# File format fuzzing in Android: Giving Stagefright to the Android installer

Alexandru Blanda
Intel OTC Security SQE

# Agenda

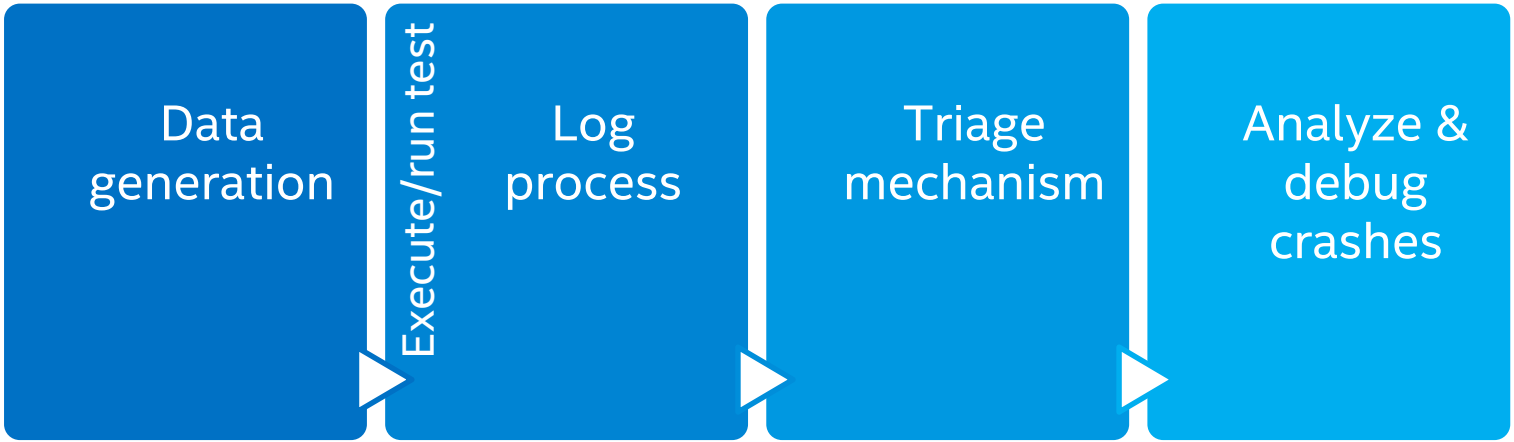
- File format fuzzing in Android
- Fuzzing the Stagefright media framework
- Fuzzing the Android application installer
- Fuzzing with AFL in Android

# File format fuzzing in Android

# File format fuzzing in Android

- Possible targets
  - Media Players
  - Document Viewers
  - Web Browsers
  - Antivirus products
  - Binary (ELF)

# Steps


Data generation
Execute/run test
Log process
Triage mechanism
Analyze & debug crashes

# Data generation

- Mutational vs. generational fuzzing
- Evolutionary fuzzing
- Tools
  - Basic Fuzzing Framework (BFF) / zzuf
  - FuzzBox
  - Radamsa
  - American Fuzzy Lop (AFL)*

# Log process

- Log every test case with fatal priority

```
$ adb shell log  -p F -t <Component> <test_case_index> *** <reproducibility_info>
```

- Log template

```
$ adb shell logcat -v time *:F

01-16 17:46:12.240 F/<Component> (PID):  <test_case_index> ***
<reproducibility_info>
01-16 17:46:19.676 F/<Component> (PID):  <test_case_index> ***
<reproducibility_info>
17:46:24.405 F/libc (8321): Fatal signal 11 (SIGSEGV) at 0x18
(code=1), thread 831 (process_name)
01-16 17:46:25.128 F/<Component> (PID):  <test_case_index> ***
<reproducibility_info>
```

# Triage mechanism

- Input that produces a crash generates an entry in /data/tombstones and /data/system/dropbox

```
pid: 3438, tid: 3438, name: stagefright  >>> stagefright <<<
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr deadbaad
eax b3ee0ff8  ebx b7b18f38  ecx b7b1d900  edx b3ee0ff8
    esi 8004d748  edi af6d4dee
    xcs 00000073  xds 0000007b  xes 0000007b  xfs 00000000  xss 0000007b
    eip b7a7202c  ebp bffff418  esp bffff3d0  flags 00010286

backtrace:
    #00  pc 0001402c  /system/lib/libc.so (dlfree+1948)
    #01  pc 000dcf1c  /system/lib/libstagefright.so
(android::MediaBuffer::~MediaBuffer()+108)
    #02  pc 000dd6eb  /system/lib/libstagefright.so
(android::MediaBuffer::release()+267)
```

# Triage mechanism

1. Parse generated logs
   - Identify input that causes crashes
2. Execute input to confirm crash
3. For each identified input
   - Grab generated tombstone
   - Parse tombstone – get the PC value
   - Check if PC value has been previously encountered
   - Save tombstone and input if issue is unique

# Analyze and debug crashes

- /data/tombstones

```
pid: 3438, tid: 3438, name: stagefright  >>> stagefright <<<
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr deadbaad
eax b3ee0ff8  ebx b7b18f38  ecx b7b1d900  edx b3ee0ff8
    esi 8004d748  edi af6d4dee
    xcs 00000073  xds 0000007b  xes 0000007b  xfs 00000000  xss 0000007b
    eip b7a7202c  ebp bffff418  esp bffff3d0  flags 00010286

backtrace:
    #00  pc 0001402c  /system/lib/libc.so (dlfree+1948)
    #01  pc 000dcf1c  /system/lib/libstagefright.so
(android::MediaBuffer::~MediaBuffer()+108)
    #02  pc 000dd6eb  /system/lib/libstagefright.so
(android::MediaBuffer::release()+267)
```

# Analyze and debug crashes

- Dmesg

```
<6>[73801.130320] stagefright[12469]: segfault at 14 ip
00000000f72a5fff sp 00000000fff98710 error 4 in
libstagefright.so[f71c6000+1b5000]

<6>[73794.579462] stagefright[12455]: segfault at c ip
00000000f728bcfe sp 00000000ff9d6f90 error 6 in
libstagefright.so[f71e8000+1b5000]
```

```
 * Page fault error code bits:
 *   bit 0 ==  0: no page found      1: protection fault
 *   bit 1 ==  0: read access        1: write access
 *   bit 2 ==  0: kernel-mode access 1: user-mode access
 *   bit 3 ==                        1: use of reserved bit
 *   bit 4 ==                        1: instruction fetch fault
```

# Analyze and debug crashes

- gdbserver (on device) / gdb (on local machine)

```
$ gdbserver :5039 --attach <process_pid>
    OR
$ gdbserver :5039 /path/to/executable <options> (ex: gdbserver
:5039 /system/bin/stagefright -a file.mp3)
-----------------------------------------------------------------
$ adb forward tcp:5039 tcp:5039
$ gdb
    (gdb) target remote :5039 (from the gdb shell)
    (gdb) continue (to resume process execution)
-----------------------------------------------------------------
(gdb) set solib-absolute-prefixdb </path/to/tree/symbols>
(gdb) set solib-search-path </path/to/tree/symbols/system/lib>
```

# Analyze and debug crashes

- addr2line

```
backtrace:
    #00  pc 0001402c  /system/lib/libc.so (dlfree+1948)
    #01  pc 0000d630  /system/lib/libc.so (free+32)
    #02  pc 000dcf1c  /system/lib/libstagefright.so
(android::MediaBuffer::~MediaBuffer()+108)
```

```
$ addr2line -f -e
/path/to/tree/out/target/product/<product_id>/symbols/system/li
b/libstagefright.so 000dcf1c
```

# Fuzzing the Stagefright media framework

# Media files as attack vectors

Binary streams containing complex data

Large variety of audio and video players and associated media codecs

User perception that media files are harmless

Media playback doesn’t require special permissions

# Overview of testing process

- Create corrupt but structurally valid media files
- Direct test cases to the appropriate decoders
- Monitor the system for potential issues
- Pass the issues through a triage mechanism

# The Stagefright CLI

- /data/tombstones

```
root@android:/ # stagefright -h
usage: stagefright
-h(elp)
-a(udio)
-m max-number-of-frames-to-decode in each pass
-p(rofiles) dump decoder profiles supported
-t(humbnail) extract video thumbnail or album art
-s(oftware) prefer software codec
-r(hardware) force to use hardware codec
-o playback audio
-w(rite) filename (write to .mp4 file)
-S allocate buffers from a surface
-T allocate buffers from a surface texture
-d(ump) filename (raw stream data to a file)
-D(ump) filename (decoded PCM data to a file)
```

# Stagefright log

- Media files corrupted using BFF

```
04-14 05:02:07.698 F/Stagefright(20222): - sp_stagefright ***
958 - Filename:zzuf.32732.c8jZzT.mp4

04-14 05:02:13.382 F/Stagefright(20255): - sp_stagefright ***
959 - Filename:zzuf.26772.zh7c8g.mkv

04-14 05:02:13.527 F/libc    (20256): Fatal signal 11
(SIGSEGV), code 1, fault addr 0x0 in tid 20256 (stagefright)

04-14 05:02:20.820 F/Stagefright(20270): - sp_stagefright ***
960 - Filename:zzuf.12260.ayDuIA.mpg

04-14 05:02:21.259 F/Stagefright(20281): - sp_stagefright ***
961 - Filename:zzuf.6488.F8drye.mp4
```

# Results

- Initial fuzzing campaigns started in March 2014
- Results were extremely surprising: thousands of crashes per week (triage mechanism)
- First severe issues in the September 2014 Android security bulletin:
  - Integer overflows in libstagefright (CVE-2014-7915, CVE-2014-7916, CVE-2014-7917)
- The tool was open-sourced in February 2015:
  - https://github.com/fuzzing/MFFA
- Currently used as a complementary solution alongside AFL

# Fuzzing the Android application installer

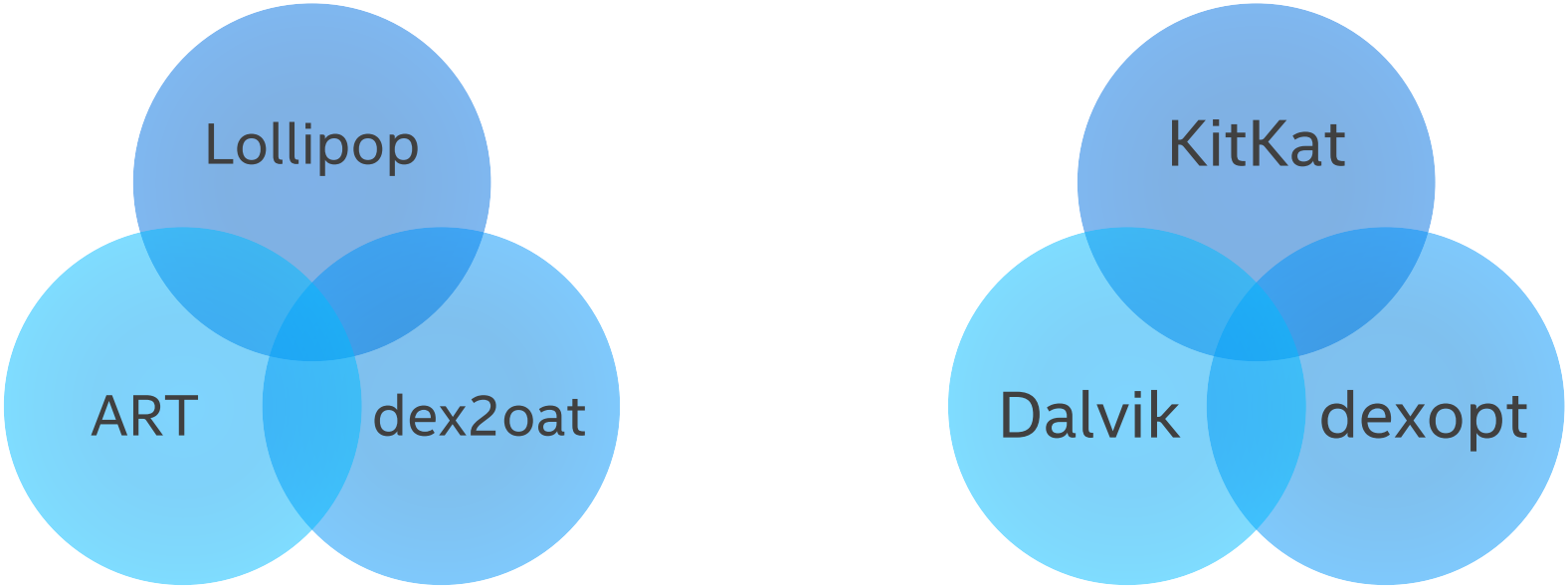
# Application installer – attractive target

Process runs with high system privileges

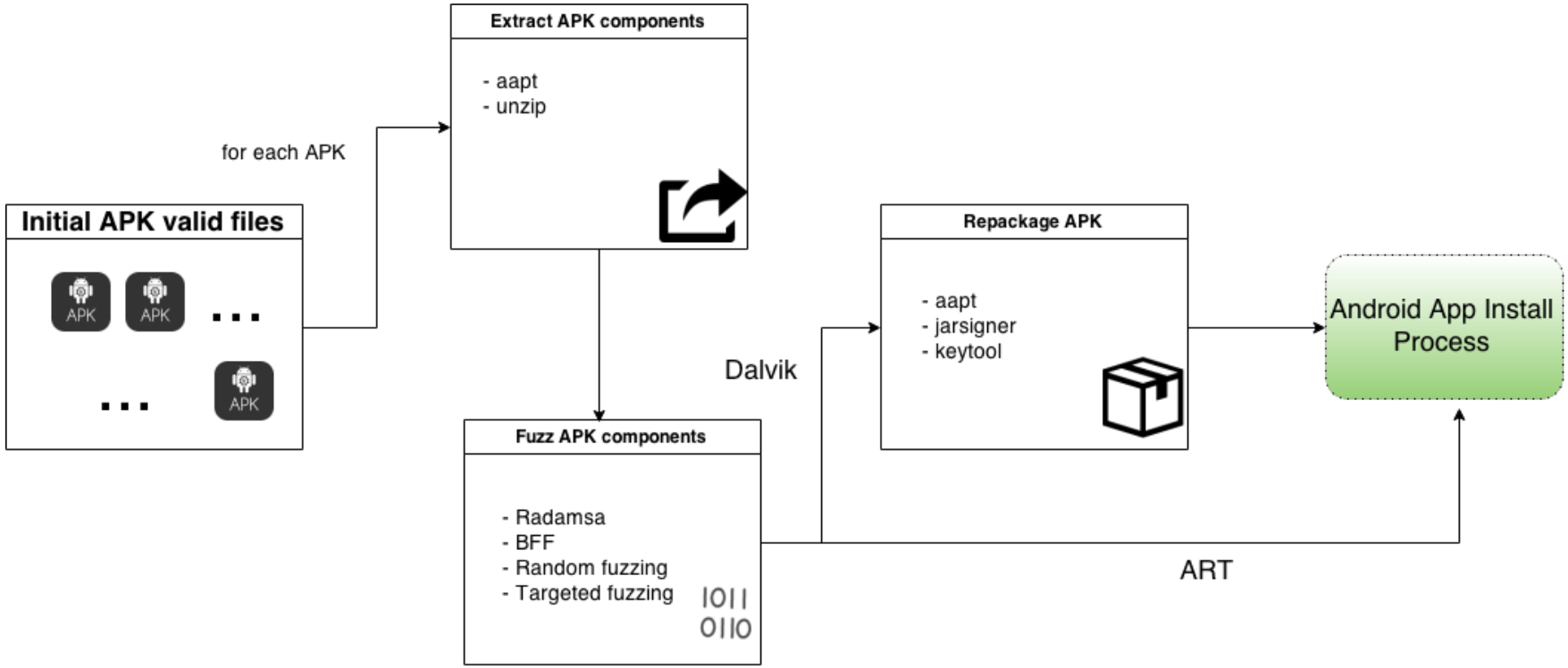
Method for unprivileged users to send input to system components

Check for issues that are not discovered during regular validation

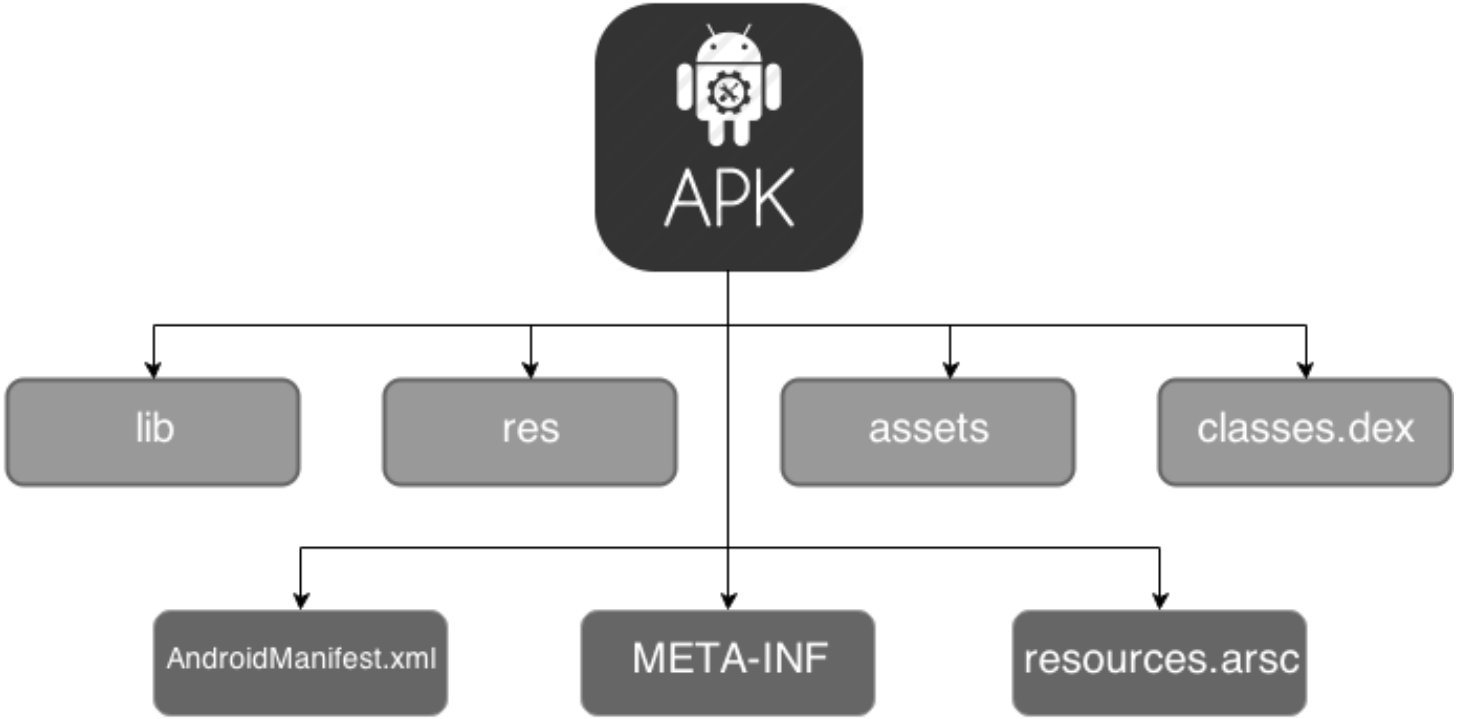
# Install process overview



# Testing process overview



# APK structure



# Approach on Dalvik (dexopt)

- Simulate the regular APK install process
  - Extract classes.dex file from seed APK
    - unzip –d  </local/path/> </apk/path/>
  - Fuzz extracted dex file*
    - <fuzz>–s <seed> classes.dex > fuzzed.dex
  - Remove original .dex file from initial APK
    - aapt r <original_apk> classes.dex
  - Repackage APK with fuzzed dex file
    - aapt a <original_apk> classes.dex

# Approach on Dalvik (dexopt)

- Simulate the regular APK install process
  - Create local keystore
    - keytool -genkey -v -keystore keystore.keystore -alias keystore -keyalg RSA -keysize 2048 -validity 10000
  - Remove META-INF directory from APK
    - zip --delete </apk/path/> META-INF/*
  - Resign the APK using local keystore
    - jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore </keystore/path> </apk/path> <keystore_alias>

# Approach on Dalvik (dexopt)

- Log example

```
06-26 17:43:29.732 F/dexopt (14881): - sp_lib.py - APK_id =
imangi.templerun.apk combination = radamsa -s 2086

06-26 17:43:54.620 F/dexopt (14988): - sp_lib.py - APK_id =
imangi.templerun.apk seed = radamsa -s 5011

06-26 17:44:44.079 F/libc    (15227): Fatal signal 11 (SIGSEGV)
at 0xaa4c04f8 (code=1), thread 15227 (mangi.templerun)

06-26 17:45:09.950 F/dexopt (15338): - sp_lib.py - APK_id =
imangi.templerun.apk seed = radamsa -s 8098
```

# Approach on ART (dex2oat)

- Dex files can be passed directly to dex2oat binary

```
Usage: dex2oat [options]...
--dex-file=<dex-file>: specifies a .dex file to compile.
--zip-fd=<file-descriptor>: specifies a file descriptor of a
zip file containing a classes.dex file to compile.
--zip-location=<zip-location>: specifies a symbolic name for
the file
--oat-file=<file.oat>: specifies the oat output destination via
a filename.
--oat-fd=<number>: specifies the oat output destination via a
file descriptor.
--oat-location=<oat-name>: specifies a symbolic name for the
file corresponding to the file descriptor specified by --oat-
fd.
...
```

# Approach on ART (dex2oat)

- Log example

```
09-29 11:32:20.460 F/dex2oat ( 8041): - sp_libd.py - dex_id =
com.evernote.dex seed = radamsa -s 1012528

09-29 11:32:46.277 F/dex2oat ( 8066): - sp_libd.py - dex_id =
com.evernote.dex seed = radamsa -s 7338683

09-29 11:32:49.121 F/libc    (15227): Fatal signal 11 (SIGSEGV)
at 0xaa4c0302 (code=1), thread 15227 (evernote)

09-29 11:32:57.249 F/dex2oat ( 8079): - sp_libd.py - dex_id =
com.evernote.dex seed = radamsa -s 231131
```

# Install verification process

```
01-03 13:24:13.511 I/dex2oat ( 5671): dex2oat --dex-
file=test1.dex --oat-file=output.oat
01-03 13:24:13.125 W/dex2oat ( 5671): Failed to open .dex from
file 'test1.dex': Bad checksum (790931db, expected 745631bc)

01-03 03:22:23.581 I/dex2oat ( 5671): dex2oat --dex-
file=test2.dex --oat-file=output.oat
01-03 03:22:23.635 W/dex2oat ( 5671): Failed to open .dex from
file 'test2.dex': Bad file size (143221ab, expected 435611cd)

01-03 04:21:13.181 I/dex2oat ( 5671): dex2oat --dex-
file=test3.dex --oat-file=output.oat
01-03 04:21:13.235 W/dex2oat ( 5671): Failed to open .dex from
file 'test3.dex': Invalid header size (7f, expected 70)
```

# Actual fuzzing methods

## Completely random fuzzing

- Applies to dex2oat

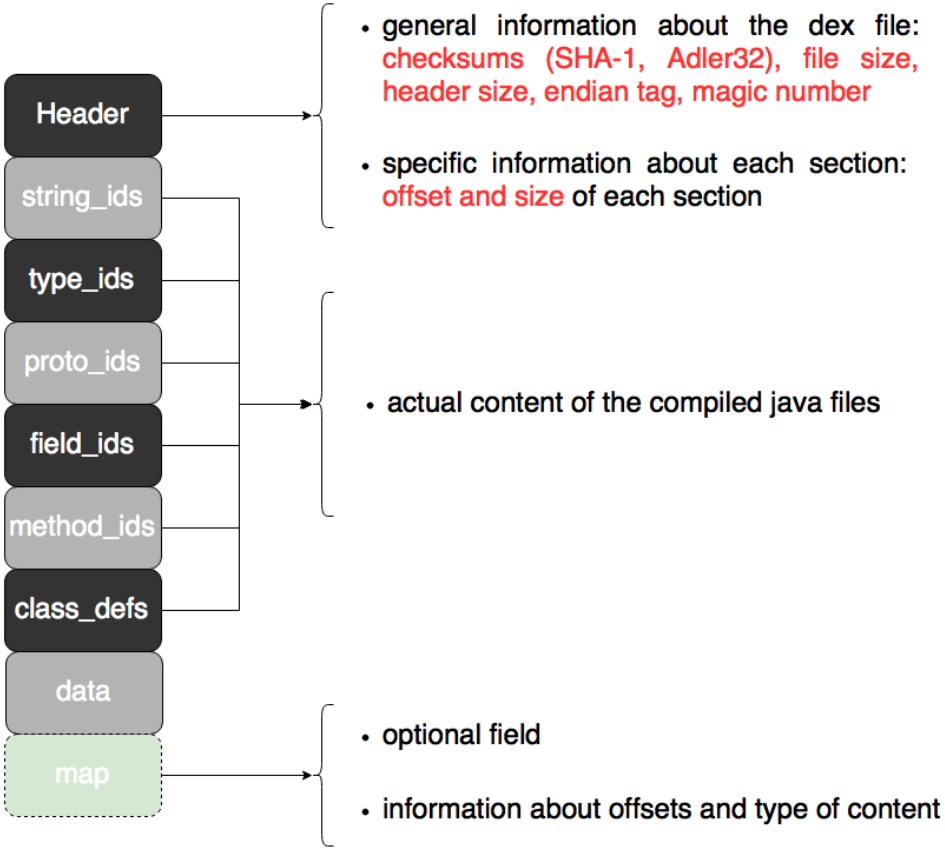
## Partially guided fuzzing

- Applies to dexopt & dex2oat
- Random fuzzing / partial header reconstruction

## Targeted fuzzing

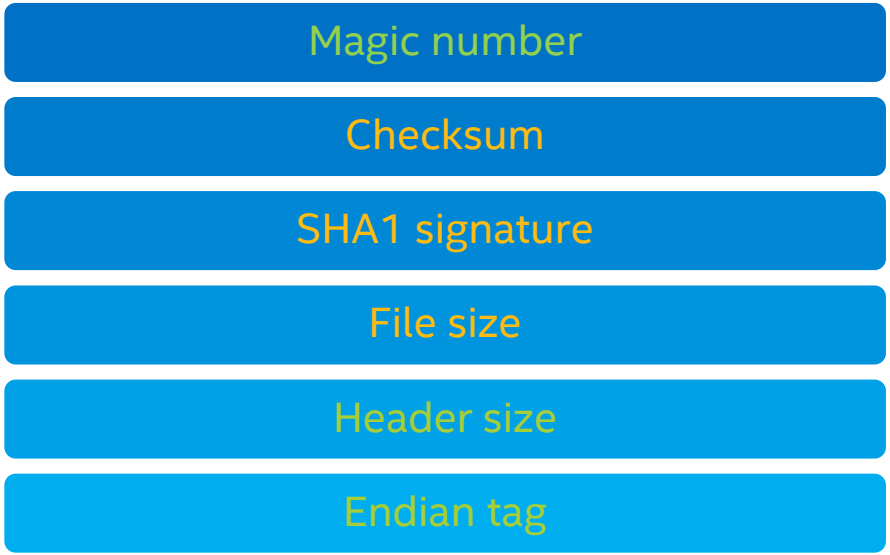
- Applies to dexopt & dex2oat
- Target a section of the DEX file / complete header reconstruction

# DEX file format

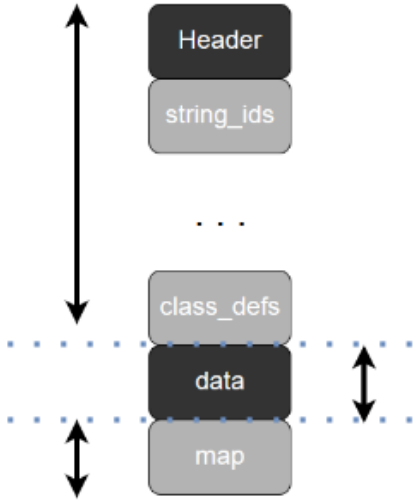


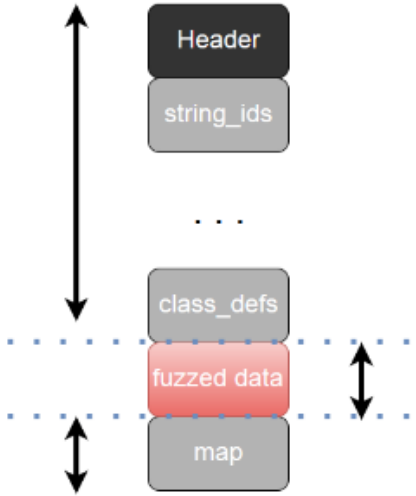
# Partially guided fuzzing

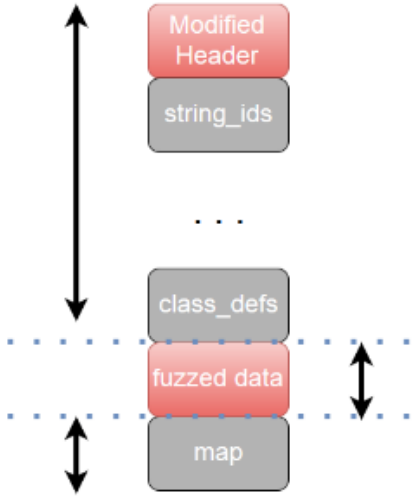
- Randomly modify contents of all sections of the DEX file
- Re-compute and/or rewrite the header fields we have information about



# Targeted fuzzing




- Magic number
- Checksum
- SHA1 signature
- File size
- Header size
- Endian tag

- Map offset
- Data size

# Completely random fuzzing

- Generate random fuzzed DEX files – no changes to the header after fuzzing

```
09-19 11:57:00.346 F/dex2oat_bff(16102): - sp_libd.py - dex_id
= zzuf.16185.sOaX7i.dex

09-19 11:57:01.193 F/dex2oat_bff(16113): - sp_libd.py - dex_id
= zzuf.2554.pfKpqy.dex

09-19 11:57:04.218 F/libc    (16127): Fatal signal 11 (SIGSEGV)
at 0xaa2c14f4 (code=1), thread 16127 (evernote)

09-19 11:57:05.767 F/dex2oat_bff(16136): - sp_libd.py - dex_id
= zzuf.17117.vuTEiB.dex
```

# Results

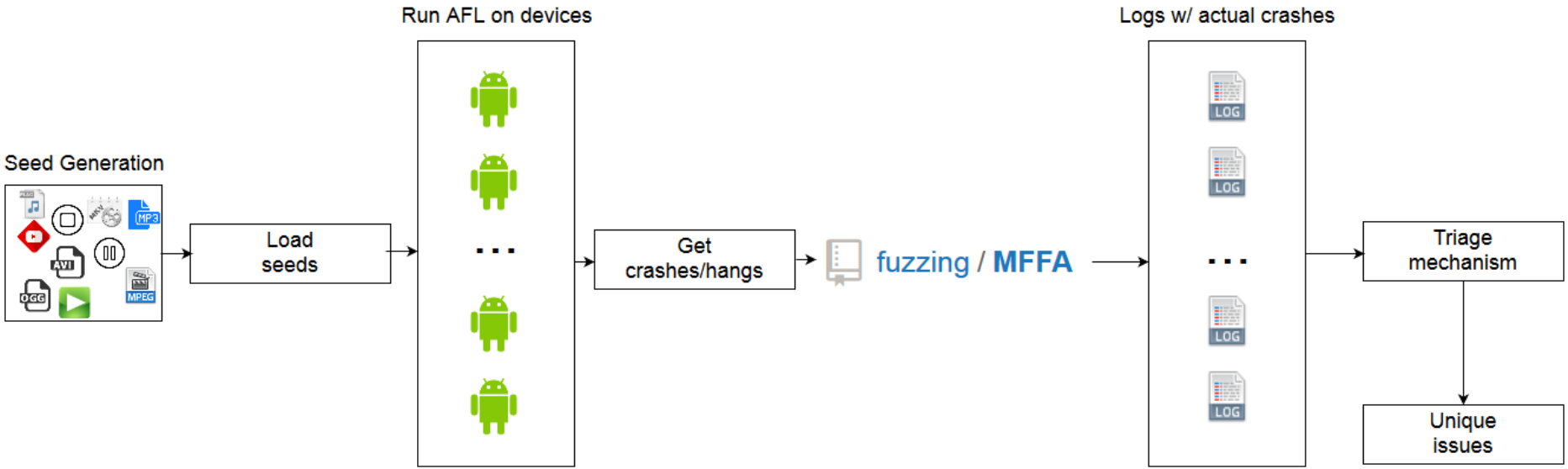
- Number of crashes not as spectacular as in the case of Stagefright
- 1 critical issue affecting dex2oat – CVE-2014-7918
- A number of low priority issues reported and fixed both in KitKat and Lollipop
- Several issues under investigation

# Fuzzing with AFL in Android

# American Fuzzy Lop – overview

- Instrumentation based fuzzing tool developed by Michal Zalewski
- Two fuzzing modes: dumb-mode, instrumented-mode (peruvian rabbit mode)
- Instrumented mode detects changes to program control flow to find new code paths
- Detects both crashes and hangs and sorts out the unique issues
- Android port of the tool developed by Adrian Denkiewicz of Intel (patch available on the mailing list)

# Using AFL for Stagefright fuzzing



# Results

- 1 critical issue discovered using this approach:
  - heap corruption that can lead to arbitrary code execution in the mediaserver process (CVE-2015-3832)

- Multiple low priority issues reported to and fixed by Google (null-pointer dereferences, integer division by zero issues)

# Q&A

ioan-alexandru.blanda@intel.com