sa...@gmail.com <sa...@gmail.com> #8

Please note that per instructions, `wrap.sh` should be packaged from `resources/lib/x86`, not alongside the shared libs in `jniLibs/x86` directory.

---

le...@google.com <le...@google.com> #9

Yes, Scott, I think you're right that we could do something a bit more clever than just looking at the extension. I think we could have for example a basic ELF parser to ensure that these are n

Yes, Scott. I think you're right that we could do something a bit more clever than just looking at the extension. I think we could have for example a basic ELF parser to ensure that those are na
The only issue I can think of with this approach is some developers have started using "fake" native libraries (i.e. empty files with extension `.so`) since another requirement is that if a DFM h
present (otherwise the Android platform may incorrectly start the app in 32-bit or 64-bit mode and not be able to load the native libraries once the DFM is installed and loads native libraries la

I'll file some internal bugs for each.

---

sa...@gmail.com <sa...@gmail.com> #10

"to ensure that those are native libraries" - do you intend to disable executables completely? Then, musch easier way would be to simply remove `Runtime.exec() (and seal the kernel API).

---

le...@google.com <le...@google.com> #11

No, I don't intend to disallow executables. Wouldn't those be ELF files as well?

Message last modified on Apr 9, 2020 11:11PM

---

le...@google.com <le...@google.com> #12

Support for wrap.sh has been added to bundletool.

---

da...@google.com <da...@google.com> #13

> No, I don't intend to disallow executables. Wouldn't those be ELF files as well?

Can we lift the naming restriction then? That's just an annoyance if we're not actually going to block executables in general.

---

jc...@gmail.com <jc...@gmail.com> #14

Agree, it's a completely arbitrary annoyance that serves no true purpose. My use case is a set of third-party programs that call each other by name. My app will just feed an image file to it anc
working in Termux (target API 28), but need to integrate it into my own app for work that targets 29. Since these programs call each other by name, it's more complicated than just "rename th
(and while I can do that since they are open source, it's a lot of work since the code that calls its siblings is very complex and layered, and multiple things have to be changed in multiple place

But the thing is, I shouldn't *have* to jump through all these hoops. There's no real reason for it. It should Just Work®.

If you must keep a filter around to eliminate junk, at least give us a way to bypass it by whitelisting filenames.

---

da...@google.com <da...@google.com> #15

> Having this validation is important because we find that a lot of developers put random things in there (e.g. jar, testing artifacts, etc.) which eventually get sent to end-user devices, so catcl

If this is still the concern, I think keeping the behavior behind a feature flag would be okay. I'd sort of like to see whatever policy enforced entirely in one place though rather than in both AGP
through bundletool (does everything bundletool consumes come from AGP?).

Alternatively, downgrading to a warning for non-ELF files would probably work.

---

le...@google.com <le...@google.com> #16

Thank you for the feedback, we'll be discussing those options internally.

---

sa...@gmail.com <sa...@gmail.com> #17

> No, I don't intend to disallow executables. Wouldn't those be ELF files as well?

The binaries are ELF, but there may be legitimate use cases for shell scripts other than `wrap.sh`.

---

le...@google.com <le...@google.com> #18

We could possibly allow ELF (regardless of their name) and shell scripts (based on their extension). That would eliminate JARs, classes, and other files that have no place there.

Message last modified on Jun 26, 2020 08:07PM

---

da...@google.com <da...@google.com> #19

Probably still a job best left to AGP, unless there are other tools you're defending against.

---

sa...@gmail.com <sa...@gmail.com> #20

I am afraid this battle is already lost. Unfortunately, even in current master, the 'wrong' names are filtered out. I was wrong that the restriction had been lifted: actually, in Nov'2016 a cha
apps.

The bottom line, to match the **platform** logic, you can let *anything* in the `lib/` directory, but only when the application is debuggable.

---

da...@google.com <da...@google.com> #21

Did some digging and that has *always* been the case. The original change that added packagemanager support for the NDK required the lib prefix, and a 2009 change added the .so suf

That's in contrast to what I'd previously thought, which was that gradle was the source of this restriction. Given that I think changing this behavior in any of the other tools without also chang

---

sa...@gmail.com <sa...@gmail.com> #22

On one hand, this restriction may be removed for debuggable apps. On the other hand, the platform limitation is not relevant when `extractNativeLibs` is **false**. On the *third* hand, native exe
launched when `extractNativeLibs` is **true** (platform restriction again).

And, given that the platform restriction cannot be fixed retroactively, and that it has been this way since the beginning of the times, I'd propose to carefully document all this (including a recor

---

le...@google.com <le...@google.com> #23

Re comment #19, AGP is not the only build system to build App Bundles, so even if AGP adds restrictions, we would still want to keep the restrictions in bundletool.

---

da...@google.com <da...@google.com> #24

> And, given that the platform restriction cannot be fixed retroactively, and that it has been this way since the beginning of the times, I'd propose to carefully document all this (including a rec

That was my thinking too. Filed https://github.com/android/ndk/issues/1300.

lecesne: Given that, anything left to do here? I see above that wrap.sh is already accounted for, so I think this can be closed?

---

le...@google.com <le...@google.com> #25

Is there anything that would prevent developers from extracting the executable themselves from the APK before running it?

---

da...@google.com <da...@google.com> #26

I don't know if there's a location that's both writable (post install) and
executable in the latest releases.

On Tue, Jun 30, 2020, 01:47 lecesne <buganizer-system+lecesne@google.com>
wrote:

- Show quoted text -

---

sa...@gmail.com <sa...@gmail.com> #27

> Is there anything that would prevent developers from extracting the executable themselves from the APK before running it?

Yes, there is. Since API 29, there are no locations that are user-writable and and executable. This change in Android 10 explicitly quoted the W^X violation, so even if there still exists any lo

---

da...@google.com <da...@google.com> #28

https://issuetracker.google.com/160129591 was filed to track the platform restrictions.

---

le...@google.com <le...@google.com>

Status: New

---

je...@google.com <je...@google.com> #29

Assigned to sp...@google.com.

Scott, can you have a look if there is anything left to do for AGP.

---

le...@google.com <le...@google.com> #30

Reassigned to iu...@google.com.

This is probably something we have to do in bundletool to relax these constraints and ensure in a different way that unnecessary files don't end up there.

sa...@gmail.com <sa...@gmail.com> #31

Why do you care that unnecessary files end up there? They can do no harm there: the platform will not extract them, not execute them. They present no more risks than the files in assets or a

le...@google.com <le...@google.com> #32

Some app developers put weird stuff in a directory called "lib", e.g. their library dependency jars. The harm is on end users having to download these jars unnecessarily.

sa...@gmail.com <sa...@gmail.com> #33

Oh, I see. A `lib` directory can happen in the app and intended to be packed at all. Just like `src` directory. But, unlike `src`, it clashes with the name of a zip folder that we need int the bundle.

This does not hold water: the directory that is involved here, is `src/main/resources/lib`. I don't believe that some developer will unintentionally create such directory and populate it with u

sa...@gmail.com <sa...@gmail.com> #34

BTW, I was able to work around this issue by essentially extracting everything with Gradle before passing it over:
https://github.com/bytedeco/javacv/issues/1117#issuecomment-982336622
It would be great to have at least something like this integrated in AGP.

sa...@gmail.com <sa...@gmail.com> #35

Hi Samuel,

I believe what JavaCV was doing with `lib` is exactly the case of "weird stuff" that Pierre described above.