Finding media bugs in Android using file format fuzzing

Costel Maxim

Intel OTC Security

Agenda

- File format fuzzing
- Generate high-quality corpus of files
- Fuzzing the Stagefright Framework
- Logging & Triage Mechanisms
- Results

Introduction to Fuzzing

- Form of black-box testing

- Involves sending corrupt input to a software system and monitoring for crashes
- Purpose: find security-related problems or any other critical defects that could lead to an undesirable behavior of the system

File format fuzzing

- Possible targets:



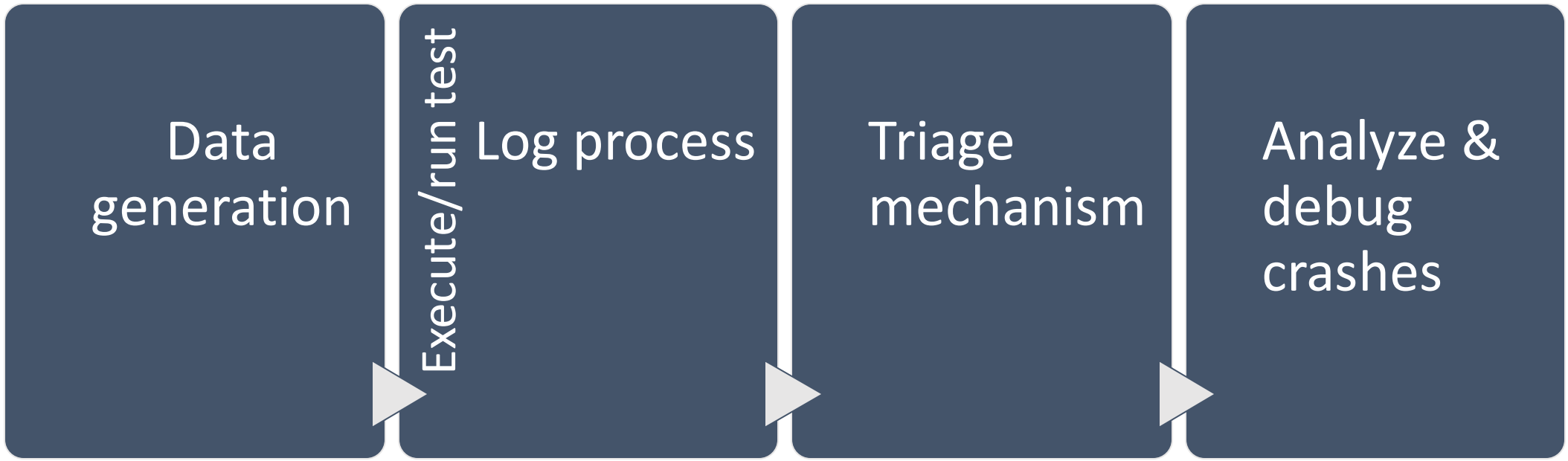







- Media Players
- Document Viewers
- Web Browsers
- Antivirus products
- Binary (ELF)

Walkthrough



Audio and video 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

Fuzzing Media Content in Android

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

Stagefright CLI

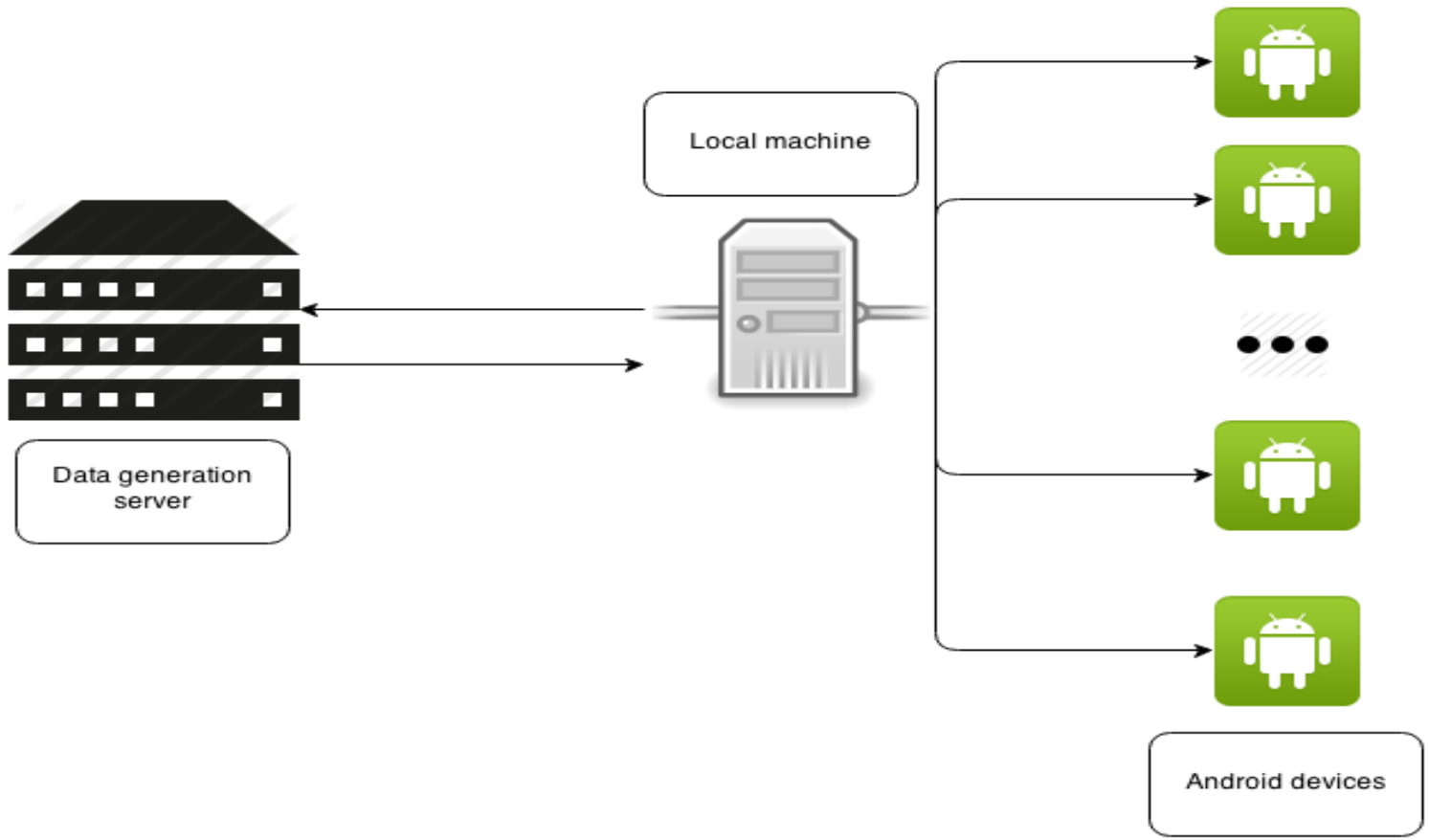
```
root@android:/ # stagefright -h
usage: stagefright
-h(elp)
-a(udio)
-n repetitions
-l(ist) components
-m max-number-of-frames-to-decode in each pass
-p(rofiles) dump decoder profiles supported
-t(humbnail) extract video thumbnail or album art
-s(oftware) prefer software codec
-r(hardware) force to 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)
```

Data Generation

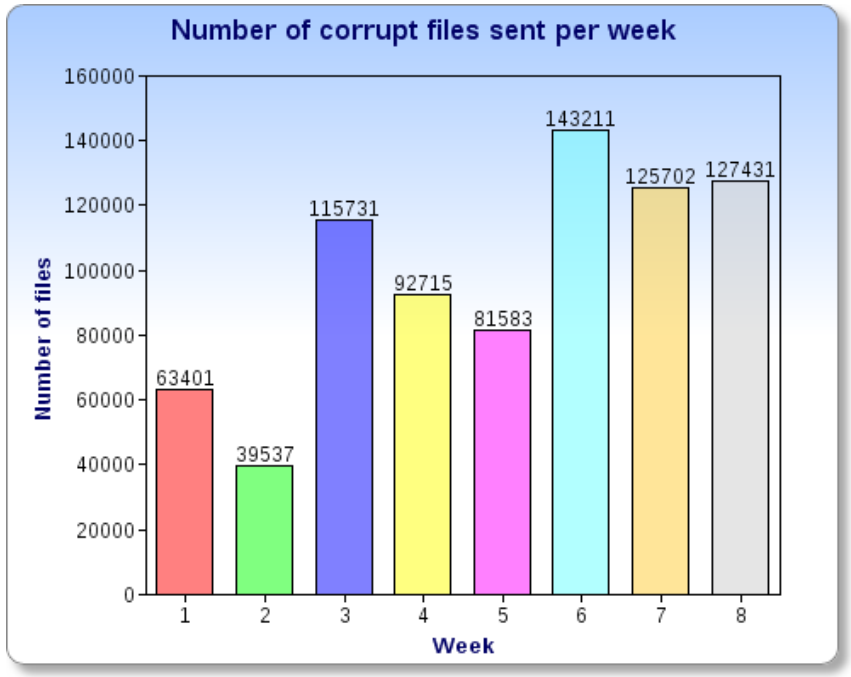
- Mutational vs. generational fuzzing

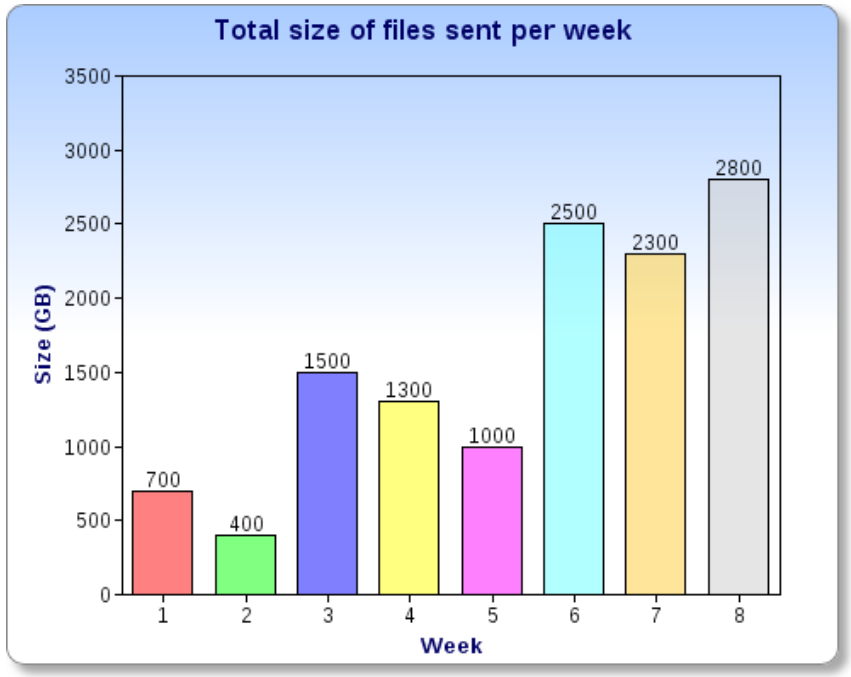
- Tools
  - Zzuf
  - Basic Fuzzing Framework (BFF)
  - American Fuzzy Lop (AFL)

Fuzzing infrastructure



Initial corpus of inputs for the fuzzer





Total: 801311 files	Total: 11.5 TB

Logging and Triage Mechanisms

- Log every test case with fatal priority

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

- Log template

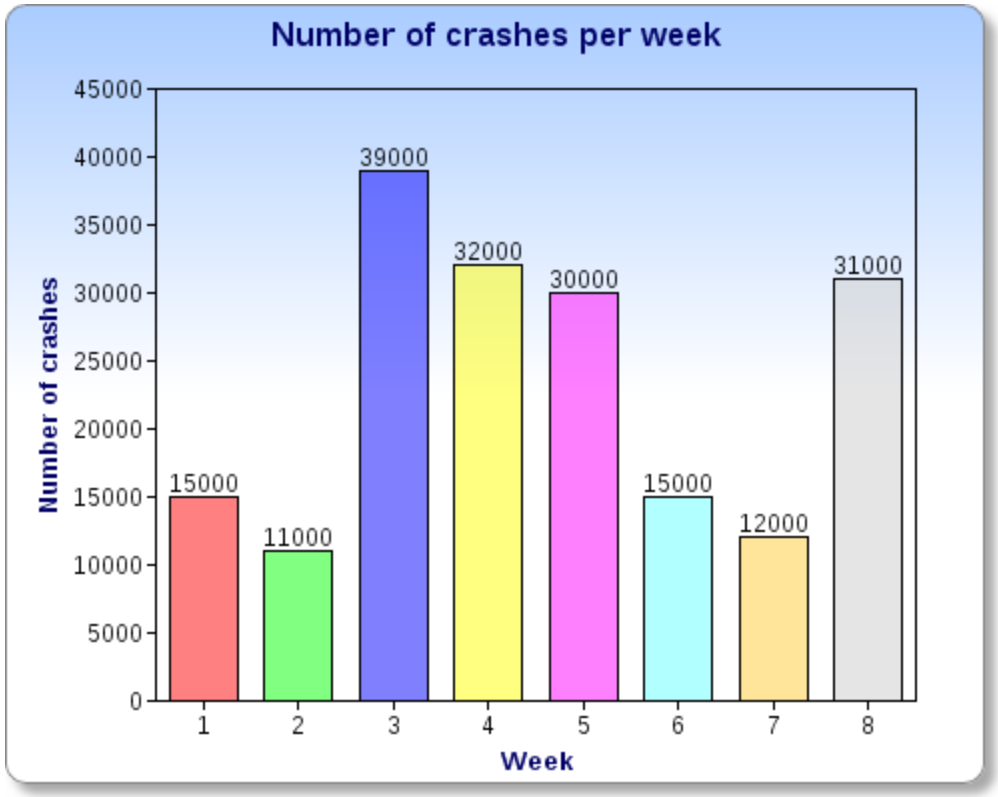
```
$ adb shell logcat -v time *:F

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>
```

If everything goes “well”

- Crashes (SIGSEGV, SIGFPE, SIGABRT, SIGILL)
- Process hangs (synchronization issues, memory leaks, infinite loops)
- Denial of Service situations (device reboots, application crashes, stack/heap exhaustion)
- Buffer overflows, null-pointer dereference, integer overflows

Crashes



Total: 185000 crashes

Triage mechanism

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

Triage phase - implementation

- 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)
```

Bugs overflow

- Initial results were extremely surprising: thousands of crashes per week
- First severe issues were published in September 2014 Android security bulletin:
  - Integer overflows in libstagefright: CVE-2014-7915, CVE-2014-7916, CVE-2014-7917
  - Arbitrary code execution in the mediaserver process: CVE-2015-3832
- The tool was open-sourced in February 2015: https://github.com/fuzzing/MFFA
- Currently used as a complementary solution along with AFL

https://github.com/fuzzing/MFFA

costel.maxim@intel.com