... we gave a mouse an NDK

some non android developers' experience with NDK

Armin Ronacher
Director of Engineering, Sentry
@mitsuhiko

Bruno Garcia
Senior Software Engineer, Sentry
@brungarc
our NDK experience was a bit of an unexpected rabbit hole
let's talk about us
we're a stack trace company
### Error Information

**EXC_BAD_ACCESS / KERN_INVALID_ADDRESS**

Fatal Error: EXC_BAD_ACCESS / KERN_INVALID_ADDRESS

<table>
<thead>
<tr>
<th>mechanism</th>
<th>minidump</th>
<th>handled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>

### Code Snippet

```c
12. void initialize_memory(char *mem) {
13.     sentry_add_breadcrumb(sentry_value_new_breadcrumb(0, "Initializing memory");
14.     memset(mem, 1, 100);
15. }
```

### Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r14</td>
<td>0x0000000000000000</td>
</tr>
<tr>
<td>r15</td>
<td>0x0000000000000000</td>
</tr>
<tr>
<td>r12</td>
<td>0x0000000000000000</td>
</tr>
</tbody>
</table>

**Show More**

### Other Frames

- `example_cra... 0x0001024a8a8 example_crashpad:14` (startup) 
- `example_cra... 0x0001024a8ce3 main` (main) 

Called from: `libdyld <unknown>`
Armin Ronacher

Director of Engineering

@mitsuhiiko

Python & Rust Developer
Bruno Garcia
Senior Software Engineer
@brungarc
.NET Developer
what do we have to do with Android anyways?
You probably know Android better than we do.
But we know quite a few things about crash reporting
The goal: stack traces for C, C++, Java, Kotlin, ...
NDK
What NDK is

NDK gives us native (C/C++/etc.) code on Android

It interacts heavily with the JVM (ART) via JNI

Android NDK's environment is Linux-ish
NDK Components

What's it based on:

Bionic for libc

some hand picked common libraries (zlib)
we already did Java, we already did C++, …

but we didn't do NDK.
Production Crash Reporting
Production Crash Reporting is Fighting a Paradigm
Production Crash Reporting

Performance and debuggability are often at odds

The lower level the language, the higher the disparity between debug and production build performance

The performance gains come at cost of debuggability
production is all that matters
(for us)
Production on Android
The Runtimes
“Java Runtime”
&
“C Runtime”
Java Runtime

Android Runtime

Runs via some layers of indirection Java bytecode. Resembles mostly what you get on a traditional JVM.

Specifically you get stack traces from the runtime system from every exception thrown
C Runtime

Very low level, bare minimums.

Interactions with Java via JNI

No native support for producing useful stack traces, dozens of different unwinders for Android non built-in that are good.
Stack Traces
Readable Java Stack Traces

Proguard/R8 obfuscation make stack traces unreadable

Mapping files can be used to resolve method names in stack traces back to the original names.
Readable C Stack Traces

A whole different ballpark.

DWARF information is generally used to restore location information and method names in stack traces once we have them.

To get them in the first place is tricky.
turning numbers and funny strings into stuff humans can comprehend
Java is easy because Java stack traces are good
Proguard mappings:

a.b.c:2 -> was.WeirdThing.method
class name: a.b.C -> io.sentry.FooBar
method name: a -> doSomeFoo
line number: 42
Preventing Obfuscation
-keep public class * extends java.lang.Exception
-keep class com.example.myapp.MyBridge { *, }
But C ...
How do we get a stack trace?
stack walk or memory dump?
the problem of unwinding
unwinding memory dumps
Facebook can upload the entire files of all system libraries to their server through their Android apps.

The app compresses each system library file using gzip and uploads them to server.

Interestingly, the files are uploaded to a specific collection that’s related to my phone.

```
POST /[ID FOR MY PHONE] HTTP/1.1
Authorization: OAuth [redacted my access token]
X-FB-Friendly-Name: Upload library to GLC
Host: graph.facebook.com
Content-Type: multipart/form-data; boundary=xxxx
Content-Length: 692804

// @wongmjane
--xxxx
Content-Disposition: form-data; name="filetype"
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit

1
--xxxx
Content-Disposition:
form-data; name="lib"; filename="llbsqlite.so.gz"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
```
okay ... so what can we do?
stack walk on device
stackwalkers

libcorkscrew
  deprecated, 32bit only

libunwind
  deprecated, google provides android patches

libunwindstack
  C++ monstrosity, actively maintained
libunwindstack

requires custom patches to compile with NDK

requires large sigaltstack to not overflow the stack in the signal handler

development in android master deviated from most NDK compatible forks
gief stackwalker

android can already stackwalk (see ndk-stack)

why is the stack walker not exposed to us?
build id and image addresses

now we need the GNU build id and the image offset for each loaded executable / dynamic library

normally one would use dl_iterate_phdr

this one is missing on older NDKs,

Workaround: parse /proc/self/maps
sigaltstack / async safety
static const size_t SIGNAL_STACK_SIZE = 65536;
stack_t g_signal_stack;

g_signal_stack.ss_sp = malloc(SIGNAL_STACK_SIZE);
g_signal_stack.ss_size = SIGNAL_STACK_SIZE;
g_signal_stack.ss_flags = 0;
sigaltstack(&g_signal_stack, 0);
all we want is a symbol server
Putting it Together
NDK side

sentry-native

> SDK hooks signal handler
> enumerate loaded images
> dump state to disk before crash
  - stack walk with libunwindstack
SDK side

sentry-android

> watches file system for new events
> deserializes them, enhances them and uploads
Server side

> process crash reports
  - symbolicate native stacks on symbolicator
  - check for well known symbols in our buckets
  - resolve proguard for java stacks
> store
Shipping It
Android Gradle Plugin :'( 
Structure

> cmake builds libraries per platform
  - these end up in folders for each architecture

where do the headers go?
how do we link to the libraries?
Do The Ugly Dance

> needs a gradle plugin to
  - copy header libs out of AAR :( 
  - so that code can link against the native lib

github.com/android/ndk-samples/issues/261
https://github.com/android/ndk/issues/916
Improving It
NDK asks

> a maintained and included stack walker
> make ucontext_t/getcontext available
> add support for shipping libs/headers in AARs
> Have OEMs/Google provide symbol servers