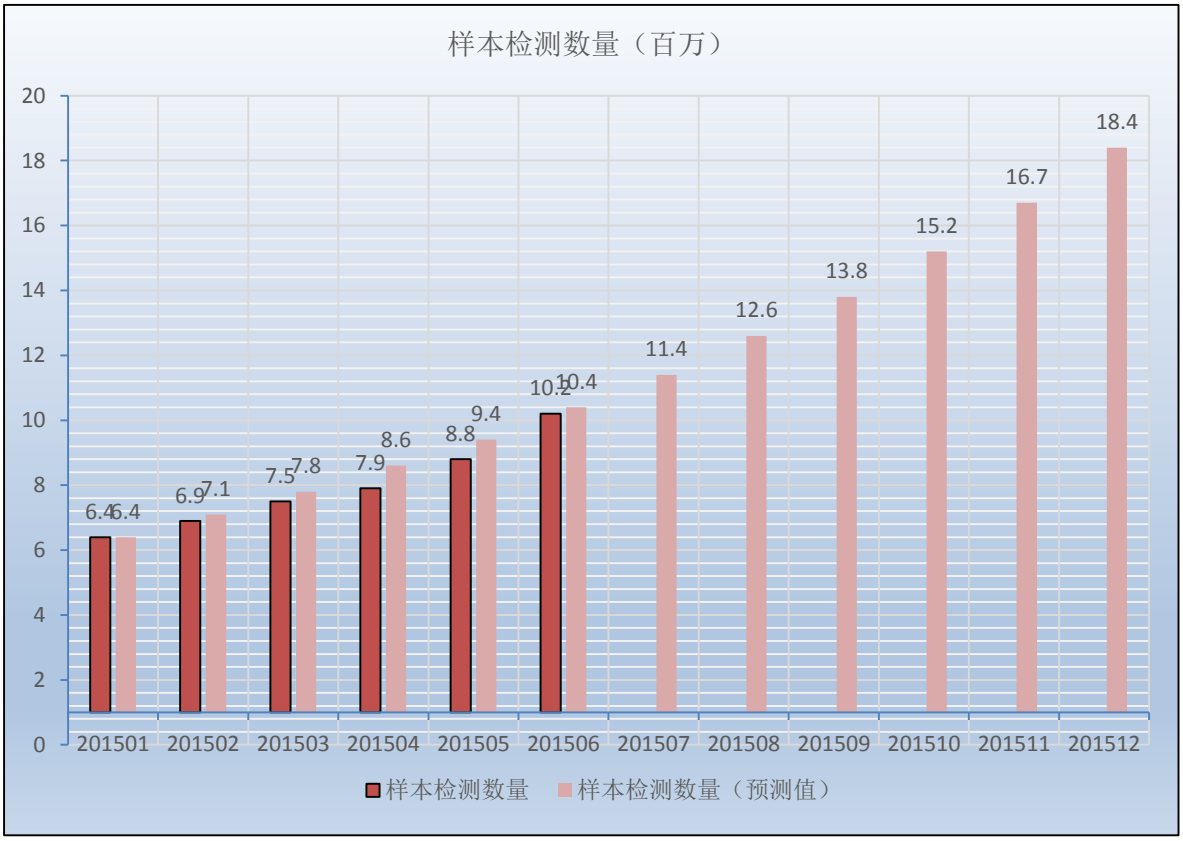
趋势科技移动客户端病毒报告

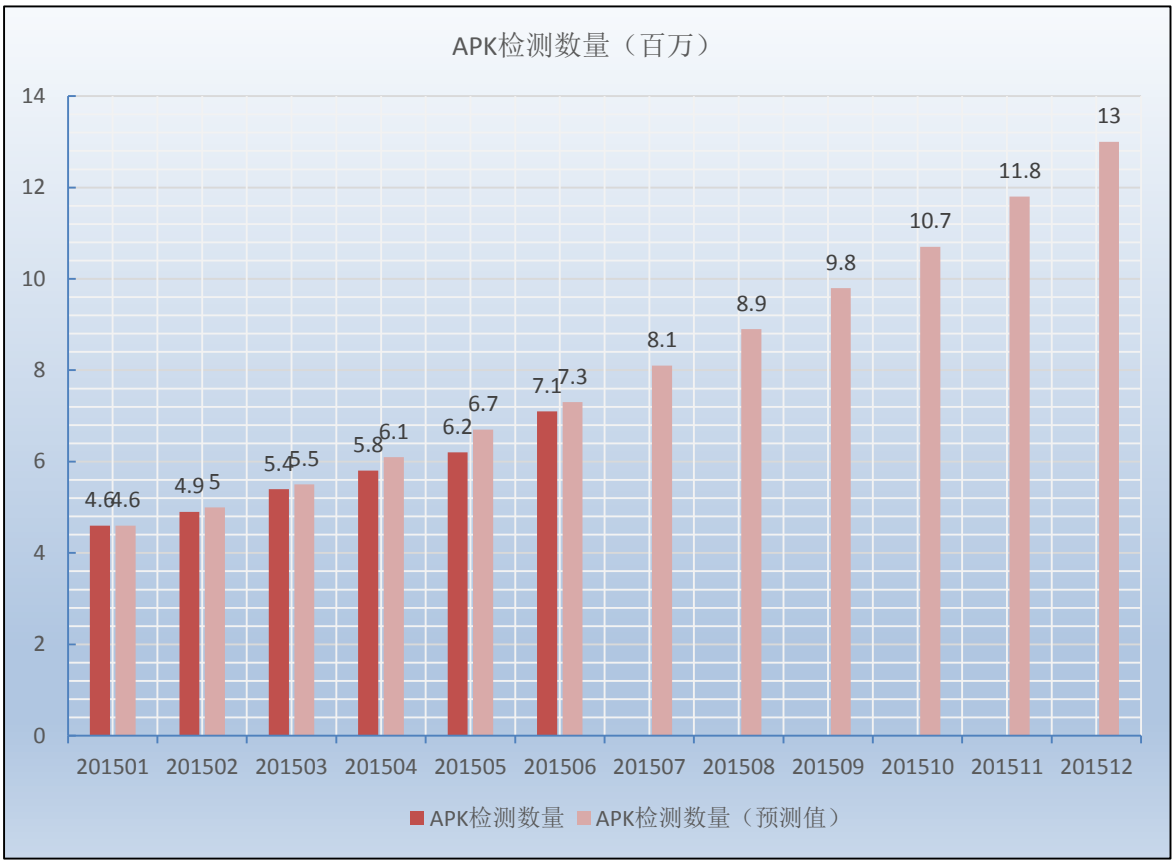
2015年6月移动客户端安全威胁概况

本月，截至 2015.6.30 日，发布中国区移动客户端病毒码 1.907.00，大小 6,471,696 字节。

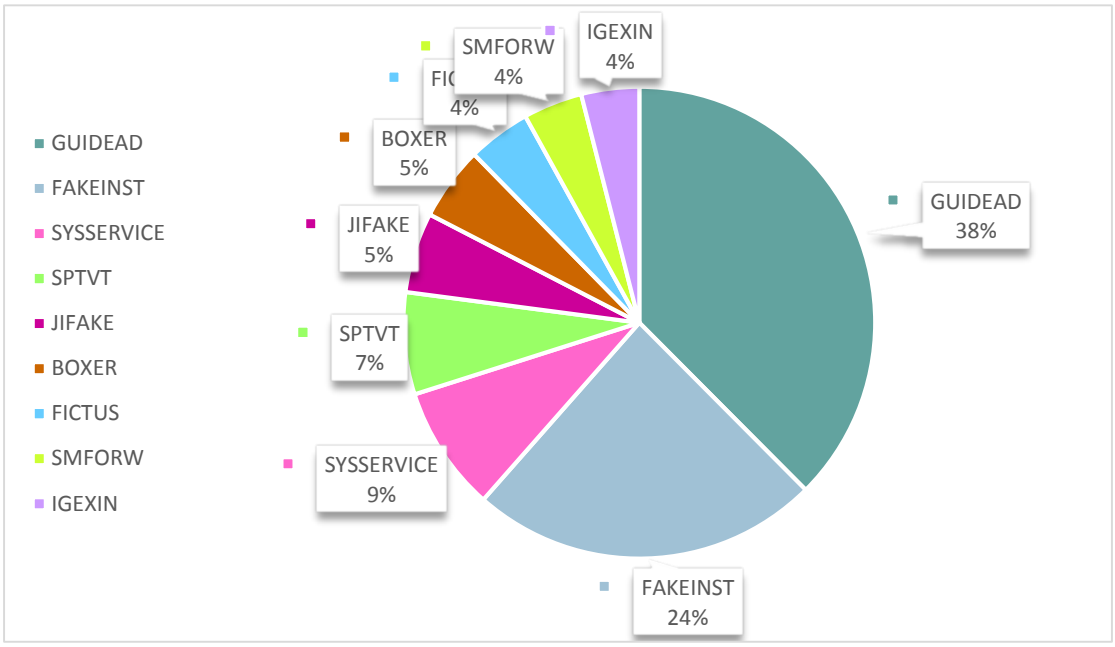
样本检测数量



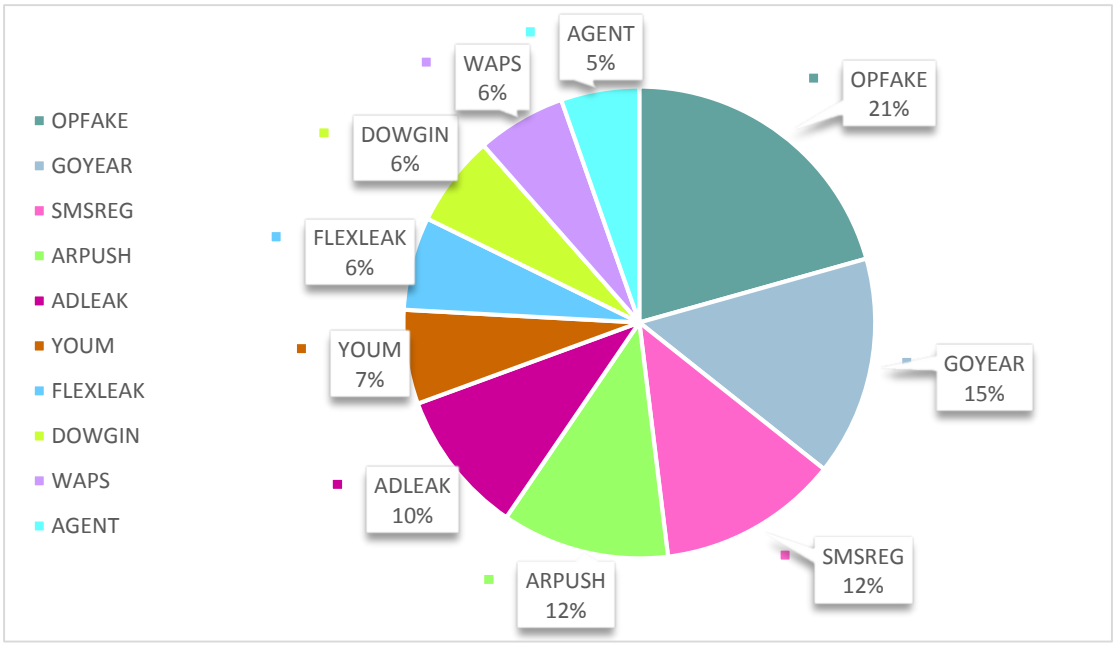
APK检测数量


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
8.1
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



趋势科技曝出安卓内存信息泄露漏洞

我们在研究过程中发现了一个存在于 Android 调试组件 Debuggerd，可以造成设备内存信息的泄露，涉及的安卓版本为从 Ice Cream Sandwich 到 Lollipop.

攻击者可以通过一个精心设计的 ELF 文件造成调试器崩溃，并通过 tombstone 文件和 log 文件泄露出内存。这些泄露的信息可以用来发起拒绝服务攻击，以及绕过 ASLR。漏洞本身不能导致远程代码执行，但是可以和其他漏洞配合来达到该目的。

该漏洞可以通过恶意程序或者重新打包的程序来实现利用，尽管漏洞的严重程度相对较低，不能远程执行代码。该漏洞设计范围还是比较广的，从 Android 4.0 (Ice Cream Sandwich)到 Lollipop(5.x)。该漏洞已经在 Android M 中被修复。

漏洞详情

漏洞的根源在于 Debuggerd 用 sym->st_name 作为字符串拷贝操作的偏移量时，没有进行合理的出错检查。导致攻击者可以利用构造 ELF 控制这个值，来读写不可访问的内存，导致调试器崩溃。反复制造崩溃，可以导致拒绝服务，造成其他程序异常不能发送到调试器。合理地设计攻击时的参数，可以使 Debuggerd 通过 dump 或 log 文件泄露出内存信息。

Android 5.0-5.1 中可以定位到有漏洞的文件为 *external/libunwind/src/elfxx.c:*

```
126     for (sym = symtab;
127          sym < symtab_end;
128          sym = (Elf_W (Sym) *) ((char *) sym + syment_size))
129       {
130         if (ELF_W (ST_TYPE) (sym->st_info) == STT_FUNC
131         && sym->st_shndx != SHN_UNDEF)
132       {
133         if (tdep_get_func_addr (as, sym->st_value, &val) < 0)
134           continue;
135         if (sym->st_shndx != SHN_ABS)
136           val += load_offset;
137         Debug (16, "0x%016lx info=0x%02x %s\n",
138           (long) val, sym->st_info, strtab + sym->st_name);
139
140               /* ANDROID support update */
141               if ((Elf_W (Addr)) (ip - val) < *min_dist
142                   && (Elf_W (Addr)) (ip - val) < sym->st_size)
143               /* End of ANDROID update */
144           {
145             *min_dist = (Elf_W (Addr)) (ip - val);
146             strncpy (buf, strtab + sym->st_name, buf_len);  //the address st_name may be easily controlled by malformed  ELF
```

```
147             buf[buf_len - 1] = '';
148             ret = (strlen (strtab + sym->st_name) >= buf_len
149               ? -UNW_ENOMEM : 0);
150           }
151       }
152       }
```

为了重现这个漏洞，我们可以使用一个可以制作崩溃的 ELF 文件。这可以通过修改 symbol 便宜来实现。当这个 ELF 嵌入 APK 后，前者会被循环执行，漏洞重现。

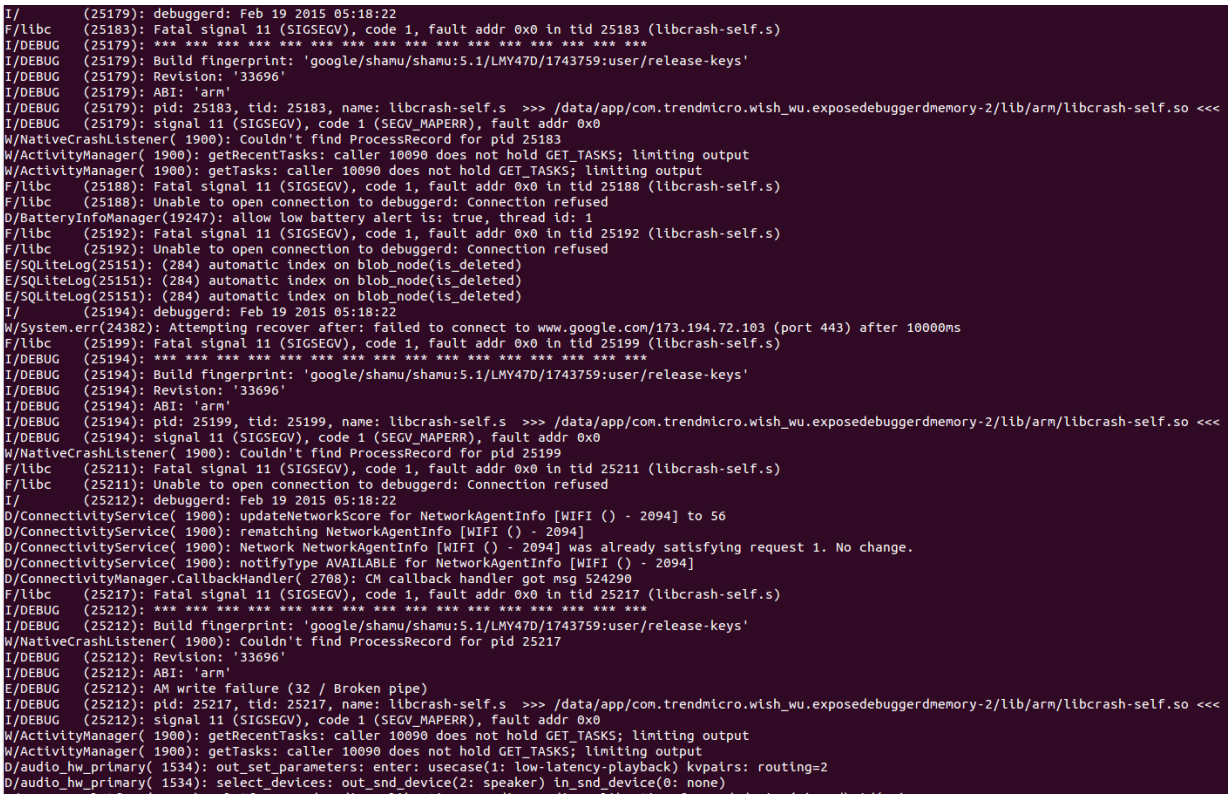


图 1.漏洞的触发

于此类似，相同的漏洞也出现在旧版本的 Android 中（4.x 版，如 Ice Cream Sandwich ,Jelly Bean, and KitKat）.这些版本不使用第三方库 libunwind。Android 4.0 中，漏洞位于文件 *system/core/debuggerd/symbol_table.c* 中。

```
155     int j = 0;
156     if (dynsym_idx != -1) {
157         // …and populate them
158         for(i = 0; i < dynnumsyms; i++) {
159             if(dynsyms[i].st_shndx != SHN_UNDEF) {
160                 table->symbols[j].name = strdup(dynstr + dynsyms[i].st_name);//st_name is not carefully checked before using
161                 table->symbols[j].addr = dynsyms[i].st_value;
```

```
162             table->symbols[j].size = dynsyms[i].st_size;
163             XLOG2("name: %s, addr: %x, size: %x\n",
164                 table->symbols[j].name, table->symbols[j].addr, table->symbols[j].size);
165             j++;
166         }
167     }
168 }
169
170 if (sym_idx != -1) {
171     // …and populate them
172     for(i = 0; i < numsyms; i++) {
173         if((syms[i].st_shndx != SHN_UNDEF) &&
174             (strlen(str+syms[i].st_name)) &&
175             (syms[i].st_value != 0) && (syms[i].st_size != 0)) {
176             table->symbols[j].name = strdup(str + syms[i].st_name);
                                //st_name is not checked before using
177             table->symbols[j].addr = syms[i].st_value;
178             table->symbols[j].size = syms[i].st_size;
179             XLOG2("name: %s, addr: %x, size: %x\n",
180                 table->symbols[j].name, table->symbols[j].addr, table->symbols[j].size);
181             j++;
182         }
183     }
184 }
```

Android 4.1-4.4 中，漏洞位于 *system/core/libcorkscrew/symbol_table.c*

```
150 size_t symbol_index = 0;
151 if (dynsym_idx != -1) {
152     // …and populate them
153     for (int i = 0; i < dynnumsyms; i++) {
154         if (dynsyms[i].st_shndx != SHN_UNDEF) {
155             table->symbols[symbol_index].name = strdup(dynstr + dynsyms[i].st_name);//st_name is
not checked before using
156             table->symbols[symbol_index].start = dynsyms[i].st_value;
157             table->symbols[symbol_index].end = dynsyms[i].st_value + dynsyms[i].st_size;
158             ALOGV("  [%d] '%s' 0x%08x-0x%08x (DYNAMIC)",
159                 symbol_index, table->symbols[symbol_index].name,
160                 table->symbols[symbol_index].start, table->symbols[symbol_index].end);
161             symbol_index += 1;
162         }
163     }
164 }
165
```

```
166     if (sym_idx != -1) {
167         // …and populate them
168         for (int i = 0; i < numsyms; i++) {
169             if (syms[i].st_shndx != SHN_UNDEF
170                     && str[syms[i].st_name]
171                     && syms[i].st_value
172                     && syms[i].st_size) {
173                 table->symbols[symbol_index].name = strdup(str + syms[i].st_name);//st_name is type of
                                                                                    uint32_t not be checked
174                 table->symbols[symbol_index].start = syms[i].st_value;
175                 table->symbols[symbol_index].end = syms[i].st_value + syms[i].st_size;
176                 ALOGV("   [%d] '%s' 0x%08x-0x%08x",
177                         symbol_index, table->symbols[symbol_index].name,
178                         table->symbols[symbol_index].start, table->symbols[symbol_index].end);
179                 symbol_index += 1;
180             }
181}
182   }
```

```
F/libc    ( 8274): Fatal signal 11 (SIGSEGV) at 0x00000000 (code=1), thread 8274 (libcrash-self.s)
I/DEBUG   ( 8261): *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
I/DEBUG   ( 8261): Build fingerprint: 'Lenovo//:4.1.2/JZO54K/eng.andforce.20121229.012430:user/test-keys'
I/DEBUG   ( 8261): pid: 8274, tid: 8274, name: libcrash-self.s  >>> /data/data/com.trendmicro.wish_wu.expo
I/DEBUG   ( 8261): signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 00000000
I/DEBUG   ( 8261):     r0 00000001  r1 befffb64  r2 befffb6c  r3 00000000
I/DEBUG   ( 8261):     r4 2a0002ec  r5 befffb64  r6 00000001  r7 befffb6c
I/DEBUG   ( 8261):     r8 00000000  r9 00000000  sl 00000000  fp befffb2c
I/DEBUG   ( 8261):     ip 4007a00c  sp befffb20  lr 40037ed5  pc 2a000304  cpsr 60000050
I/DEBUG   ( 8261):     d0  365695402de5314b  d1  0000000000000069
I/DEBUG   ( 8261):     d2  0000000000000062  d3  0000000000000063
I/DEBUG   ( 8261):     d4  0000000000000000  d5  0000000000000000
I/DEBUG   ( 8261):     d6  0000000000000000  d7  3662550b00000000
I/DEBUG   ( 8261):     d8  0000000000000000  d9  0000000000000000
I/DEBUG   ( 8261):     d10 0000000000000000  d11 0000000000000000
I/DEBUG   ( 8261):     d12 0000000000000000  d13 0000000000000000
I/DEBUG   ( 8261):     d14 0000000000000000  d15 0000000000000000
I/DEBUG   ( 8261):     d16 41cb312a858a9fbe  d17 3f50624dd2f1a9fc
I/DEBUG   ( 8261):     d18 41c6f298a5800000  d19 0000000000000000
I/DEBUG   ( 8261):     d20 0000000000000000  d21 0000000000000000
I/DEBUG   ( 8261):     d22 0000000000000000  d23 0000000000000000
I/DEBUG   ( 8261):     d24 0000000000000000  d25 0000000000000000
I/DEBUG   ( 8261):     d26 0000000000000000  d27 0000000000000000
I/DEBUG   ( 8261):     d28 0000000000000000  d29 0000000000000000
I/DEBUG   ( 8261):     d30 0000000000000000  d31 0000000000000000
I/DEBUG   ( 8261):     scr 00000010
I/DEBUG   ( 8261):
I/DEBUG   ( 8261): backtrace:
I/DEBUG   ( 8261):     #00  pc 00000304  /data/data/com.trendmicro.wish_wu.exposedebuggerdmemory/lib/libcr
<99>^C#<C0><F8>+24)
I/DEBUG   ( 8261):     #01  pc 00016ed3  /system/lib/libc.so (__libc_init+38)
I/DEBUG   ( 8261):
I/DEBUG   ( 8261): stack:
I/DEBUG   ( 8261):         befffae0  00001000
I/DEBUG   ( 8261):         befffae4  4003a6f3  /system/lib/libc.so (mmap+60)
I/DEBUG   ( 8261):         befffae8  4007a00c
I/DEBUG   ( 8261):         befffaec  00001000
I/DEBUG   ( 8261):         befffaf0  4006a38c
I/DEBUG   ( 8261):         befffaf4  00000000
I/DEBUG   ( 8261):         befffaf8  00001000
I/DEBUG   ( 8261):         befffafc  40048f31  /system/lib/libc.so (__libc_fini)
I/DEBUG   ( 8261):         befffb00  ffffffff
I/DEBUG   ( 8261):         befffb04  00000000
I/DEBUG   ( 8261):         befffb08  00000000
I/DEBUG   ( 8261):         befffb0c  2a0002ec  /data/data/com.trendmicro.wish_wu.exposedebuggerdmemory/li
<E7>^H<99>^C#<C0><F8>)
I/DEBUG   ( 8261):         befffb10  befffb64  [stack]
I/DEBUG   ( 8261):         befffb14  00000001
```

图 2 内存泄露

我们首先发现该漏洞并于今年 4 月 27 日将其报告给 Google，对方随即确认了该漏洞并给与低风险评级。5 月 15 日 Android Open Source Project(AOSP)已经包含了对这个漏洞的修补。

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。