Fuzzing Android: A Recipe for Uncovering Vulnerabilities Inside System Components in Android

Alexandru Blanda
Intel OTC Security SQE
A fuzzing approach
• Set of basic methods and concepts for fuzzing in Android

Real-life fuzzing campaigns
• Fuzzing Stagefright
• Fuzzing the Android installer

Alternatives
• Fuzzing with AFL in Android
A fuzzing approach in Android

1. Data generation
2. Execute/run test
3. Log process
4. Triage mechanism
5. Analyze & debug crashes
- Mutational vs. generational fuzzing

- Tools
  - Basic Fuzzing Framework (BFF)
  - FuzzBox
  - Radamsa
  - American Fuzzy Lop (AFL)*

- Seed gathering
  - Python mass downloader using Google and Bing search engines
    - `inurl:htm` `inurl:html` `intitle:“index of”` `.mp3` + `wget`
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>
01-16 17:46:24.328 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>
01-16 17:46:55.933 F/<Component> (PID):  <test_case_index> *** <reproducibility_info>
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
Abort message: 'invalid address or address of corrupt block 0x8004d748 passed to dlfree'

```
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 0000d630 /system/lib/libc.so (free+32)
#02 pc 000dcf1c /system/lib/libstagefright.so
(android::MediaBuffer::~MediaBuffer())+108)
#03 pc 000dd6eb /system/lib/libstagefright.so
(android::MediaBuffer::release())+267)
#04 pc 000ddf7b /system/lib/libstagefright.so
(android::MediaBufferGroup::~MediaBufferGroup())+187)
```
1. Parse generated logs
   - Identify input that causes crashes

2. Retest crashing input

3. For each identified test case
   - 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
pid: 3438, tid: 3438, name: stagefright >>> stagefright <<<
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr deadbaad
Abort message: 'invalid address or address of corrupt block 0x8004d748 passed to dlfree'

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 0000d630 /system/lib/libc.so (free+32)
  #02 pc 000dcf1c /system/lib/libstagefright.so
(android::MediaBuffer::~MediaBuffer()+108)
  #03 pc 000dd6eb /system/lib/libstagefright.so
(android::MediaBuffer::release()+267)
  #04 pc 000ddf7b /system/lib/libstagefright.so
(android::MediaBufferGroup::~MediaBufferGroup()+187)
• 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 detected
 *  bit 4 == 1: fault was an instruction fetch
 */
• **gdbserver (on device)**

```bash
$ gdbserver :5039 --attach <process_pid>
  OR
$ gdbserver :5039 /path/to/executable <options> (ex: gdbserver :5039 /system/bin/stagefright -a file.mp3)
```

• **gdb (on local machine)**

```bash
$ adb forward tcp:5039 tcp:5039
$ gdb
  (gdb) target remote :5039 (from the gdb shell)
  (gdb) continue (to resume process execution)
```

• **Load symbols for shared libraries**

```bash
(gdb) set solib-absolute-prefixdb
/path/to/tree/out/target/product/<product_id>/symbols/
(gdb) set solib-search-path
/path/to/tree/out/target/product/<product_id>/symbols/system/lib/
```
• 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/lib/libstagefright.so
  000dcf1c
Fuzzing the Stagefright media framework

black hat

EUROPE 2015
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
• Create corrupt but structurally valid media files
• Direct them to the appropriate decoders
• Monitor the system for potential issues
• Pass the issues through a triage mechanism
• frameworks/av/cmds/stagefright

root@android:/ # stagefright -h
usage: stagefright
-h(elp)
-a(udio)
-n repetitions
-l(ist) components
-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)
-x display a histogram of decoding times/fps (video only)
-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)
- Media files are corrupted on the local machine using the Basic Fuzzing Framework (BFF) tool

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
• Initial fuzzing campaigns started in March 2014; first issues reported to Google

• Initial 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 along with AFL
Fuzzing the application install process

black hat
EUROPE 2015
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
• Approach on Dalvik: dexopt cannot be called as a standalone binary from the device shell

• Simulate the regular APK install process:

  ➢ Extract classes.dex file from seed APK
    o unzip -d </local/path/> </apk/path/>
  ➢ Fuzz extracted dex file*
    o <fuzz>-s <seed> classes.dex > fuzzed.dex
  ➢ Remove original .dex file from initial APK
    o aapt r <original_apk> classes.dex
  ➢ Repackage APK with fuzzed APK
    o aapt a <original_apk> classes.dex
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 */META-INF/*`

- **Resign the APK using local keystore**
  - `jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore */keystore/path> <keystore_alias>`
Log example:

```
06-26 17:43:05.568 F/dexopt (14769): - sp_lib.py - APK_id = imangi.templerun.apk
seed = radamsa -s 1927
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 17:44:19.763 F/dexopt (15105): - sp_lib.py - APK_id = imangi.templerun.apk
seed = radamsa -s 1543
06-26 17:44:43.524 F/dexopt (15215): - sp_lib.py - APK_id = imangi.templerun.apk
seed = radamsa -s 9090
06-26 17:44:44.079 F/libc (15227): Fatal signal 11 (SIGSEGV) at 0xaaa4c04f8 (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
06-26 17:45:33.771 F/dexopt (15451): - sp_lib.py - APK_id = imangi.templerun.apk
seed = radamsa -s 1069
06-26 17:45:59.802 F/dexopt (15570): - sp_lib.py - APK_id = imangi.templerun.apk
seed = radamsa -s 8925
```
• Approach on ART: dex2oat can be used directly from the command line for our fuzzing purposes

Usage: dex2oat [options]...
   -j<number>: specifies the number of threads used for compilation.
   --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.
...
...
...
### Log example:

<table>
<thead>
<tr>
<th>Time</th>
<th>Process</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:32:20.460</td>
<td>F/dex2oat (8041)</td>
<td>- sp_libd.py - dex_id = com.evernote.apk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>seed = radamsa -s 1012528</td>
</tr>
<tr>
<td>11:32:33.405</td>
<td>F/dex2oat (8054)</td>
<td>- sp_libd.py - dex_id = com.evernote.apk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>seed = radamsa -s 6186726</td>
</tr>
<tr>
<td>11:32:46.277</td>
<td>F/dex2oat (8066)</td>
<td>- sp_libd.py - dex_id = com.evernote.apk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>seed = radamsa -s 7338683</td>
</tr>
<tr>
<td>11:32:49.121</td>
<td>F/libc (15227)</td>
<td><strong>Fatal signal 11 (SIGSEGV) at 0xaaa4c0302 (code=1), thread 15227</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(evernote)</td>
</tr>
<tr>
<td>11:32:57.249</td>
<td>F/dex2oat (8079)</td>
<td>- sp_libd.py - dex_id = com.evernote.apk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>seed = radamsa -s 231131</td>
</tr>
<tr>
<td>11:33:08.528</td>
<td>F/dex2oat (8093)</td>
<td>- sp_libd.py - dex_id = com.evernote.apk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>seed = radamsa -s 4456070</td>
</tr>
</tbody>
</table>
• 3 alternatives:

1. Completely random fuzzing (dex2oat)
2. Random fuzzing and partial header reconstruction (applies to dex2oat & dexopt)
3. Targeted fuzzing and complete header reconstruction (dex2oat & dexopt)
01-03 13:24:13.125 W/dex2oat (5671): Failed to open .dex from file 'test7.dex': verify
dex file 'test7.dex': **Bad checksum** (790931db, expected 745631bc)
01-03 13:24:13.115 E/dex2oat (5671): Failed to open some dex files: 1
01-03 13:24:13.447 I/dex2oat (5671): dex2oat took 255.693ms (threads: 4)

01-03 03:22:23.635 W/dex2oat (5671): Failed to open .dex from file 'test7.dex': verify
dex file 'test7.dex': **Bad file size** (143221ab, expected 435611cd)
01-03 03:22:23.635 E/dex2oat (5671): Failed to open some dex files: 1
01-03 03:22:23.837 I/dex2oat (5671): dex2oat took 255.693ms (threads: 4)

01-03 04:21:13.235 W/dex2oat (5671): Failed to open .dex from file 'test7.dex': verify
dex file 'test7.dex': **Invalid header size** (7f, expected 70)
01-03 04:21:13.641 E/dex2oat (5671): Failed to open some dex files: 1
01-03 04:21:13.857 I/dex2oat (5671): dex2oat took 255.693ms (threads: 4)
DEX file format

- header
- string_ids
- type_ids
- proto_ids
- field_ids
- method_ids
- class_defs
- data
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct dex_magic magic</td>
<td>dex 035</td>
<td>0h</td>
<td>Magic value</td>
</tr>
<tr>
<td>uint checksum</td>
<td>B3D20217h</td>
<td>8h</td>
<td>Alder32 checksum of rest of file</td>
</tr>
<tr>
<td>SHA1 signature[20]</td>
<td>6DB8EDA774Ch</td>
<td>14h</td>
<td>SHA-1 signature of rest of file</td>
</tr>
<tr>
<td>uint file_size</td>
<td>1430508</td>
<td>20h</td>
<td>File size in bytes</td>
</tr>
<tr>
<td>uint header_size</td>
<td>112</td>
<td>24h</td>
<td>Header size in bytes</td>
</tr>
<tr>
<td>uint endian_tag</td>
<td>12345678h</td>
<td>28h</td>
<td>Endianness tag</td>
</tr>
<tr>
<td>uint link_size</td>
<td>0</td>
<td>2Ch</td>
<td>Size of link section</td>
</tr>
<tr>
<td>uint link_off</td>
<td>0</td>
<td>30h</td>
<td>File offset of link section</td>
</tr>
<tr>
<td>uint map_off</td>
<td>1430336</td>
<td>34h</td>
<td>File offset of map list</td>
</tr>
<tr>
<td>uint string_ids_size</td>
<td>11029</td>
<td>38h</td>
<td>Count of strings in the string ID list</td>
</tr>
<tr>
<td>uint string_ids_off</td>
<td>112</td>
<td>3Ch</td>
<td>File offset of string ID list</td>
</tr>
<tr>
<td>uint type_ids_size</td>
<td>2068</td>
<td>40h</td>
<td>Count of types in the type ID list</td>
</tr>
<tr>
<td>uint type_ids_off</td>
<td>44228</td>
<td>44h</td>
<td>File offset of type ID list</td>
</tr>
<tr>
<td>uint proto_ids_size</td>
<td>2592</td>
<td>48h</td>
<td>Count of items in the method prototype ID list</td>
</tr>
<tr>
<td>uint proto_ids_off</td>
<td>52500</td>
<td>4Ch</td>
<td>File offset of method prototype ID list</td>
</tr>
<tr>
<td>uint field_ids_size</td>
<td>5335</td>
<td>50h</td>
<td>Count of items in the field ID list</td>
</tr>
<tr>
<td>uint field_ids_off</td>
<td>83604</td>
<td>54h</td>
<td>File offset of field ID list</td>
</tr>
<tr>
<td>uint method_ids_size</td>
<td>12925</td>
<td>58h</td>
<td>Count of items in the method ID list</td>
</tr>
<tr>
<td>uint method_ids_off</td>
<td>126284</td>
<td>5Ch</td>
<td>File offset of method ID list</td>
</tr>
<tr>
<td>uint class_def_size</td>
<td>1427</td>
<td>60h</td>
<td>Count of items in the class definitions list</td>
</tr>
<tr>
<td>uint class_def_off</td>
<td>229684</td>
<td>64h</td>
<td>File offset of class definitions list</td>
</tr>
<tr>
<td>uint data_size</td>
<td>1155160</td>
<td>68h</td>
<td>Size of data section in bytes</td>
</tr>
<tr>
<td>uint data_off</td>
<td>275348</td>
<td>6Ch</td>
<td>File offset of data section</td>
</tr>
</tbody>
</table>
Random fuzzing and partial header reconstruction

- Alter randomly the contents of all sections of the dex file

- Recompute and/or rewrite the header fields we have information about

<table>
<thead>
<tr>
<th>Field</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magic number</td>
<td>(constant value)</td>
</tr>
<tr>
<td>Checksum</td>
<td>(needs to be computed)</td>
</tr>
<tr>
<td>SHA1 signature</td>
<td>(needs to be computed)</td>
</tr>
<tr>
<td>File size</td>
<td>(needs to be computed)</td>
</tr>
<tr>
<td>Header size</td>
<td>(constant value)</td>
</tr>
<tr>
<td>Endian tag</td>
<td>(constant value)</td>
</tr>
</tbody>
</table>
Targeted fuzzing and complete header reconstruction

- Split the initial dex in 3 parts: data section, map section and the unmodified rest of the file section
- Fuzz only the data chunk as a separate file using Radamsa
- Glue all the chunks back together and rewrite the header

<table>
<thead>
<tr>
<th>Magic number (constant value)</th>
<th>Map offset (needs to be computed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checksum (needs to be computed)</td>
<td>SHA1 signature (needs to be computed)</td>
</tr>
<tr>
<td>File size (needs to be computed)</td>
<td>Data size (needs to be computed)</td>
</tr>
<tr>
<td>Header size (constant value)</td>
<td>endian tag (constant value)</td>
</tr>
</tbody>
</table>
• Generate random fuzzed dex files using BFF, starting from an initial set of seed dex files
• No changes made to the dex file header
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

black hat
EUROPE 2015
• 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
1. Check device prerequisites
   1) Root
   2) Remount
   3) Push afl target binary
   4) Load initial seeds
   5) Set scalling governor
2. Eliminate crashing test cases from initial seeds on each device
   1) Run AFL in a loop with timeout
   2) Identify crashing test case and delete it from input folder
   3) Restart AFL with timeout -> if crash occurs goto 2) else goto 4)
   4) No crash occurred after the timeout -> AFL successfully started -> kill the process
3. Restart the AFL process with clean input directory and redirect output to /dev/null
Seed Generation

Load seeds

Run AFL on devices

Get crashes/hangs

fuzzing / MFFA

Logs w/ actual crashes

Triage mechanism

Unique issues
• 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)
Acknowledgements

Adrian Denkiewicz – adrian.o.denkiewicz@intel.com
Costel Maxim – costel.maxim@intel.com
Ionut Gavaz – ionut.a.gavaz@intel.com
Intel OTC Security Romania Team

black hat
EUROPE 2015