signal 11 crash only in Android 8.0 version.

STATUS UPDATE No update yet. Edit

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

Jun 24, 2022 10:32AM

I've been getting reports of repeated crashes with Android 8 users.

The crash only occurs on Android 8.0 version and once the crash occurs, the user can't get out of it because it occurs repeatedly.
I suffered a lot from this crash and did some research and found something interesting.

1) R8 3.2.53 & gradle version 7.1.3 -> No such crash report yet.
2) R8 3.3.50 & gradle version 7.1.3 -> Can see many crash reports.
3) R8 3.2.53 & gradle version 7.2.1 -> Can see many crash reports.
4) R8 3.3.50 & gradle version 7.2.1 -> Can see many crash reports.

The only workaround I found is to revert both the R8 version and the gradle version to the old version.
The above experimental results are all using app bundles and Kotlin.

The Kotlin version also seems to have an effect on error occurrence.

1) R8 3.2.53 & gradle version 7.1.3 & Kotlin 1.6.10 -> No such crash report yet.
2) R8 3.2.53 & gradle version 7.1.3 & Kotlin 1.7.0 -> Can see many crash reports.

Crash Detail :

```
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR)
bool art::mirror::Class::ResolvedMethodAccessTest<true, true, (art::InvokeType)0>(art::ObjPtr<art::mirror::Class>, art::ArtMethod*,
unsigned int, art::ObjPtr<art::mirror::DexCache>)

  #00 pc 0000000000272da4 /system/lib64/libart.so (bool art::mirror::Class::ResolvedMethodAccessTest<true, true,
(art::InvokeType)0>(art::ObjPtr<art::mirror::Class>, art::ArtMethod*, unsigned int, art::ObjPtr<art::mirror::DexCache>)+356)
  #00 pc 00000000002a6b60 /system/lib64/libart.so (bool art::interpreter::DoInvoke<(art::InvokeType)0, false, true>(art::Thread*,
art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+496)
  #00 pc 000000000029e3bc /system/lib64/libart.so (art::JValue art::interpreter::ExecuteSwitchImpl<true, false>(art::Thread*,
art::DexFile::CodeItem const*, art::ShadowFrame&, art::JValue, bool)+45964)
  #00 pc 000000000026bffc /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*,
art::ShadowFrame&, art::JValue, bool)+668)
  #00 pc 0000000000272804 /system/lib64/libart.so (art::interpreter::ArtInterpreterToInterpreterBridge(art::Thread*,
art::DexFile::CodeItem const*, art::ShadowFrame*, art::JValue*)+212)
  #00 pc 000000000028bd38 /system/lib64/libart.so (bool art::interpreter::DoCall<false, false>(art::ArtMethod*, art::Thread*,
art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+648)
  #00 pc 00000000002a71d8 /system/lib64/libart.so (bool art::interpreter::DoInvokeVirtualQuick<false>(art::Thread*,
art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+520)
  #00 pc 000000000029daac /system/lib64/libart.so (art::JValue art::interpreter::ExecuteSwitchImpl<true, false>(art::Thread*,
art::DexFile::CodeItem const*, art::ShadowFrame&, art::JValue, bool)+43644)
  #00 pc 000000000026bffc /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*,
art::ShadowFrame&, art::JValue, bool)+668)
  #00 pc 0000000000272804 /system/lib64/libart.so (art::interpreter::ArtInterpreterToInterpreterBridge(art::Thread*,
art::DexFile::CodeItem const*, art::ShadowFrame*, art::JValue*)+212)
  #00 pc 000000000028bd38 /system/lib64/libart.so (bool art::interpreter::DoCall<false, false>(art::ArtMethod*, art::Thread*,
art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+648)
  #00 pc 0000000000515c40 /system/lib64/libart.so (MterpInvokeInterface+1744)
  #00 pc 000000000051f194 /system/lib64/libart.so (ExecuteMterpImpl+14740)
  #00 pc 000000000026bf20 /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*,
art::ShadowFrame&, art::JValue, bool)+448)
  #00 pc 0000000000508890 /system/lib64/libart.so (artQuickToInterpreterBridge+1472)
  #00 pc 0000000000536a1c /system/lib64/libart.so (art_quick_to_interpreter_bridge+92)
  #00 pc 000000000100799c /system/framework/arm64/boot-framework.oat (offset 0x68d000)
(android.os.AsyncTask$InternalHandler.handleMessage+348)
  #00 pc 000000000073e09c /system/framework/arm64/boot-framework.oat (offset 0x68d000)
(android.os.Handler.dispatchMessage+188)
  #00 pc 000000000109ed30 /system/framework/arm64/boot-framework.oat (offset 0x68d000) (android.os.Looper.loop+1184)
  #00 pc 0000000000758968 /system/framework/arm64/boot-framework.oat (offset 0x68d000)
(android.app.ActivityThread.main+1160)
  #00 pc 000000000052d838 /system/lib64/libart.so (art_quick_invoke_static_stub+600)
  #00 pc 00000000000d86c4 /system/lib64/libart.so (art::ArtMethod::Invoke(art::Thread*, unsigned int*, unsigned int, art::JValue*,
char const*)+260)
  #00 pc 000000000044ed08 /system/lib64/libart.so (art::InvokeWithArgArray(art::ScopedObjectAccessAlreadyRunnable const&,
art::ArtMethod*, art::ArgArray*, art::JValue*, char const*)+104)
  #00 pc 0000000000450aa0 /system/lib64/libart.so (art::InvokeMethod(art::ScopedObjectAccessAlreadyRunnable const&,
_jobject*, _jobject*, _jobject*, unsigned long)+1456)
  #00 pc 00000000003d48a4 /system/lib64/libart.so (art::Method_invoke(_JNIEnv*, _jobject*, _jobject*, _jobject*)+52)
```

Reporter	de...@gmail.com
Type	Bug
Priority	P3
Severity	S3
Status	Assigned
Access	Default access View
Assignee	ze...@google.com
Verifier	--
Collaborators	
CC	be...@google.com de...@gmail.com ng...@google.com r8...@googlegroups.com ss...@google.com ze...@google.com
AOSP ID	--
Blocking Release	--
Release Status	--
Found In	--
Targeted To	--
Verified In	--
In Prod	

Show 1 additional field

```
  #00  pc 0000000000267f64  /system/framework/arm64/boot.oat (offset 0x1dc000) (java.lang.Class.getDeclaredMethodInternal
[DEDUPED]+180)
  #00  pc 00000000018e5168  /system/framework/arm64/boot-framework.oat (offset 0x68d000)
(com.android.internal.os.Zygote$MethodAndArgsCaller.run+136)
  #00  pc 00000000018e9eb8  /system/framework/arm64/boot-framework.oat (offset 0x68d000)
(com.android.internal.os.ZygoteInit.main+3336)
  #00  pc 000000000052d838  /system/lib64/libart.so (art_quick_invoke_static_stub+600)
  #00  pc 00000000000d86c4  /system/lib64/libart.so (art::ArtMethod::Invoke(art::Thread*, unsigned int*, unsigned int, art::JValue*,
char const*)+260)
  #00  pc 000000000044ed08  /system/lib64/libart.so (art::InvokeWithArgArray(art::ScopedObjectAccessAlreadyRunnable const&,
art::ArtMethod*, art::ArgArray*, art::JValue*, char const*)+104)
  #00  pc 000000000044e878  /system/lib64/libart.so (art::InvokeWithVarArgs(art::ScopedObjectAccessAlreadyRunnable const&,
_jobject*, _jmethodID*, std::__va_list)+424)
  #00  pc 000000000035af04  /system/lib64/libart.so (art::JNI::CallStaticVoidMethodV(_JNIEnv*, _jclass*, _jmethodID*,
std::__va_list)+628)
  #00  pc 00000000000fb050  /system/lib64/libandroid_runtime.so (_JNIEnv::CallStaticVoidMethod(_jclass*, _jmethodID*, ...)+128)
  #00  pc 00000000000fd318  /system/lib64/libandroid_runtime.so (android::AndroidRuntime::start(char const*,
android::Vector<android::String8> const&, bool)+840)
  #00  pc 00000000000025d4  /system/bin/app_process64 (main+1636)
  #00  pc 000000000001b94c  /system/lib64/libc.so (__libc_init+92)
  #00  pc 0000000000001ec8  /system/bin/app_process64 (do_arm64_start+80)
```

✓ Links (3) Hide all

Links (3)

"Also, if possible, could you share the compiler input dump for the non-working build?"	ze...@ #2, ze...@ #12, ze...@ #17
"Yes, this could be https://android-review.googlesource.com/c/platform/art/+/1823219 ."	ng...@ #4
" https://cs.android.com/search?q=profileinstaller%20path:.grad..."	ga...@ #13

COMMENTS All comments Oldest first

ra...@google.com <ra...@google.com> Jun 24, 2022 08:21PM

Assigned to ra...@google.com.

ra...@google.com <ra...@google.com> Jun 24, 2022 09:35PM

Reassigned to an...@google.com.

ze...@google.com <ze...@google.com> #2 Jun 24, 2022 10:21PM

Reassigned to ze...@google.com.

Thanks for the report.

Are you able to reproduce this issue locally on a device?

Do you have exact details on which device(s) and OS version(s) you are seeing this crash on?

Would it be possible to share the APK build for the version you have found to work and one of the versions you are getting the crashes on?

Also, if possible, could you share the compiler input dump for the non-working build?

You can share any of the above privately with zerny@google.com and/or sgjesse@google.com

ze...@google.com <ze...@google.com> #3 Jun 24, 2022 10:21PM

CC ngeoffray: does the stack above look like an issue you know about?

ng...@google.com <ng...@google.com> #4 Jun 25, 2022 02:48AM

Yes, this could be https://android-review.googlesource.com/c/platform/art/+/1823219.

de...@gmail.com <de...@gmail.com> #5 Jun 27, 2022 09:51AM

Reply to zerny@google.com

I cannot reproduce this issue because not all Android 8 devices will have this issue.

But on a device that occurs once, it happens over and over again.
It makes my loyal users leave my app.

Message last modified on Jun 27, 2022 09:51AM

A8.png

427 KB View Download

ze...@google.com <ze...@google.com> #6 Jun 27, 2022 05:46PM

If it is an instance of the issue linked in #comment4, it looks very difficult for D8/R8 to do any general workaround for as the compiler does not have any knowledge of classloader relationships between classes.

To the reporter:

1. Do you know if your application is doing any kind of special classloading?
2. Do you have any indication of which code is being executed when this crashes? The suspected issue is in a code path guarded by a normal accessibility check, so if you can deduce where the problematic invoke is you may be able to ensure that path is public and use `-keep` rules to retain that access property. A possible workaround may be to use R8 with `-allowaccessmodification` which will make as much of the program `public` as is possible. Bit of a shot in the dark however.

If you can share the APK that is causing the crashes, we may be able to reproduce the issue and help find a solution/workaround. You can attach it publicly here or share it in private with zerny@google.com and/or sgjesse@google.com

de...@gmail.com <de...@gmail.com> #7 Jun 30, 2022 09:34AM

1. No special classloading in my application, but I have integrate Meta Audience Network SDK and I know that it has (*.dex) file in their library.
2. According to my users, the app is forced to close 2-3 seconds after starting the app.

de...@gmail.com <de...@gmail.com> #8 Aug 1, 2022 11:38AM

I guess this issue is related to Google Drive API.

com.google.apis:google-api-services-drive:v3-rev20220709-1.32.1

jc...@nextdoor.com <jc...@nextdoor.com> #9 Feb 15, 2023 08:13AM

Was a solution found for this issue? I'm currently experiencing a similar stacktrace on Android 8 devices when using proguard-android-optimize.txt to minify our builds. I also haven't been able to reproduce it locally.

Message last modified on Feb 15, 2023 10:40AM

ze...@google.com <ze...@google.com> #10 Feb 15, 2023 11:26PM

Unfortunately, there has not been any progress on this issue as we do not currently know of a way to workaround the issue in the runtime.

jc...@nextdoor.com <jc...@nextdoor.com> #11 Feb 17, 2023 06:13AM

Thanks for the update. Using proguard-android-optimize.txt significantly improves Jetpack Compose performance so I feel like an issue like this should be escalated. This could become a large blocker to apps that want to migrate. Is there additional documentation around that shows correct versions to use for AGP, R8 and Kotlin? We were using the following config.

AGP: 7.4.0
Gradle: 7.6.0
Kotlin: 1.7.20

We're updating to Kotlin 1.8, will report back on if this issue happens again.

Message last modified on Feb 17, 2023 06:14AM

crash_devices.png
235 KB View Download

ze...@google.com <ze...@google.com> #12 Feb 17, 2023 07:58PM

We have some ideas for how we might work around this issue, but likely the work around would only be applicable when compiling with R8 where we have full world knowledge.

To evaluate a potential workaround it would be very useful if we could obtain a reproduction of the issue. Would it be possible for you to share a compiler input dump for the build that produces the APK that triggered this issue in the wild? The exact APK build would also be helpful so we can verify our hypothesis on the code patterns triggering this issue. You can share both in private with zerny@google.com and/or christofferqa@google.com

Could you check if you have `-allowaccessmodification` in your proguard config. Doing so allows R8 to make more classes public and our understanding of this issue is that it triggers only when a non-public class method is indirectly targeted via a public sub-class. If you do have that option set, you could update the other keep rules in your config to have the `-keepX,allowaccessmodification` option on the rules which will allow R8 to make them public even when targeted by the keep rule.

ga...@linecorp.com <ga...@linecorp.com> #13 Feb 22, 2023 07:53PM

Would you try to exclude `profileinstaller` library?

```
// app/build.gradle
configurations.all {
    exclude group: "androidx.profileinstaller", module: "profileinstaller"
}
```

With recent androidx libraries, `profileinstaller` library is added as transitive dependency.
https://cs.android.com/search?q=profileinstaller%20path:.gradle&ss=androidx

So, it seems JIT compiler performs for unreached code forcibly, yet by `profileinstaller` when app is started.

In my case, I met same case, but after exclude `profileinstaller` library from app, crashes were gone.

jc...@nextdoor.com <jc...@nextdoor.com> #14 Feb 25, 2023 04:03AM

Thank you all for following up on this. It looks like this is still occurring with Kotlin 1.8.

I don't have `-allowaccessmodification` set in the proguard-config but it seems to be in `proguard-android-optimize-7.4.1`. Does it need to be set in our own config as well?

That's an interesting find with `profileinstaller`. I have some questions
- How were you able to find out that it was this dependency that was causing the issue?
- Were you able to find out what the unreachable code was and why it's only on Android 8? Wondering if there's a proguard rule we can add instead of excluding a dependency.

jc...@nextdoor.com <jc...@nextdoor.com> #15 Mar 14, 2023 06:08AM

Wanted to follow up on this crash, we actually managed to find a fix. After thinking about `-allowaccessmodification`, we realized we had some very generic keep rules that could be causing this issue. After removing this, the crashers went away.

```
-keep class kotlin.** { *; }
-keep class org.jetbrains.** { *; }
```

This makes sense as this is very generic and not specific to our project's code so it seems something like profile installer was expecting this code under this package to be updated by R8. This keep rule also isn't necessary as it probably prevents a lot of shrinking and minifying from happening in the first place.

If anybody else is running into a similar issue I encourage you to look at your own keep rules and see if there is anything that might be too generic. Thanks for all the support!

Message last modified on Mar 14, 2023 08:16AM

kr...@mobisystems.com <kr...@mobisystems.com> #16 Sep 5, 2023 03:01PM

After the last update we have A LOT of crashes of this type which are only on Android 8 devices.

```
backtrace:
  #00  pc 0x0000000000263c78  /system/lib64/libart.so (bool art::mirror::Class::ResolvedMethodAccessTest<true, true, (art::InvokeType)0>(art::ObjPtr<art::mirror::Class>, art::ArtMethod*, unsigned int, art::ObjPtr<art::mirror::DexCache>)+352)
  #01  pc 0x0000000000297288  /system/lib64/libart.so (bool art::interpreter::DoInvoke<(art::InvokeType)0, false, true>(art::Thread*, art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+508)
  #02  pc 0x000000000028c204  /system/lib64/libart.so (art::JValue art::interpreter::ExecuteSwitchImpl<true, false>(art::Thread*, art::DexFile::CodeItem const*, art::ShadowFrame&, art::JValue, bool)+36540)
  #03  pc 0x000000000025d0dc  /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*, art::ShadowFrame&, art::JValue, bool)+664)
  #04  pc 0x00000000004e227c  /system/lib64/libart.so (artQuickToInterpreterBridge+1468)
  #05  pc 0x000000000051001c  /system/lib64/libart.so (art_quick_to_interpreter_bridge+92)
  #06  pc 0x0000000000506e38  /system/lib64/libart.so (art_quick_invoke_static_stub+600)
  #07  pc 0x00000000000d804c  /system/lib64/libart.so (art::ArtMethod::Invoke(art::Thread*, unsigned int*, unsigned int, art::JValue*, char const*)+260)
  #08  pc 0x0000000000134734  /system/lib64/libart.so (art::ClassLinker::InitializeClass(art::Thread*, art::Handle<art::mirror::Class>, bool, bool)+3392)
  #09  pc 0x000000000011e3cc  /system/lib64/libart.so (art::ClassLinker::EnsureInitialized(art::Thread*, art::Handle<art::mirror::Class>, bool, bool)+172)
  #10  pc 0x000000000027f7f8  /system/lib64/libart.so (art::ResolveVerifyAndClinit(art::dex::TypeIndex, art::ArtMethod*, art::Thread*, bool, bool)+256)
  #11  pc 0x00000000004d7718  /system/lib64/libart.so (artInitializeStaticStorageFromCode+64)
  #12  pc 0x0000000000507738  /system/lib64/libart.so (art_quick_initialize_static_storage+168)
  #13  pc 0x00000000014962f8  /data/app/com.package-D7Jwh6OqGThOTW9ZGPHFuw==/oat/arm64/base.odex
```

ze...@google.com <ze...@google.com> #17 Sep 5, 2023 04:36PM

Thanks for reporting that you are also hit by this.

Would it be possible for you to share a compiler input dump for the build that produces the APK that triggered this issue in the wild? The exact APK build would also be helpful so we can verify our hypothesis on the code patterns triggering this issue.

You can share both in private with zerny@google.com and/or christofferqa@google.com

After obtaining the above, you can try and workaround the issue. First see if you can add `-allowaccessmodification` in your proguard config (if don't already have it). If you still hit the issue with that, you could update the other keep rules in your config to have the `-keepX,allowaccessmodification` option on the rules. That may successfully work around the issue by making more of the methods public.

kr...@mobisystems.com <kr...@mobisystems.com> #18 Sep 5, 2023 05:29PM

Thank you for the fast response!

We have already enabled "allowaccessmodification" as we include the default "proguard-android-optimize.txt" file but the issue still exists. What do you mean by "keepX" ?

Message last modified on Sep 5, 2023 05:30PM

ze...@google.com <ze...@google.com> #19 Sep 5, 2023 06:58PM

Sorry for the not so clear suggestion. It is for updating the various rules you may have to also allow access modification, e.g.,

```
-keep class <pattern>
becomes:
-keep,allowaccessmodfication class <pattern>

-keepclasseswithmembers class <pattern>
becomes:
-keepclasseswithmembers,allowaccessmodfication class <pattern>

-keepclassmembers class <pattern>
becomes:
-keepclassmembers,allowaccessmodfication class <pattern>
```

etc.

kr...@mobisystems.com <kr...@mobisystems.com> #20 Sep 6, 2023 01:05AM

We have also encourted another crash which may be the same as it also happens on Android 8 only and on the same set of devices:

```
Thread:  art_method.cc:612] Check failed: existing_entry_point != nullptr java.lang.String[]
android.icu.impl.ICUResourceBundle.getStringArray()@0x7739fb40


backtrace:
  #00  pc 0x0000000000069d88  /system/lib64/libc.so (tgkill+8)
  #01  pc 0x000000000001de90  /system/lib64/libc.so (abort+88)
  #02  pc 0x0000000000436954  /system/lib64/libart.so (art::Runtime::Abort(char const*)+528)
  #03  pc 0x0000000000437064  /system/lib64/libart.so (art::Runtime::Aborter(char const*)+24)
  #04  pc 0x0000000000521628  /system/lib64/libart.so (android::base::LogMessage::~LogMessage()+900)
  #05  pc 0x00000000000d9eb0  /system/lib64/libart.so (art::ArtMethod::GetOatQuickMethodHeader(unsigned long)+584)
  #06  pc 0x00000000001b4c2c  /system/lib64/libart.so (art::FaultManager::IsInGeneratedCode(siginfo*, void*, bool)+508)
  #07  pc 0x00000000001b4824  /system/lib64/libart.so (art::FaultManager::HandleFault(int, siginfo*, void*)+92)
  #08  pc 0x0000000000002df0  /system/bin/app_process64 (art::SignalChain::Handler(int, siginfo*, void*)+536)
  #09  pc 0x000000000000068c  [vdso:0000007098557000]
  #10  pc 0x0000000000263c74  /system/lib64/libart.so (bool art::mirror::Class::ResolvedMethodAccessTest<true, true,
(art::InvokeType)0>(art::ObjPtr<art::mirror::Class>, art::ArtMethod*, unsigned int, art::ObjPtr<art::mirror::DexCache>)+348)
  #11  pc 0x0000000000297288  /system/lib64/libart.so (bool art::interpreter::DoInvoke<(art::InvokeType)0, false, true>
(art::Thread*, art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+508)
  #12  pc 0x000000000028c204  /system/lib64/libart.so (art::JValue art::interpreter::ExecuteSwitchImpl<true, false>
(art::Thread*, art::DexFile::CodeItem const*, art::ShadowFrame&, art::JValue, bool)+36540)
  #13  pc 0x000000000025d0dc  /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*,
art::ShadowFrame&, art::JValue, bool)+664)
  #14  pc 0x00000000004e227c  /system/lib64/libart.so (artQuickToInterpreterBridge+1468)
  #15  pc 0x000000000051001c  /system/lib64/libart.so (art_quick_to_interpreter_bridge+92)
  #16  pc 0x0000000000506e38  /system/lib64/libart.so (art_quick_invoke_static_stub+600)
  #17  pc 0x00000000000d804c  /system/lib64/libart.so (art::ArtMethod::Invoke(art::Thread*, unsigned int*, unsigned int,
art::JValue*, char const*)+260)
  #18  pc 0x0000000000134734  /system/lib64/libart.so (art::ClassLinker::InitializeClass(art::Thread*,
art::Handle<art::mirror::Class>, bool, bool)+3392)
  #19  pc 0x000000000011e3cc  /system/lib64/libart.so (art::ClassLinker::EnsureInitialized(art::Thread*,
art::Handle<art::mirror::Class>, bool, bool)+172)
  #20  pc 0x000000000027f7f8  /system/lib64/libart.so (art::ResolveVerifyAndClinit(art::dex::TypeIndex, art::ArtMethod*,
art::Thread*, bool, bool)+256)
  #21  pc 0x00000000004d7718  /system/lib64/libart.so (artInitializeStaticStorageFromCode+64)
  #22  pc 0x0000000000507738  /system/lib64/libart.so (art_quick_initialize_static_storage+168)
  #23  pc 0x0000000001155f38  /data/app/package-_VL_HZlUsuYevOHGH0UQPQ==/oat/arm64/base.odex
```

kr...@mobisystems.com <kr...@mobisystems.com> #21 Sep 15, 2023 09:02PM

Hi,
We recently shared dump files and apks with you to help for resolving the issue with the R8 shrinker. We hope you have received the files.

Best Regards!

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

Sep 29, 2023 10:29PM

One of the users with Android 8.0 has a similar problem. The symptoms are the same, the application closes a few seconds after launch.

```
backtrace:
  #00  pc 0x0000000000272e84  /system/lib64/libart.so (bool art::mirror::Class::ResolvedMethodAccessTest<true, true,
(art::InvokeType)0>(art::ObjPtr<art::mirror::Class>, art::ArtMethod*, unsigned int, art::ObjPtr<art::mirror::DexCache>)+356)
  #01  pc 0x00000000002a6c40  /system/lib64/libart.so (bool art::interpreter::DoInvoke<(art::InvokeType)0, false, true>
(art::Thread*, art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+496)
  #02  pc 0x000000000029e49c  /system/lib64/libart.so (art::JValue art::interpreter::ExecuteSwitchImpl<true, false>
(art::Thread*, art::DexFile::CodeItem const*, art::ShadowFrame&, art::JValue, bool)+45964)
  #03  pc 0x000000000026c0dc  /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*,
art::ShadowFrame&, art::JValue, bool)+668)
  #04  pc 0x00000000002728e4  /system/lib64/libart.so (art::interpreter::ArtInterpreterToInterpreterBridge(art::Thread*,
art::DexFile::CodeItem const*, art::ShadowFrame*, art::JValue*)+212)
  #05  pc 0x000000000028be18  /system/lib64/libart.so (bool art::interpreter::DoCall<false, false>(art::ArtMethod*,
art::Thread*, art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+648)
  #06  pc 0x00000000002a72b8  /system/lib64/libart.so (bool art::interpreter::DoInvokeVirtualQuick<false>(art::Thread*,
art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+520)
  #07  pc 0x000000000029db8c  /system/lib64/libart.so (art::JValue art::interpreter::ExecuteSwitchImpl<true, false>
(art::Thread*, art::DexFile::CodeItem const*, art::ShadowFrame&, art::JValue, bool)+43644)
  #08  pc 0x000000000026c0dc  /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*,
art::ShadowFrame&, art::JValue, bool)+668)
  #09  pc 0x00000000002728e4  /system/lib64/libart.so (art::interpreter::ArtInterpreterToInterpreterBridge(art::Thread*,
art::DexFile::CodeItem const*, art::ShadowFrame*, art::JValue*)+212)
  #10  pc 0x000000000028c5b0  /system/lib64/libart.so (bool art::interpreter::DoCall<false, true>(art::ArtMethod*,
art::Thread*, art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+1520)
  #11  pc 0x00000000002a5310  /system/lib64/libart.so (bool art::interpreter::DoInvoke<(art::InvokeType)1, false, true>
(art::Thread*, art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+656)
  #12  pc 0x000000000029cbcc  /system/lib64/libart.so (art::JValue art::interpreter::ExecuteSwitchImpl<true, false>
(art::Thread*, art::DexFile::CodeItem const*, art::ShadowFrame&, art::JValue, bool)+39612)
  #13  pc 0x000000000026c0dc  /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*,
art::ShadowFrame&, art::JValue, bool)+668)
  #14  pc 0x00000000002728e4  /system/lib64/libart.so (art::interpreter::ArtInterpreterToInterpreterBridge(art::Thread*,
art::DexFile::CodeItem const*, art::ShadowFrame*, art::JValue*)+212)
  #15  pc 0x000000000028be18  /system/lib64/libart.so (bool art::interpreter::DoCall<false, false>(art::ArtMethod*,
art::Thread*, art::ShadowFrame&, art::Instruction const*, unsigned short, art::JValue*)+648)
  #16  pc 0x0000000000518388  /system/lib64/libart.so (MterpInvokeVirtualQuick+680)
  #17  pc 0x0000000000523214  /system/lib64/libart.so (ExecuteMterpImpl+29972)
  #18  pc 0x000000000026c000  /system/lib64/libart.so (art::interpreter::Execute(art::Thread*, art::DexFile::CodeItem const*,
art::ShadowFrame&, art::JValue, bool)+448)
  #19  pc 0x0000000000508dc0  /system/lib64/libart.so (artQuickToInterpreterBridge+1472)
  #20  pc 0x000000000053701c  /system/lib64/libart.so (art_quick_to_interpreter_bridge+92)
  #21  pc 0x0000000000e42e38  /data/app/monitor.kmv.multinotes-xWYKUm7nA6XPln0MGi4ReQ==/oat/arm64/base.odex
```