Vulnerability Research of Android apps: from 0 to 0-day using fuzzing

DEF CON Russia | DCG7812

GitHub: **saruman9** Telegram: **@dura_lex**

August 10, 2024

About me

- **Vulnerability Researcher**: IoT, ICS, embedded, mobile, etc.
- **System Developer**: tools for automatic analysis, observability systems, fuzzers, emulators, etc.

Agenda

Beginning
Telegram
Viber
WhatsApp
General summary

The further down the list, the more difficult it is…

Background

- Money

Background

- Money
- BugBounty

Background

- Money
- BugBounty
- Mobile Applications, OSs, HW

Background

- Money
- BugBounty
- Mobile Applications, OSs, HW
- Test Task

Beginning

Disclaimer

- No WEB vulnerabilities
- No Java vulnerabilities
- No vulnerabilities in protocols and specifications

Disclaimer

- No WEB vulnerabilities
- No Java vulnerabilities
- No vulnerabilities in protocols and specifications

- Memory corruptions
- Binary vulnerabilities
- RCE and data-only exploits

Not about exploitation or a specific vulnerability/CVE, but the methodology

Without meme, sry 😢

APK Analysis

What’s interesting for me?

Android manifest file:

- Activities
- Services
- Broadcasts Receivers
- Content Providers
- Permissions

Resources:

- Libraries
- DSL parsers
- Protocol Buffers files
- Custom binary blobs

Java Decompilation:

- Deobfuscation
- Refactoring
- Analysis (control-flow, data-flow, etc.)

Shared/Native Libraries — my main target

Is source code exist?

- Telegram — YES
- Viber — NO
- WhatsApp — NO

Telegram

Why first?

- Source Code — ✅

Why first?

- Source Code — ✅
- BugBounty — ✅

Why first?

- Source Code — ✅
- BugBounty — ✅
- I’m a user of the app — ✅

Static Analysis

- Manifest file — ✅
- Resources — ❌

- Java
- C++

$\Longrightarrow$ Android Studio

Shared/Native libraries

Ordered by the “low-hanging fruit” principle:

1. Legacy code
2. Self-written components
3. …
5. Crypto implementation
6. …
9. Popular open-source frameworks
10. RFC, protocols and manifests

Plan

1. Translate the code architecture into a convenient format (mind map, graph, wiki, Zettelkasten, etc.)

Plan

1. Translate the code architecture into a convenient format (mind map, graph, wiki, Zettelkasten, etc.)
2. Identify the entry points and sinks

Plan

1. Translate the code architecture into a convenient format (mind map, graph, wiki, Zettelkasten, etc.)
2. Identify the entry points and sinks
3. Building attack vectors

Plan

1. Translate the code architecture into a convenient format (mind map, graph, wiki, Zettelkasten, etc.)
2. Identify the entry points and sinks
3. Building attack vectors
4. Isolating target components

Plan

1. Translate the code architecture into a convenient format (mind map, graph, wiki, Zettelkasten, etc.)
2. Identify the entry points and sinks
3. Building attack vectors
4. Isolating target components
5. Analysis (fuzzing in our case)

Attack Vectors & Components

- File parsers and decoders
  - FLAC
  - GIF
  - Opus
  - Lottie (modified)

- Connection
  - `tgnet`
  - `TLObject` — (de)serialization (legacy)
- VoIP
  - `tgcalls` (legacy)
  - WebRTC (modified)
- etc.

Fuzzing

Harness

Do you have the source code?

Harness

Do you have the source code?

YES ✅

Harness

Do you have the source code? YES ✅ $\Longrightarrow$ easy peasy lemon squeezy?..

NO

- Isolation is not always possible in complex components

- Isolation is not always possible in complex components
- Behavior emulation (sockets, files, server, protocol, Java code, etc.)

- Isolation is not always possible in complex components
- Behavior emulation (sockets, files, server, protocol, Java code, etc.)
- Legacy code is a legacy code

- Isolation is not always possible in complex components
- Behavior emulation (sockets, files, server, protocol, Java code, etc.)
- Legacy code is a legacy code
- Build for Android or for a host x86_64 POSIX machine?

Example. `tgnet`

1. Replace Android code
2. ~~Modify~~Write CMake file
3. Develop server[1] (MTProto[2]) and emulate Java code & socket file
4. Develop a MitM PoC attack for triaging

[1] https://github.com/saruman9/tg_srv
[2] https://github.com/saruman9/010_editor_templates

Fuzzers

- AFL/AFL++
- libFuzzer / centipede / fuzztest
- honggfuzz
- LibAFL
- etc.

Catches

- DoS
- Leaks
- Cryptography weaknesses
- Vulnerabilities in open source components

Summary

- Not so interesting for the presentation, but important as a base
- Not good BugBounty program
- Good for a first research in this field
- Many other methods of analysis can be applied

Viber

Rakuten Viber

Sources

- Препарируем Viber. Мини-гид по анализу приложений для Android[1] © *Хакер*
- fuzzer + harness[2]

[1] https://xakep.ru/2023/05/16/analyzing-viber/
[2] https://github.com/saruman9/viber_linkparser_fuzzer/

Static Analysis

- Manifest file — ✅
- Resources — ❌
- 1-day analysis + binary diffing — ✅

Shared/Native Libraries

Architecture — x86_64

- More tools
- Emulation at high speeds
- Partial analysis on a host machine

Native functions

- IDA Pro
- Binary Ninja
- rizin

Native functions

- ~~IDA Pro~~
- ~~Binary Ninja~~
- ~~rizin~~

```
$ readelf -W --demangle --symbols $(LIBRARY_SO) | \
tail -n +4 | \
sort -k 7 | \
# optional rg "FUNC.*Java_.*"
less
```

```
33: 0000000000001bb5    55 FUNC    GLOBAL DEFAULT   13 Java_com_viber_libnativehttp_HttpEngine_nativeCreateHttp
34: 0000000000001bec    15 FUNC    GLOBAL DEFAULT   13 Java_com_viber_libnativehttp_HttpEngine_nativeDelete
38: 0000000000001bfb   622 FUNC    GLOBAL DEFAULT   13 Java_com_viber_libnativehttp_HttpEngine_nativeTest
44: 00000000000018c3   109 FUNC    GLOBAL DEFAULT   13 Java_com_viber_libnativehttp_NativeDownloader_nativeOnConnected
39: 00000000000015e8   366 FUNC    GLOBAL DEFAULT   13 Java_com_viber_libnativehttp_NativeDownloader_nativeOnData
35: 0000000000001b0c    40 FUNC    GLOBAL DEFAULT   13 Java_com_viber_libnativehttp_NativeDownloader_nativeOnDisconnected
40: 0000000000001930   476 FUNC    GLOBAL DEFAULT   13 Java_com_viber_libnativehttp_NativeDownloader_nativeOnHead
```

```
$ rg "native.*nativeCreateHttp"
```

```
app/src/main/java/com/viber/libnativehttp/HttpEngine.java
9:    public static native long nativeCreateHttp();
```

Goals:

- Find open source components
- Find the target library
- Superficial analysis

Attack Vectors & Components

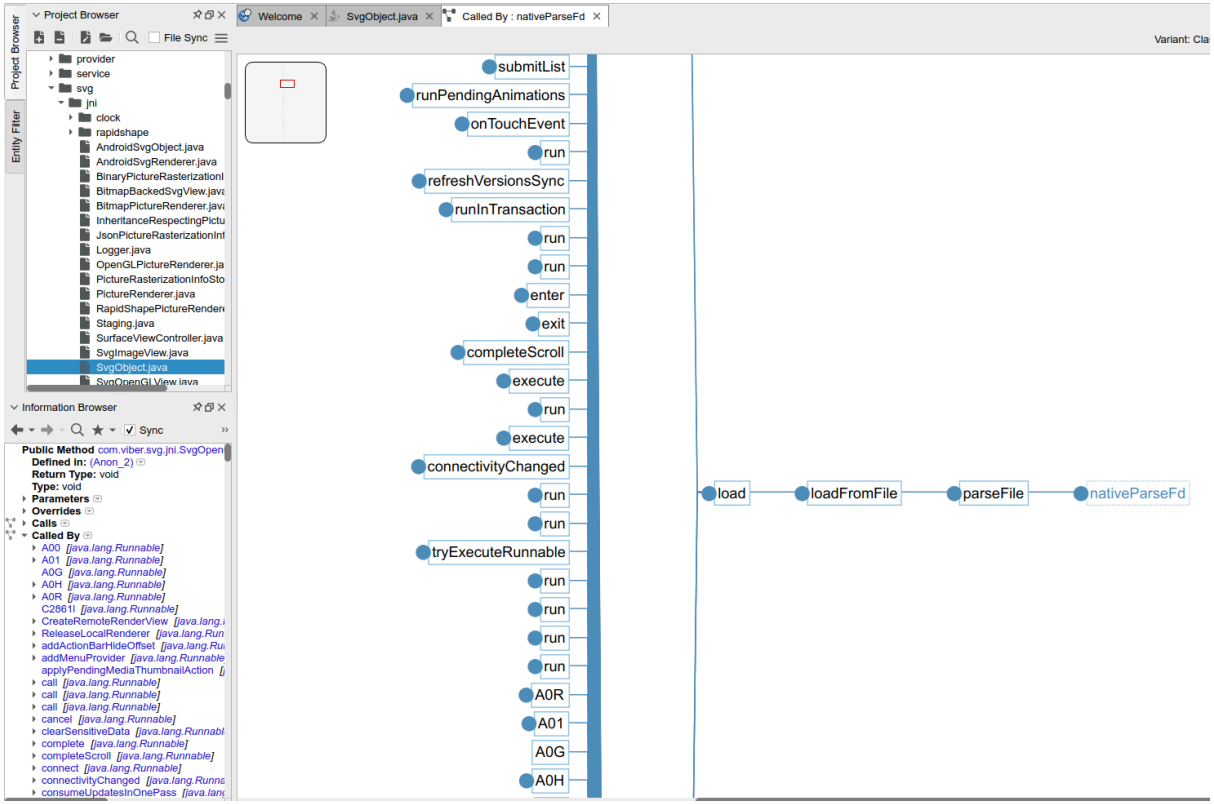
- Link parser
- SVG
- Viber RTC (WebRTC)
- VoIP engine

Accessibility (real sink?)

Static Analysis

- jadx[1] — decompilation IntelliJ IDEA — deobfuscation, refactoring
- SciTools Understand[2] — code-flow, data-flow analysis
- `strings`/`rizin`, `grep`/`ripgrep`

[1] https://github.com/skylot/jadx
[2] https://www.scitools.com/



Call graph of SVG native function in Understand

Dynamic Analysis

- Frida and public scripts
- `frida-trace`
- Smali patching
- Binary patching

Fuzzing

Greybox is more interesting

- libFuzzer / centipede / fuzztest

Greybox is more interesting

- ~~libFuzzer / centipede / fuzztest~~

Greybox is more interesting

- ~~libFuzzer / centipede / fuzztest~~
- honggfuzz

Greybox is more interesting

- ~~libFuzzer / centipede / fuzztest~~
- ~~honggfuzz~~

Greybox is more interesting

- ~~libFuzzer / centipede / fuzztest~~
- ~~honggfuzz~~
- AFL++

Greybox is more interesting

- ~~libFuzzer / centipede / fuzztest~~
- ~~honggfuzz~~
- AFL++
- LibAFL
- etc.

Fuzzer	Instru-mentation	Emulator (x86_64)	Real device, aarch64
AFL++[1]	Frida	✅ AFL++ in	✅
AFL++[2]	Qemu	❌	✅
AFL++[3]	Qemu	✅	❌
AFL++	Unicorn + qiling	Unicorn	❌/✅?
honggfuzz/ AFL++	QBDI	QBDI	❌/✅?
LibAFL	Qemu	❌	✅

LibAFL	Qemu	✅	❌
LibAFL	Frida	✅ LibAFL in	✅

[1] **Android greybox fuzzing with AFL++ Frida mode** by Eric Le Guevel from Quarkslab
[2] **AFL++ on Android with QEMU support** by Itai Greenhut (@Gr33nh4t) from Aleph Research; **fpicker-aflpp-android** by marcinguy
[3] **MMS Exploit Part 2: Effective Fuzzing of the Qmage Codec** by Mateusz Jurczyk from Project Zero; **Sloth** by ant4g0nist

LibAFL + Frida

Why? Later we will review the remaining options

- I'm Rust developer
- I've already used LibAFL
- Frida is true cross platform software
- Rust is better for cross-compilation[1]

[1] not for Android, but not because Rust is bad

Harness

Reverse Engineering

Ghidra

- Ghidra fork[1]
- `ghidra_scripts`[2]
- Recaster plugin[3]
- “Ghidra. Dev” presentation[4]

[1] https://github.com/saruman9/ghidra
[2] https://github.com/saruman9/ghidra_scripts
[3] https://github.com/saruman9/recaster
[4] https://github.com/saruman9/ghidra_dev_pres

- Binary Ninja — `binja_snippets`[1]
- IDA Pro
- rizin

[1] https://github.com/saruman9/binja_snippets

C++

Ghidra

- `RecoverClassesFromRTTIScript.java`
- `ApplyClassFunctionSignatureUpdatesScript.java`
- `ApplyClassFunctionDefinitionUpdatesScript.java`
- C++ directory in Script Manager
- Ghidra-Cpp-Class-Analyzer[1] by astrelsky

[1] https://github.com/astrelsky/Ghidra-Cpp-Class-Analyzer

IDA Pro

- ida_medigate[1] by Metadorius — fork of fork of fork…
- Referee[2] by joeleong — a python port of James Koppel's Referee

[1] https://github.com/Metadorius/ida_medigate
[2] https://github.com/joeleong/ida-referee

Binary Ninja

- Use development release channel
- ClassyPP[1] by CySHell
- binja_itanium_cxx_abi[2] by whitequark

[1] https://github.com/CySHell/ClassyPP
[2] https://github.com/whitequark/binja_itanium_cxx_abi

```
liblinkparser.bndb
Types  Search types
0018     int64_t statusCode;
0020     int64_t contentLength;
0028     struct jni_string redirectedUrl;
0040     void* field_40;
0048     int128_t field_48;
0058 };

     struct __packed LinkParser::AndroidHttp
     {
0000     struct LinkParser_vtbl* vtbl;
0008     jobject http;
0010     JNIEnv* env;
0018 };

     struct __packed LinkParser::PushParserDownloader
     {
0000     struct LinkParser::PushParserDownloader_vtbl* vtbl;
0008     struct inner_LinkParser::PushParserDownloader* inner;
0010 };

     struct __packed LinkParser::PushParserDownloader_vtbl
     {
0000     void* LinkParser::PushParserDownloader::dtr;
0008     void* LinkParser::~PushParserDownloader;
0010     void* field_10;
0018     void* field_18;
0020     void* field_20;
0028     void (* w_init_inner_push_parser_downloader)(struct LinkParser::PushParserDow
0030     void* field_30;
0038 };

Cross References
Filter (0)
```

```
ELF ▾  Linear ▾  High Level IL ▾

00011db0  jobject nativeGeneratePreview(JNIEnv* env, jstring url, jobject http)

00011dbd  void* fsbase
00011dbd  int64_t rax = *(fsbase + 0x28)
00011dcb  jobject r15 = nullptr
00011ddd  if (url != 0 && http != 0)
00011df1      struct jni_string jni_url
00011df1      jni_getUtf8Bytes(dst: &jni_url, env: env, jstr: url)
00011e01      struct LinkParser::AndroidHttp link_parser_http
00011e01      LinkParser::AndroidHttp(android_http: &link_parser_http, env: env, http: http)
00011e15      struct jni_string preview_jni_str
00011e15      sub_18e00(preview_jni_str: &preview_jni_str, jni_url: &jni_url, http: &link_parser_http)
00011e1d      struct jni_string dst_jni_str
00011e1d      dst_jni_str.small_len.o = 0
00011e21      dst_jni_str.buf = 0
00011e2f      if ((preview_jni_str.small_len & 1) == 0)
00011e36          dst_jni_str.buf = preview_jni_str.buf
00011e40          dst_jni_str.small_len.o = preview_jni_str.small_len.o
00011e46      else
00011e46          uint64_t rbx_2 = preview_jni_str.len
00011e4f          if (rbx_2 u>= 0xfffffffffffffff0)
00011f1d              w_throw()
00011f1d              noreturn
00011e55          void* r15_1 = preview_jni_str.buf
00011e5e          char* r12_1
00011e5e          if (rbx_2 u>= 0x17)
00011e76              uint64_t r13_2 = (rbx_2 + 0x10) & 0xfffffffffffffff0
00011e7d              char* rax_3 = operator new(r13_2)
00011e82              r12_1 = rax_3
00011e85              dst_jni_str.buf = rax_3
00011e8e              dst_jni_str.small_len.q = r13_2 | 1
00011e92              dst_jni_str.len = rbx_2
00011e63          else
```

BTW try Binary Ninja[1]

[1] A very old comparison of IDA and Binary Ninja — **Binary Ninja 1.1.1184-dev vs IDA Pro 7.0.171130 (RU)**

Signatures

- Ghidra: Function ID[1]
- IDA Pro: lumen[2] — Lumina private server
- Binary Ninja: Signature Libraries[3]

[1] **FunctionID** help topic
[2] **https://github.com/naim94a/lumen**
[3] **https://binary.ninja/2020/03/11/signature-libraries.html**

Diffing

- Version Tracking[1] in Ghidra
- Program Differences[2] in Ghidra
- BinDiff[3]
- Diaphora[4]

[1] **Version Tracking** help topic
[2] **Program Differences** help topic
[3] **https://www.zynamics.com/bindiff.html**
[4] **https://github.com/joxeankoret/diaphora**

Difficulties & Resolving

- ? Java + C++
- ? Threads
- ? Other shared libraries as dependencies
- ? Initialization in `JNI_OnLoad`

Difficulties & Resolving

❓ Java + C++

💡 Find “pure” functions

❓ Threads

❓ Other shared libraries as dependencies

❓ Initialization in `JNI_OnLoad`

Difficulties & Resolving

✅ Java + C++

❓ Threads

❓ Other shared libraries as dependencies

❓ Initialization in `JNI_OnLoad`

Difficulties & Resolving

✅ Java + C++

❓ Threads

💡 Find a target function in a call graph without threads

❓ Other shared libraries as dependencies

❓ Initialization in `JNI_OnLoad`

Difficulties & Resolving

✅ Java + C++

✅ Threads

❓ Other shared libraries as dependencies

❓ Initialization in `JNI_OnLoad`

Difficulties & Resolving

✅ Java + C++

✅ Threads

❓ Other shared libraries as dependencies

💡 To do patching of shared libraries

💡 Load dependencies inside harness code

❓ Initialization in `JNI_OnLoad`

Difficulties & Resolving

✅ Java + C++

✅ Threads

✅ Other shared libraries as dependencies

❓ Initialization in `JNI_OnLoad`

Difficulties & Resolving

✅ Java + C++

✅ Threads

✅ Other shared libraries as dependencies

❓ Initialization in `JNI_OnLoad`

💡 Write stubs, call initialization functions

Difficulties & Resolving

- ✅ Java + C++
- ✅ Threads
- ✅ Other shared libraries as dependencies
- ✅ Initialization in `JNI_OnLoad`

Example of a harness for the target function

```
const ptrdiff_t ADDR_JNI_ONLOAD = 0x0000000000011640;
const ptrdiff_t ADDR_PARSE_LINK = 0x000000000002F870;
const ptrdiff_t ADDR_COPY_JNI_STRING_FROM_STR = 0x0000000000011160;
[...]
Functions *load_functions()
{
  LIBC_SHARED = dlopen("/data/local/tmp/libc++_shared.so", RTLD_NOW | RTLD_GLOBAL);
  LIBICU_BINDER = dlopen("/data/local/tmp/libicuBinder.so", RTLD_NOW | RTLD_GLOBAL);
  LIBLINKPARSER = dlopen("/data/local/tmp/liblinkparser.so", RTLD_NOW | RTLD_GLOBAL);
  if (LIBLINKPARSER != NULL && LIBC_SHARED != NULL && LIBICU_BINDER != NULL)
  {
    int (*JNI_OnLoad)(void *, void *) = dlsym(LIBLINKPARSER, "JNI_OnLoad");
    void (*binder_init)() = dlsym(LIBICU_BINDER, "_ZN22IcuSqliteAndroidBinder4initEv");

    if (JNI_OnLoad != NULL && binder_init != NULL /* && binder_getInstance != NULL */)
    {
[...]
```

Catches

- DoS
- Leaks
- Deadlocks

Sources (once again)

- Препарируем Viber. Мини-гид по анализу приложений для Android[1] © *Хакер*
- fuzzer + harness[2]

[1] https://xakep.ru/2023/05/16/analyzing-viber/
[2] https://github.com/saruman9/viber_linkparser_fuzzer/

Summary

- More details in the article 👆
- The research has been interrupted, so go ahead!
- The basic things for graybox fuzzing were considered, further — more

WhatsApp

Static Analysis

- Manifest file — ✅
- Resources — ✅, see the next slide
- 1-day analysis + binary diffing — ✅

Shared/Native Libraries

Superpack

Android app compression, which combines compiler analysis with data compression.

See *Superpack: Pushing the limits of compression in Facebook's mobile apps*[1] by Sapan Bhatia from Facebook.

[1] https://engineering.fb.com/2021/09/13/core-data/superpack/

Solutions:

- Reverse engineering and developing
- Reverse engineering and developing a wrapper (calling functions from a shared library in an emulator)
- Decompression in an emulator/Docker

Solutions:

- Reverse engineering and developing
- Reverse engineering and developing a wrapper (calling functions from a shared library in an emulator)
- ✅ Decompression in an emulator/Docker

Attack Vectors & Components

- Java part
- Many open-source components
- `libwhatsapp.so`
  - Statically linked
  - More and more Rust

After a long CFG/DFG analysis…

- MP4 checking (incoming messages), converting (outgoing messages)
- GIF checking
- WEBP parsing (stickers)
- `libmagi` (MIME type identification)
- VoIP (PJSIP project[1])

[1] https://github.com/pjsip/pjproject

Fuzzing

AFL++ + Frida

- Not as hard to build for Android as I expected[1]
- Perfect for those who prefer C++
- Not as flexible as LibAFL, but rich in functionality

[1] https://github.com/saruman9/AFLplusplus/tree/android

LibAFL + Frida

Android NDK + Frida + Rust = Building is the real pain!

Works: Rust 1.67, NDK 22, clang30

Doesn’t work:

- Rust 1.67, NDK 25, clang*
- Rust 1.70, NDK 21, clang*
- Rust 1.70, NDK 22, clang*
- Rust 1.70, NDK 25, clang*

1. Moving Android toolchains from libgcc to libclang_rt[1]
2. Updating the Android NDK in Rust 1.68[2]
3. Fixing build error for NDK 23 and above[3]
4. Patches for Frida (only for NDK below 23)[4]
5. Workaround for aarch64 `__clear_cache` issue[5]
6. A dirty hack for `frida-rust`[6]

[1] https://github.com/android/ndk/wiki/Changelog-r23#changes
[2] https://blog.rust-lang.org/2023/01/09/android-ndk-update-r25.html
[3] https://github.com/rust-lang/rust/pull/85806#issuecomment-1096266946
[4] https://github.com/AFLplusplus/LibAFL/issues/1359#issuecomment-1693328137
[5] https://github.com/AFLplusplus/LibAFL/issues/1359#issuecomment-1695346506
[6] https://github.com/frida/frida-rust/pull/112

LibAFL problems

- DrCov coverage doesn’t work as expected[1,2]
- Asan doesn’t work for Android x86_64[3]
- miniBSOD doesn’t work for Android x86_64[4]

[1] https://github.com/AFLplusplus/LibAFL/pull/1579
[2] https://github.com/AFLplusplus/LibAFL/pull/1581
[3] https://github.com/AFLplusplus/LibAFL/pull/1578
[4] https://github.com/AFLplusplus/LibAFL/pull/1577

- Additional changes[1]:
  - Option to continue fuzzing
  - Catching of timeout objectives
  - Option to disable coverage
  - The option of minimizing a corpus

[1] https://github.com/saruman9/LibAFL/branches/all

Frida

- I had a lot of problems because I didn’t understand how Stalker works. Especially when analyzing complex objects (JIT is terrible)
- Be sure to read the documentation[1] for Stalker (and Gum interface) before using it

[1] https://frida.re/docs/stalker/

Frida

- I had a lot of problems because I didn’t understand how Stalker works. Especially when analyzing complex objects (JIT is terrible)
- Be sure to read the documentation[1] for Stalker (and Gum interface) before using it
- LibAFL + Frida = Multithreading doesn’t work
- The sanitizer based on Frida doesn’t work correctly on some arch/platforms

[1] https://frida.re/docs/stalker/

Java VM

Harness = Java + Native Libraries

But how?

Java VM

Harness = Java + Native Libraries

But how?

Create Java VM from C/C++/Rust code of a harness/ fuzzer!

Sources:

- Creating a Java VM from Android Native Code[1] by Caleb Fenton
- Calling JNI Functions with Java Object Arguments from the Command Line[2] by Caleb Fenton
- Loading Android ART virtual machine from native executables[3] by Eugene Gershnik

[1] https://calebfenton.github.io/2017/04/05/creating_java_vm_from_android_native_code/
[2] https://calebfenton.github.io/2017/04/14/calling_jni_functions_with_java_object_arguments_from_the_command_line/
[3] https://gershnik.github.io/2021/03/26/load-art-from-native.html

Where hell begins?

Creating a Java VM is a non-trivial task!

Where hell begins?

Compliance with all legacy designs in Android is hard!

I did a separate research on the ASOP source code

Where hell begins?

Running Java VM under a fuzzer and Frida is a pain!

I spent many hours debugging

Where hell begins?

A real device and an emulator are two different things!

I have used 3 real devices and countless versions of an emulator

Where hell begins?

It still doesn't work stably…

But it works![1,2]

[1] **https://github.com/saruman9/jnienv**

[2] BTW Valgrind for Android: **https://github.com/saruman9/valgrind**

Smali patching

Does anyone know a tool that is comfortable to use for Smali patching?

Smali patching

- Smali the Parseltongue Language[1] by Benoît Forgette from Quarkslab
- Ghidra
- Binary Ninja
- Smalise extension for VSCode[2] by LoyieKing

[1] https://blog.quarkslab.com/smali-the-parseltongue-language.html
[2] https://github.com/LoyieKing/Smalise

Catches

DoS. Sender side

- Gallery
  - TIFF, SVG
  - OGG, WAV, MP3
- Live
  - Opus (audio recorder)
  - Video stream from a camera
- Sending
  - MP4

DoS. Receiver side

- Media hijacking
- Android Java exceptions, native iOS crashes

Summary

- Only DoS… yet
- VoIP is still waiting for me

General summary

- The journey is 1 year long

General summary

- The journey is 1 year long
- From zero to ~~hero~~ some bugs with only fuzzing

General summary

- The journey is 1 year long
- From zero to ~~hero~~ some bugs with only fuzzing
- This is the vulnerability research for now, next time — exploitation development

Thank you!