Bug hunting in file format on Android: audit && fuzz

2017补天成都沙龙

关于我

- 韩子诺@奇虎360成都安全应急响应中心
- 研究方向：Android漏洞挖掘与利用
- ID && 微博：ele7enxxh
- 博客：ele7enxxh.com

议程

- 研究背（动）景（力）
- 研究内容
  - 代码审计的姿势
  - Fuzz实战
- 总结

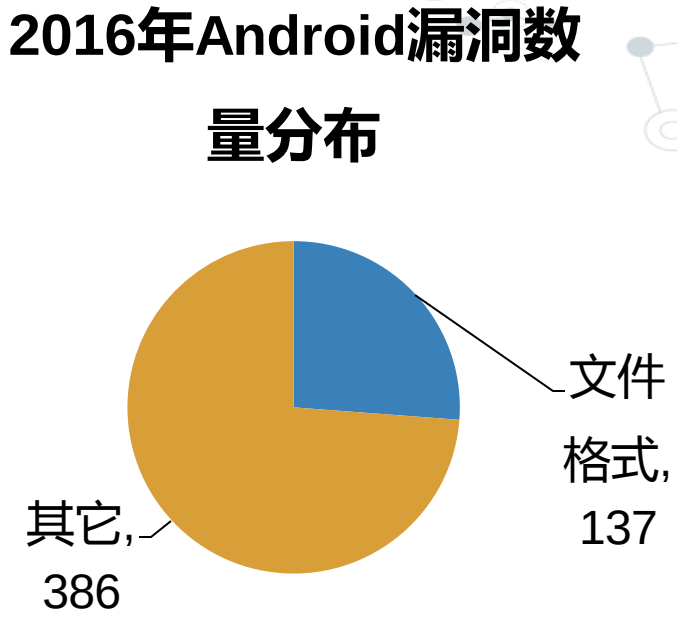
1.

研究背（动）景（力）

洞多，钱多，速来！

漏洞多

文件格式漏洞占比超过25%，至今仍未有减少的趋势！

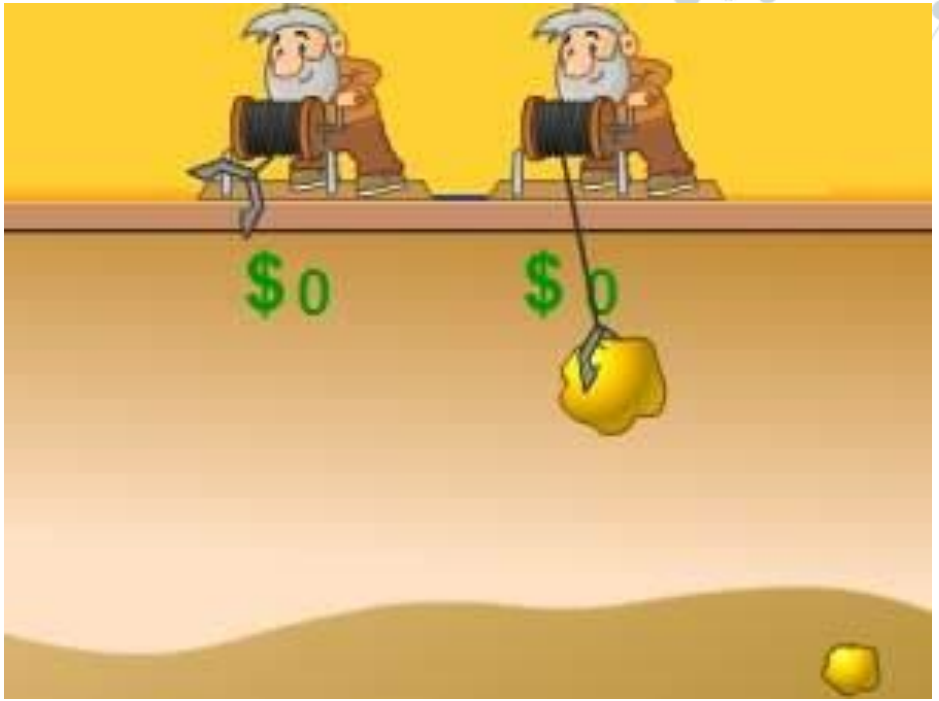


数据来源：CVE Details

奖金高

- 中危：1000$
- 高危：2000$
- 严重：4000$

以及不菲的奖金



入门快

门栏低，适合萌新快速入门

研究对象

- 软件：AOSP中用C/C++编写的用于文件格式解析的库
- 输入：各种不同的文件格式的文件
- 期望：内存破坏漏洞（堆溢出，栈溢出，UAF，信息泄漏等）

2.
代码审计的姿势

不一定挖的到

什么是代码审计

代码审计是一种通过人工阅读代码以发现程序安全漏洞的技术。

https://en.wikipedia.org/wiki/Code_audit

审计之前

- 对文件格式有一定的了解
- 熟悉审计对象所用的编程语言，本议题为C/C++
- 研究以往的安全补丁
- 熟悉所有可能发生的漏洞类型，并熟悉导致漏洞的潜在原因

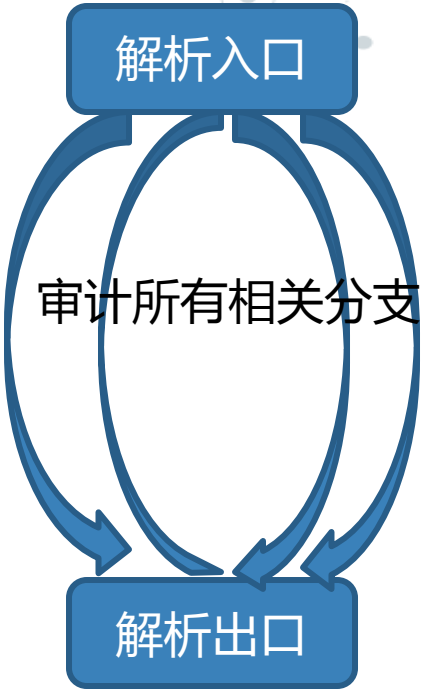
保持耐心与平静

审计工具

- vim/vi，emacs
- sublime，notepad++
- 任何其他你顺手的IDE

姿势1：从入口到出口

1. 确定入口
2. 跟踪整个解析流程
3. 审计所有与文件数据相关的代码分支，以发现可能的安全漏洞
4. 确定出口



确定入口

- 官方文档
  - README.md
  - 测试代码
- 函数名
  - libc API：fopen，open，mmap
  - Android API：FileSource
  - 常见命名：parse，decode

确定出口

- 官方文档
  - README.md
  - 测试代码
- 函数名
  - libc API：fclose，close，munmap，exit
- 调用parse，decode之后

实战：CVE-2016-6762

- 漏洞介绍
  - libziparchive中的整数溢出导致的内存破坏或权限提升
  - 由ele7enxxh发现，于2016年提交Google
  - 影响Android5.0-7.0版本
  - 可被adb，dexdump以及其他使用该库的第三方程序触发

实战：CVE-2016-6762

- 确定入口

```
grep -rn -include=*.cc -E “open|fopen|parse” libziparchive/
```

```
int32_t OpenArchive(const char* fileName, ZipArchiveHandle* handle) {
  const int fd = open(fileName, O_RDONLY | O_BINARY, 0);
  ZipArchive* archive = new ZipArchive(fd, true);
  *handle = archive;
  if (fd < 0) {
    ALOGW("Unable to open '%s': %s", fileName,strerror(errno));
    return kIoError;
  }
  return OpenArchiveInternal(archive, fileName);
}
```

OpenArchive -> OpenArchiveInternal -> MapCentralDirectory -> MapCentralDirectory0（最终的解析函数）

实战：CVE-2016-6762

审计解析流程

```
...
const EocdRecord* eocd = reinterpret_cast<const
EocdRecord*>(scan_buffer + i);
...
if (eocd->cd_start_offset + eocd->cd_size >
eocd_offset) {
    ...
}
...
android::FileMap* map = MapFileSegment(fd,
static_cast<off64_t>(eocd->cd_start_offset),
static_cast<size_t>(eocd->cd_size), true,
debug_file_name);
...
```

```
struct EocdRecord {
  ...
  uint32_t cd_size;
  uint32_t cd_start_offset;
  ...
} __attribute__((packed));
```

实战：CVE-2016-6762

```
...
// eocd指针指向文件内容，可控
const EocdRecord* eocd = reinterpret_cast<const EocdRecord*>(scan_buffer + i);
...
// cd_start_offset和cd_size均为无符号32位整型，且值可控，其和可能溢出（大于UINT_MAX ），即可绕过检测
if (eocd->cd_start_offset + eocd->cd_size > eocd_offset) {
    ...
}
...
// MapFileSegment内部调用mmap函数，使用了被控制的offset和size，也就是说映射的数据是非法的。最终会在ParseZipArchive函数中触发bug
android::FileMap* map = MapFileSegment(fd, static_cast<off64_t>(eocd-
>cd_start_offset), static_cast<size_t>(eocd->cd_size), true, debug_file_name);
...
```

姿势2：跟踪结构体

大多数情况下，软件在解析文件的时候，会将文件数据存放在结构体变量中

```
// libziparchive
struct EocdRecord {
  static const uint32_t kSignature =
0x06054b50;
  uint32_t eocd_signature;
  uint16_t disk_num;
  uint16_t cd_start_disk;
  uint16_t num_records_on_disk;
  …
} __attribute__((packed));
```

```
// libzipfile
typedef struct Zipentry {
    unsigned long fileNameLength;
    const unsigned char* fileName;
    unsigned short compressionMethod;
    unsigned int uncompressedSize;
    unsigned int compressedSize;
    const unsigned char* data;
    struct Zipentry* next;
} Zipentry;
```

专注结构体的变化

解析文件的过程

≈

对相关结构体变量的操作（赋值和读取）

步骤

1. 确定待审计的结构体
2. 掌握结构体的每个变量的含义和类型
3. 跟踪结构体及其成员变量的变化
   - 在哪里初始化？
   - 在哪里赋值，是否可控？
   - 在哪里使用？是否有安全漏洞？

实战：CVE-2016-6700

- 漏洞介绍
  - libzipfile中的缓存区溢出导致的内存破坏或权限提升
  - 由ele7enxxh发现，于2016年提交Google
  - 影响Android4.4.4、5.0.2、5.1.1版本
  - 可通过fastboot update xxx.zip方式触发

实战：CVE-2016-6700

- 确定相关结构体

```
typedef struct Zipentry {
    unsigned long fileNameLength;
    const unsigned char* fileName;
    unsigned short compressionMethod;
    unsigned int uncompressedSize;
    unsigned int compressedSize;
    const unsigned char* data;
    struct Zipentry* next;
} Zipentry;
```

```
typedef struct Zipfile {
  const unsigned char *buf;
  ssize_t bufsize;
  unsigned short  disknum;
  unsigned short  diskWithCentralDir;
  unsigned short  entryCount;
  unsigned short  totalEntryCount;
  unsigned int    centralDirSize;
  unsigned int    centralDirOffest;
  unsigned short  commentLen;
  const unsigned char*  comment;
  Zipentry* entries;
} Zipfile;
```

实战：CVE-2016-6700

- 定位所有对结构体变量的引用

```
$ grep -rn --include=*.c "uncompressedSize" libzipfile/

$ libzipfile/zipfile.c:59: return ((ZipEntry*)entry)->uncompressedSize;
$ libzipfile/zipfile.c:122: memcpy(buf, entry->data, entry->uncompressedSize);
$ libzipfile/centraldir.c:108: entry->uncompressedSize = read_le_int(&p[0x18]);
```

实战：CVE-2016-6700

- 在哪里赋值？是否可控？

```
static int read_central_directory_entry(Zipfile* file, Zipentry* entry,
const unsigned char** buf, ssize_t* len) {
  const unsigned char* p;
  ...
  // 指针p指向加载进内存的文件数据，可控
  p = *buf;
  ...
  // entry->compressionMethod为unsigned short类型，值可控
  entry->compressionMethod = read_le_short(&p[0x0a]);
  ...
  // entry->uncompressedSize为unsigned int类型，值可控，且无任何安全校验
  entry->uncompressedSize = read_le_int(&p[0x18]);
  ...
}
```

实战：CVE-2016-6700

- 在哪里使用？

```
int decompress_zipentry(zipentry_t e, void* buf, int bufsize) {
  Zipentry* entry = (Zipentry*)e;
  // 令entry->compressionMethod的值为STORED
  switch (entry->compressionMethod) {
    case STORED:
      // 令entry->uncompressedSize的值大于entry->data的实际大小，即可导致缓存区
      溢出
      memcpy(buf, entry->data, entry->uncompressedSize);
...
```

姿势3：跟踪API

- 内存分配器
- libc函数
- 目标软件API

内存分配器

- malloc和free，new和delete
  - 堆溢出，分配释放不匹配，空指针，oom，double free，use after free

```
// alloc-dealloc-mismatch
int *a = new int[2];
// delete[] a;
delete a;

// if size == small_num -> oom
// if size == big_num -> a == null ->
null point
// else -> heap overflow
while (1) {
  char *a = malloc(size);
  a[size] = 'a';
}
```

```
char *a = malloc(10);
free(a);
if (a) {
  // use after free
  printf("%s\n", a);
  // double free
  free(a);
}
```

libc API

- read，write
  - 实际获得的数据大小与期望的数据大小不总是一致的

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t nbyte);
```

- memcpy，strcmp，strcpy等不安全API
  - src数据大小可能大于dst空间大小

目标软件API

- 通过已有文档以及函数名，参数，在头文件中找到感兴趣的接口函数
- 筛选出可能有安全问题的API
- 审计每处调用该API的代码

3.
Fuzz实战

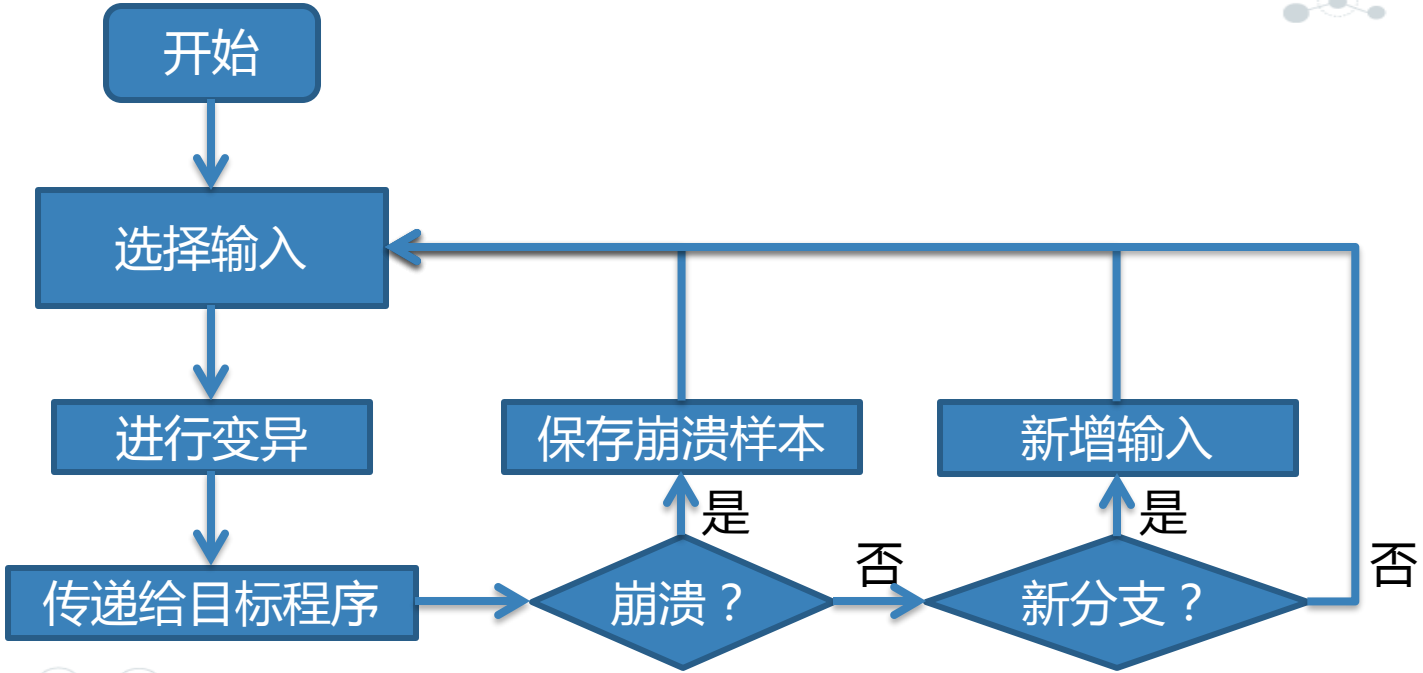
挖到了也可能被撞

什么是Fuzz

模糊测试是一种通过自动或半自动的生成随机数据输入到一个程序中，并监视程序异常，以发现可能的安全漏洞的技术。

http://en.wikipedia.org/wiki/Fuzz_testing

Fuzz流程



看起来简单，如何做？

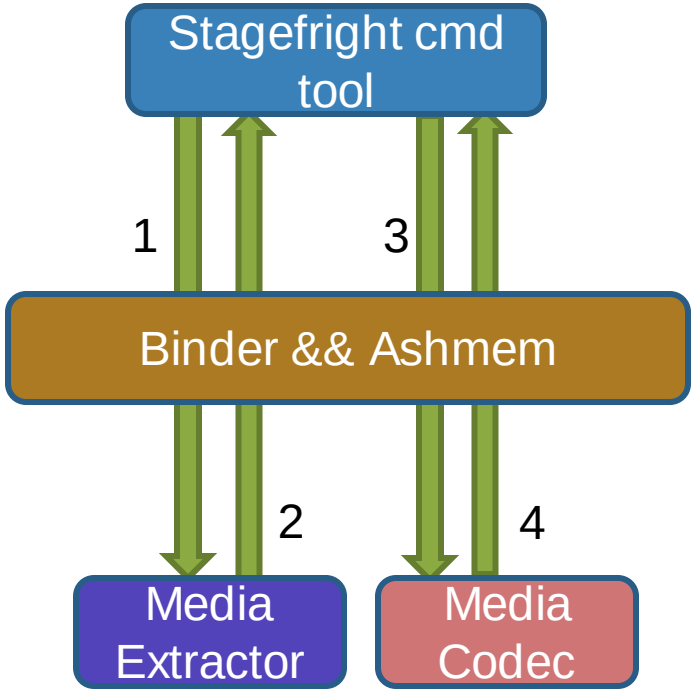
Fuzzing Stagefright

为什么是Stagefright

- Android系统中用于解析音视频文件的逻辑算法库
- 支持大量多媒体文件格式（Aac，Amr，Mp3，Ogg，Vorbis，MPEG-1，H.262，H.264等）
- 从2015年至今，持续爆出大量漏洞
- 仍然有很多0day！

Stagefright工作流程

1. 跨进程请求提取器
2. 提取音频和视频
3. 跨进程请求解码器
4. 解码，播放

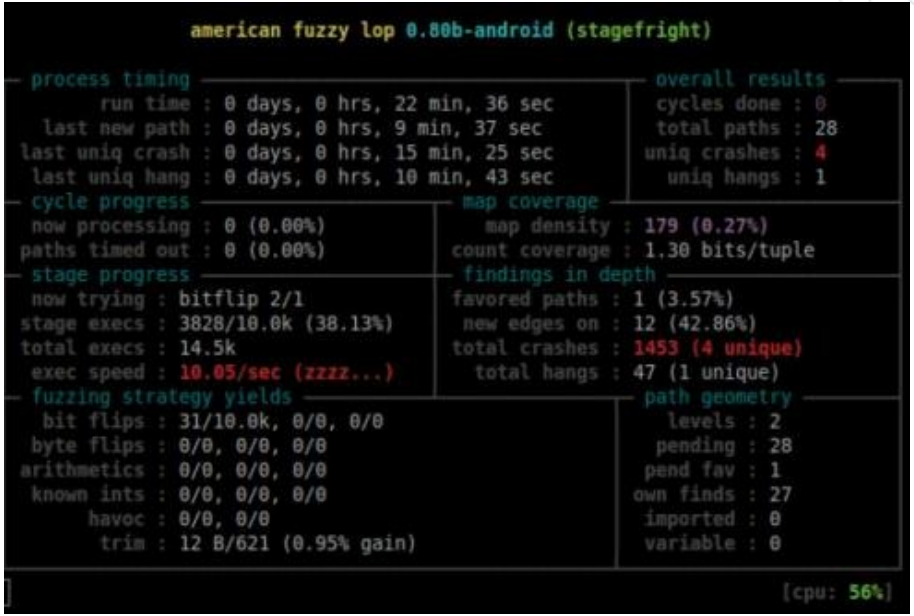


Stagefright漏洞挖掘相关研究

- MFFA
  - 生成输入样本（BFF，FuzzBox，Radamsa，AFL）
  - 通过adb push发送样本到Android设备（真机或模拟器）
  - 通过logcat监控结果
  - 保存crash样本
- AFL in Andrid
  - Fuzz Stagefright with AFL in Android

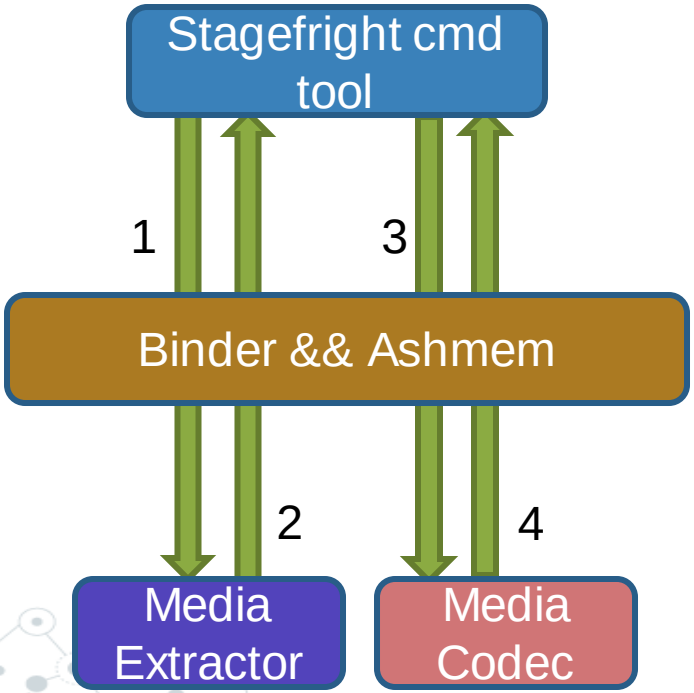
不足

- MFFA
  - 样本质量不佳
  - adb并不可靠
  - logcat捕获crash有延迟
  - Android设备性能低
- AFL in Andrid
  - Android设备性能低



Fuzzing Stagefright with AFL in Linux

需要解决的问题



1. Linux默认无Binder和Ashmem驱动
2. 依赖于服务进程
3. 指令架构不同

移植Binder和Ashmem驱动

- 重新配置，编译，安装Linux内核

```
# CONFIG_ANDROID=y && CONFIG_ANDROID_BINDER_IPC=y && CONFIG_ASHMEM=y
```

- 配置udev规则

```
$ echo -e "KERNEL==\"binder\", MODE=\"0666\"\nKERNEL==\"ashmem\",
MODE=\"0666\"" | sudo tee /etc/udev/rules.d/android.rules
```

Remote -> Local

修改代码，使得Stagefright，MediaExtractor，MediaCodec位于同一进程

```
// 跨进程获取对象实例
// 从servicemanager进程获取media.player进程服务的引用
sp<IBinder> binder = defaultServiceManager()-
>getService(String16("media.player"));
// 类型转换
sp<IMediaPlayerService> service = interface_cast<IMediaPlayerService>(binder);
// 从media.player进程中获取MediaCodecList类的实例
sRemoteList = service->getCodecList();

// 本地创建对象实例
sRemoteList = new MediaCodecList;
```

http://ele7enxxh.com/downloads/stagefright.diff

编译为x86版本

- 指令架构不同
  - 编译x86版本的AOSP

```
$ source build/envsetup.sh
$ lunch aosp_x86-eng
$ make -j16
```

  - 编译x86版本的Stagefright

```
$ cd framework/media/cmds/stagefright
$ mm -j16
```

配置运行环境

- 链接库

```
$ sudo ln -s out/system /system
```

- 拷贝配置文件

```
$ sudo cp out/system/etc/media_codecs_google_audio.xml /etc
$ sudo cp out/system/etc/media_codecs_google_telephony.xml /etc
$ sudo cp out/system/etc/media_codecs_google_video.xml /etc
$ sudo cp out/system/etc/media_codecs.xml /etc
```

修改AFL

Android toolchain无法识别shmat，所以我们需要替换shmat为syscall

```
shmat(shm_id, NULL, 0);
```

```
#ifdef __x86_64__
syscall(SYS_shmat, shm_id, NULL, 0);
#else
syscall(SYS_ipc, IPCOP_shmat, shm_id, 0, &__afl_area_ptr, NULL);
#endif
```

- http://ele7enxxh.com/downloads/afl-2.39b.diff

开始Fuzz

- 选定一个文件格式作为Fuzz对象
- 收集输入样本
- 使用afl对Stagefright中当前文件格式对应的模块进行插桩，不要对无关的库插桩
- 第一轮Fuzz
- 第二轮Fuzz，使用afl-cmin和afl-tmin优化输入样本
- 第三轮Fuzz，开启ASAN

如何获取样本

- AOSP仓库中包含各种文件格式的测试样本，且质量很高
- Google大法好

```
e.g. -inurl:htm -inurl:html intitle:"index of" .mp4
```

- 各种转换工具：ffmpeg，convert，格式工厂
- http://samples.mplayerhq.hu/
- https://github.com/MozillaSecurity/fuzzdata

样本的要求

- 控制文件大小，小于200kb
- 符合目标程序要求的文件格式
- 每个样本具有唯一性（不同的代码覆盖）
- 总的代码覆盖率越高越好

产出

- 针对所有Stagefright支持的格式，持续4个月的挖掘
  - 1w+的crashes
  - 包含大量无用（abort）以及重复的crashes
  - 19个独立crashes

提交的issues

☐	☆	S2	P2	Bug	Security Report - [crash (sig 11) in libstagefright_soft_hevcdec.so (i[redacted])]
☐	☆	S1	P3	Bug	crash in media.extractor
☐	☆	S2	P2	Bug	crash (sig 11) in libhevc
☐	☆	S3	P3	Bug	heap-buffer-overflow in libstagefright_soft_mpeg2dec[redacted]
☐	☆	S0	P3	Bug	heap-buffer-overflowin google.hevc.decoder
☐	☆	S3	P3	Bug	sig 11 crash in media.codec
☐	☆	S0	P3	Bug	heap-use-after-free in libstagefright
☐	☆	S3	P3	Bug	crash in mediacodec
☐	☆	S3	P3	Bug	crash in libstagefright_soft_mpeg4dec
☐	☆	S3	P3	Bug	crash in libstagefright_soft_mpeg2dec(0x0)

提交的issues

☆	S2	P2	Bug	Security Report - [crash (sig 11) in libstagefright_soft_hevcdec.so ([redacted])]
☆	S2	P2	Bug	crash (sig 11) in libstagefright_soft_hevcdec.so ([redacted])
☆	S3	P3	Bug	crash in mediacodec(libstagefright_soft_avcdec [redacted])
☆	S1	P3	Bug	Zero division exception occurs inside of the system MP4 decoder on Android6.0.1
☆	S3	P3	Bug	Zero division exception occurs inside of the system MP4 decoder on Android7.0
☆	S3	P3	Bug	sig11 crash in media.codec
☆	S3	P3	Bug	Several vulnerabltities in libstagefright_soft_avcdec
☆	S1	P3	Bug	Mediacodec(vp9-decoder) crashed when decode a vp9 file
☆	S3	P3	Bug	stack-buffer-overflow in libstagefright_soft_mpeg2dec([redacted])

已确认的issues

- CVE-2016-6764：拒绝服务
- CVE-2016-6766：拒绝服务
- 5 Critical issues
- 4 High issues

4.
总结

我到底说了些什么？

总结

- 阐述了为什么要听我的议题的原因
- 通过三个由我挖到的漏洞，介绍了三种代码审计的方法
- 展示了一个更加高效的Stagefright Fuzz方法

成都360招人！

hanzinuo@360.cn

感谢！