


Android Public Tracker304071459

ART segfaults while performing CallVoidMethod after updating to August 2023 security patch+16Hotlists (1)Mark as Duplicate


Comments (8)DependenciesDuplications (0)Blocking (0)Resources (8)

WAI Bug P3+ Add Hotlist

 STATUS UPDATE

No update yet.

Edit

 DESCRIPTION

ni...@gmail.com created issue #1

I'm the developer of [Strato Emulator](#) and lately I've been receiving reports of native crashes, seemingly after the August 2023 security patch.

From the debugging I was able to perform, the crashes can be traced back inside `libart.so`, and it seems to happen after it tries to call a Java void method from C++ code via `CallVoidMethod`

The Java method has the following signature `(Ljava.lang.Void)V`, and I'm guessing the issue has something to do with parameters being passed, since removing the `jLongArray` parameter seems to fix

I also noticed that pausing execution with the debugger before performing the call somehow makes the call successful? I have no idea how that would be possible, but that surely hints at some assertion fail message is printed to logcat at the same timestamp the stack trace is printed (stack trace created by our code, information is limited because of how we have to handle signals during execution)

```
thread_list.cc:1314] Check failed: self->GetState() != ThreadState::kRunnable (self->GetState()=Runnable, ThreadState::kRunnable=Runnable)
Signal: Segmentation fault (PC: 0x9D4B3938)
Stack Trace:
* 0x9D4B3938
* 0x74CEFEF485
* 0x69B292EC70
```

Here's the call site, which then goes into `libart.so` and segfaults: <https://github.com/strato-emu/strato/blob/b0207ab6456499448e8b9cb552410864c866e15f/app/src/main/cpp/skyline/jvm.cpp>


Here's the Java method that should be called: <https://github.com/strato-emu/strato/blob/b0207ab6456499448e8b9cb552410864c866e15f/app/src/main/java/emu/skyline/EmulationActivity.kt>

The crashes also affect users who are still running an older version of the app that didn't cause any problems before. Our previous app, [Skyline Emulator](#), which Strato is based on, also exhibit


This issue is a continuation from issue <https://issuetracker.google.com/issues/302680512>, where more details can be found. It was suggested sign-/zero-extend ops being possibly at fault, here

I have checked the disassembly of `JvmManager::VibrateDevice()` and there doesn't appear to be any sign-/zero-extend ops being performed on arguments before branching to `libART.so` so exactly.

✓ Mentioned issues (1) ✓ Links (7)

 Mentioned issues (1)

P3 Lots of native crashes since August 2023 / Android 13+ "<https://issuetracker.google.com/302680512>"

 Links (7)

"I'm the developer of [Strato Emulator](#) and lately I've been receiving reports of native crashes, seemingly after the August 2023 security patch."

"Here's the call site, which then goes into `libart.so` and segfaults: <https://github.com/strato-emu/strato/blob/b0207ab6456499448e8b9cb552410864c866e15f/app/src/main/cpp/skyline/jvm.cpp#L1>


"Here's the Java method that should be called: <https://github.com/strato-emu/strato/blob/b0207ab6456499448e8b9cb552410864c866e15f/app/src/main/java/emu/skyline/EmulationActivity.kt#L6>

"...es also affect users who are still running an older version of the app that didn't cause any problems before. Our previous app, [Skyline Emulator](#), which Strato is based on, also exhibits the same


"<https://cs.android.com/android/platform/superproject/main/+/main:art/runtime/>..."

See all related links

COMMENTS

 vm...@google.com <vm...@google.com>


Assigned to hb...@google.com.

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

AFAICT, the initial (presumably?) CHECK failure probably corresponds to the CHECK at the beginning of `ThreadList::Unregister`. In the current source tree:

[https://cs.android.com/android/platform/superproject/main/+/main:art/runtime/thread\\_list.cc:l=1336](https://cs.android.com/android/platform/superproject/main/+/main:art/runtime/thread_list.cc:l=1336)

It's unlikely we can do much here without more of a stack trace. It would be nice to see who is calling `Unregister`, e.g. via `DetachCurrentThread`, from a state indicating that we're still accessing

 ni...@gmail.com <ni...@gmail.com> #3

I was also able to trace that message back to the ART runtime source code, but I didn't really know what to do with it. Any suggestions on how I can get a stack trace in the `Unregister` function or set a breakpoint in there to look at the stack trace though, maybe there's a way to dump ART symbols in the debugger so that I can retrieve an address to set a breakpoint to?

In the meantime, I've also had users report the same type of crash (segfault) on a different JNI call, here: <https://github.com/strato-emu/strato/blob/b0207ab6456499448e8b9cb552410864> but I'm still trying to collect more data about it.



**en...@google.com** <en...@google.com> [#4](#)

you want a stack trace to see where you are? on new enough versions of Android, <execinfo.h> is your friend there: <https://cs.android.com/android/platform/superproject/+/-/master:bionic/lib>



**ni...@gmail.com** <ni...@gmail.com> [#5](#)

After further testing, I have confirmed the `thread_list.cc:1314` `Check failed` assertion fail to be unrelated to the issue I'm experiencing. It was happening after the segfault, because of signal handler as part of a destructor call of a thread-local object.

I'm still at a loss, with no idea how to approach the issue since pausing the execution in the debugger right before entering JNI code seems to fix the issue most of the time. I'm trying to get a running open-source homebrews.



**hb...@google.com** <hb...@google.com> [#6](#)

I don't think we can help without a lot more information. A `DetachCurrentThread()` call from inside a signal handler is certainly bad news. So is, at least officially, any access to a `thread_local`



**ni...@gmail.com** <ni...@gmail.com> [#7](#)

Final update: it turns out it was our fault. Because of the way we run games we are hooking and calling `libc`'s `sigaction` directly to set a signal handler for SIGSEGV from the game before a pointer would be intact when the signal handler is called.

We had no issue with this until recently, when the Android runtime was probably updated to rely on catching signals for some of its internal operations.

I've added a fallback to the old signal handler if the fault address was outside of the emulator's memory, and stuff seems to work correctly.

Thank you for your time.



**hb...@google.com** <hb...@google.com> [#8](#)

*Status: Won't Fix (Intended Behavior)*

Thanks for letting us know. I'll go ahead and close this.