JNI_OnLoad may be called multiple times with no JNI_OnUnload called

Assigned Bug P3 + Add Hotlist adexe s nau

STATUS UPDATE No update yet. Edit

DESCRIPTION hy...@gmail.com created issue #1 Feb 21, 2022 06:44PM

Steps to reproduce the problem:

1. Write a shared library foo WITHOUT JNI_OnLoad defined.
2. Write a shared library bar with JNI_OnLoad defined.
3. Let library foo depends on library bar (`target_link_libraries(foo bar)` in cmake)
4. Load foo in Java code by `System.loadLibrary("foo")`, `JNI_OnLoad` of library bar will be called!
5. Then load bar in Java code by `System.loadLibrary("bar")`, `JNI_OnLoad` of library bar will be called again!

What happened:

1. When we load libfoo.so, JNI_OnLoad of libbar.so will be called, which is unexpected.
2. When we load libbar.so, JNI_OnLoad of libbar.so will be called agian, which is confusing.

What you think the correct behavior should be:

When we load libfoo.so, JNI_OnLoad of libbar.so should NOT be called.

Android Version:

Tested on Android 12, but I think this issue could be reproduced on all android versions.

Example Code:

https://github.com/hyb1996/Issue-JniOnLoad

Others

I think it may be caused by code of `JavaVMExt::LoadNativeLibrary` here, `dlsym()` may return the symbol of the dependency library.

✓ Links (8) Hide a

Links (8)

"https://github.com/hyb1996/Issue-JniOnLoad" hy...@ #1, hy...@ #
"I think it may be caused by code of JavaVMExt::LoadNativeLibrary here , dlsym() may return the symbol of the dependency library." hy...@ #
" ...roid x86/x64 emulator. I think the reason is the Android VM find JNI_OnLoad symbol by native bridge instead of dlsym() on emulator, see code here ." hy...@ #
"For steps to capture a bug report, please refer: https://developer.android.com/studio/debug/bug-report#bugreportdevice" ra...@ #
"I reproduced on OnePlus 7 Pro with Android 11. Here is the screen recording of the reproduce process: issue-220523932.mov ." hy...@ #

See all related links

COMMENTS All comments Oldest first

hy...@gmail.com <hy...@gmail.com> #2 Feb 21, 2022 07:03PM

Another info

This issue will NOT happen on Android x86/x64 emulator. I think the reason is the Android VM find `JNI_OnLoad` symbol by native bridge instead of dlsym() on emulator, see code here.

ra...@google.com <ra...@google.com> Feb 22, 2022 03:12PM

Assigned to ra...@google.com.

ra...@google.com <ra...@google.com> #3 Feb 23, 2022 02:24AM

Thank you for reporting this issue. We've investigated, but unfortunately we have not been able to reproduce it.

For us to further investigate this issue, please provide the following additional information:

Android bug report (to be captured after reproducing the issue)

For steps to capture a bug report, please refer: https://developer.android.com/studio/debug/bug-report#bugreportdevice

Alternate method
Navigate to “Developer options”, ensure “USB debugging” is enabled, then enable “Bug report shortcut”. Capture bug report by holding the power button and selecting the “Take bug report” option.

Note: Please upload the bug report to google drive and share the folder to android-bugreport@google.com, then share the link here.

hy...@gmail.com <hy...@gmail.com> #4 Feb 23, 2022 06:11PM

I reproduced on OnePlus 7 Pro with Android 11. Here is the screen recording of the reproduce process: issue-220523932.mov.

And the bugreport zip file issue-220523932-bugreport-OnePlus7Pro_CH-RKQ1.201022.002-2022-02-23-15-01-26.zip.

The project I used is published on Github: https://github.com/hyb1996/Issue-JniOnLoad

NOTE: **Emulator will not reproduce this issue.**

ra...@google.com <ra...@google.com> Feb 25, 2022 02:52PM

Reassigned to en...@google.com.

en...@google.com <en...@google.com> Feb 26, 2022 02:53AM

Reassigned to ar...@google.com.

ng...@google.com <ng...@google.com> #5 Feb 28, 2022 10:22PM

Reassigned to ma...@google.com.

Are there guarantees on which symbol gets loaded when it gets defined in multiple libraries?

rp...@google.com <rp...@google.com> #6 Mar 2, 2022 02:45PM

> Are there guarantees on which symbol gets loaded when it gets defined in multiple libraries?

The issue is which symbol address is used for each reference (i.e. relocation or dlsym invocation). The general rule (for Android M+) is that symbol lookup starts with a root object (e.g. the executable or handle passed to dlsym/dlopen), then uses the first symbol found in a BFS walk of the libraries. It's expected that `dlsym(foo, "JNI_OnLoad")` will find a symbol exported from a needed library `bar` if `foo` doesn't export it.

That does seem a little unintuitive/wrong. Maybe ART should avoid calling JNI_OnLoad if a library doesn't export it directly? Practically, I guess that would mean using `dladdr` to check the result of `dlsym`, because `dlsym` doesn't have a "don't search dependencies" mode. Of course, we could add a new loader API.

I wonder what OpenJDK does. My guess is that, on glibc Linux, it will behave the same way as Android.

I'm seeing this same behavior everywhere (arm64 device, arm64 app on x86-64 emulator, x86-64 app on x86-64 emulator):

```
03-01 19:34:09.347  5554  5554 D Bar     : JNI_OnLoad: count = 1
03-01 19:34:09.361  5554  5554 D Bar     : JNI_OnLoad: count = 2
```

An x86-64 app on an x86-64 emulator doesn't use the native bridge code path, and AFAIK would have the same behavior here as an arm64 app on an arm64 device.

rp...@google.com <rp...@google.com> #7 Mar 2, 2022 02:58PM

> I wonder what OpenJDK does. My guess is that, on glibc Linux, it will behave the same way as Android.

Yes, on my gLinux (Debian) system, OpenJDK 11.0.14 behaves the same way as Android. i.e. `System.loadLibrary("foo")` calls JNI_OnLoad in libbar.so, and then `System.loadLibrary("bar")` calls it a second time. Further `loadLibrary` calls do not call JNI_OnLoad.

I noticed https://bugs.openjdk.java.net/browse/JDK-8065595, and it is marked Fixed, but I think it's working around an instance of this behavior, not changing it.

hy...@gmail.com <hy...@gmail.com> #8 Mar 7, 2022 02:21PM

I test this issue on Windows 10 with OpenJDK 11.0.11, it does not happen. When I load libfoo.dll, the JNI_OnLoad of libbar.dll is not called. I don't think this behavior is by design, it's more like it wasn't noticed. Changing it may break some existing applications. But because this behavior is really confusing, I think changing it on higher target SDK applications is a reasonable solution.

ma...@google.com <ma...@google.com> #9 Mar 9, 2022 01:43AM

Thanks for the investigation, Ryan.

> The general rule (for Android M+) /.../

Does that mean this regressed in that version?

Even if it did, given the age of M, it's far more likely that apps that are susceptible to this depend (probably inadvertently) on the new behaviour rather than the old pre-M one. Hence I suspect that changing this will have negative app compat impact.

Therefore, even though I agree it's confusing, I'm not sure we should change it. It's straightforward to work around by providing empty JNI_OnLoad (and JNI_OnUnload as well) placeholders, right?

If we don't change it I think it'd be good to document this gotcha somewhere. JNI tips on DAC seems like a suitable page to me, but I'm open for suggestions if there are other landing pages that ought to highlight it.

rp...@google.com <rp...@google.com> #10 Mar 10, 2022 10:48AM

> Does that mean this regressed in that version?

It looks like recursive dlsym searching on a handle was introduced between K and L. This issue happens with L but not K.

> It's straightforward to work around by providing empty JNI_OnLoad (and JNI_OnUnload as well) placeholders, right?

I believe that's correct. I wonder how often JNI_OnUnload is actually needed, though. AFAIK, it would require using a special classloader? There is no API in the loader for removing a linker namespace, even if all its libraries have been unloaded with dlclose.

ma...@google.com <ma...@google.com> #11 Mar 10, 2022 11:14AM

> This issue happens with L but not K.

Thanks, then it's even less likely there are affected apps that predate it.

> I wonder how often JNI_OnUnload is actually needed, though.

I think the same situation as in comment #1 applies there as well - if bar.so has a JNI_OnUnload and foo.so doesn't then the one in bar.so would be called when foo.so is unloaded, which would be equally confusing. So the safe advice is to define both JNI_OnLoad and JNI_OnUnload when there are DT_NEEDED dependencies on other DSO's whose exported symbols may possibly include them.

rp...@google.com <rp...@google.com> #12 Mar 10, 2022 11:36AM

True -- it's more a question of how often a library loaded with System.loadLibrary gets unloaded. e.g. There is no System.unloadLibrary.