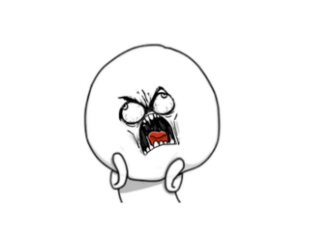
号称安卓年度最大漏洞的“Janus”漏洞来袭



下面是跟各个湿父交流总结出漏洞验证的步骤  
  
**攻击原理:**  
1、安卓在4.4中引入了新的执行虚拟机ART，这个虚拟机经过重新的设计，实现了大量的优化，提高了应用的运行效率。与“Janus”有关的一个技术点是，ART允许运行一个raw dex，也就是一个纯粹的dex文件，不需要在外面包装一层zip。而ART的前任DALVIK虚拟机就要求dex必须包装在一个zip内部且名字是classes.dex才能运行。当然ART也支持运行包装在ZIP内部的dex文件，要区别文件是ZIP还是dex，就通过文件头的magic字段进行判断：ZIP文件的开头是‘PK’, 而dex文件的开头是’dex’.

2、ZIP文件的读取方式是通过在文件末尾定位central directory, 然后通过里面的索引定位到各个zip entry，每个entry解压之后都对应一个文件。

### **影响的范围：**

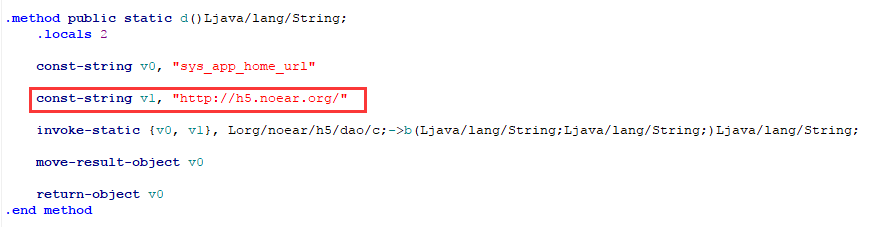
1. 安卓5.0-8.0的各个版本系统；

2. 使用安卓Signaturescheme V1签名的App APK文件。

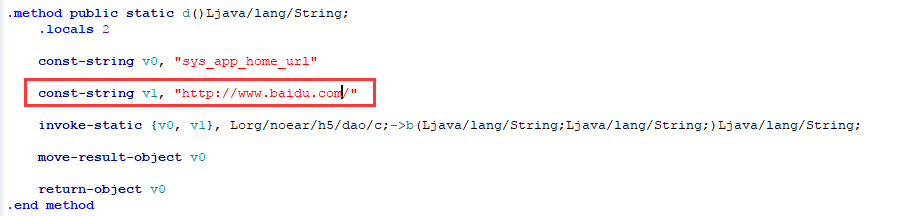
**漏洞验证：**

随机选择一个APP进行验证

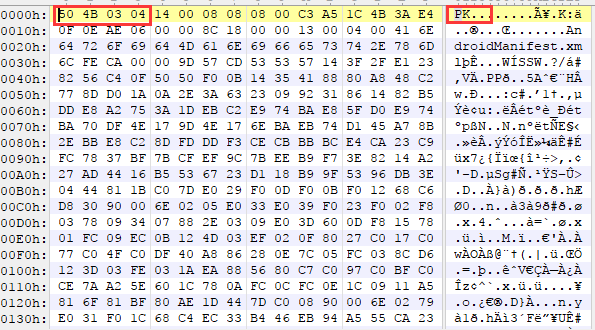
将原始APP进行反编译，修改其中的URL信息：



修改后：

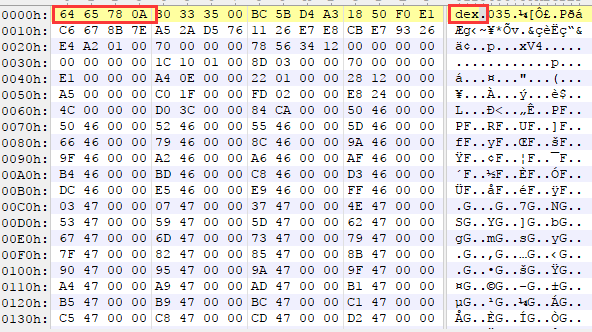


将该文件重新编译后获取dex文件，查看原始apk文件结构，在开始地址处显示是zip压缩文件格式：

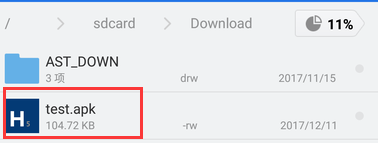
  
利用网络公布的poc将修改后的dex文件注入到原始apk文件中，并对apk文件格式进行一定修复，最后生成注入后的apk文件为test.apk：



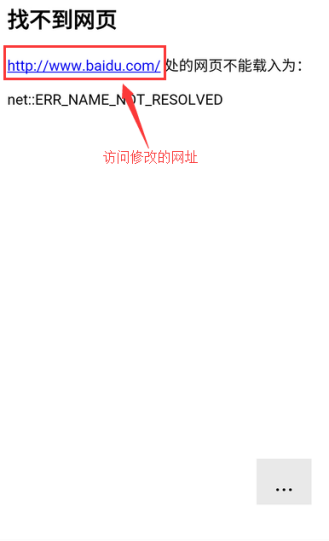
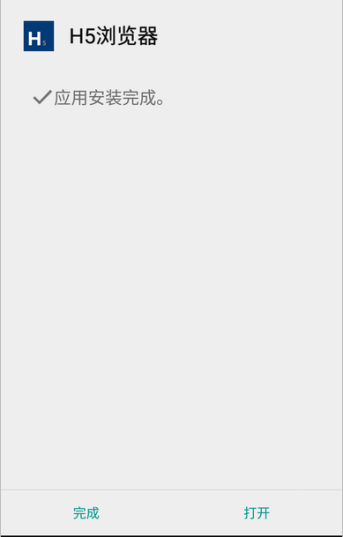
查看test.apk的文件结构，文件的开始地址处显示是dex文件类型：



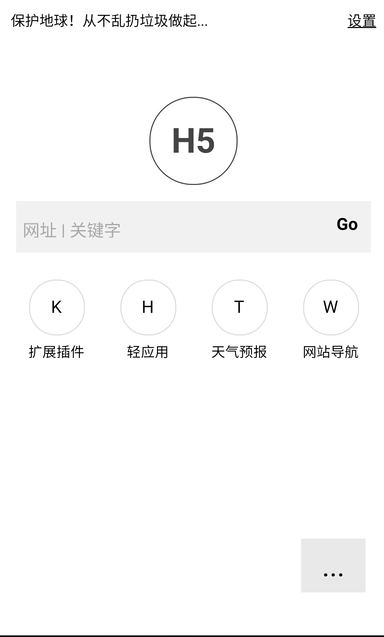
对注入后的apk进行安装：







原始apk安装之后，访问的连接：



该漏洞利用条件比较苛刻，对仿冒应用有较高利用价值，如果是第三方dex文件，则要根据原应用程序所调用的权限和组件进行编写，之后还要对apk的文件结构进行修复，修复难度较大。

**处置措施**  
3.1 使用Signature scheme V2签名机制；

3.1 升级Android系统到最新版本。

**扩展**：  
**安卓系统这些年爆出的安全漏洞**

**“MasterKey”漏洞**

“Janus”是一个签名与校验漏洞，其实，这不是安卓第一次爆出此类漏洞。在2013年 Black Hat上，Bluebox的安全团队公布了一个“MasterKey”漏洞。该漏洞影响包括当时最新的安卓6.0系统及以下所有系统。那么，这些漏洞是怎么形成的呢？

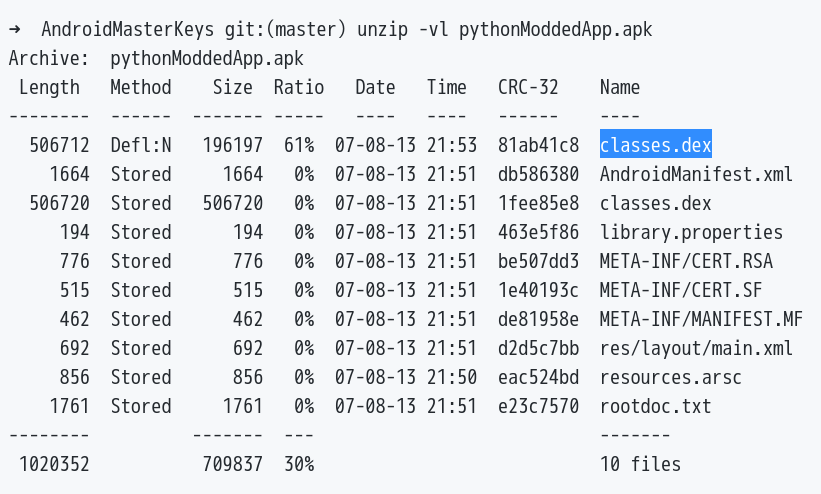
“MasterKey”漏洞原理是基于APK（ZIP文件格式）里面的多个ZipEntry实现的，具体如下：

1. 向原始的App APK的前部添加一个攻击的classes.dex文件（A)；

2. 安卓系统在校验时计算了A文件的hash值，并以”classes.dex”字符串做为key保存；

3. 然后安卓计算原始的classes.dex文件（B），并再次以”classes.dex”字符串做为key保存，这次保存会覆盖掉A文件的hash值，导致Android系统认为APK没有被修改，完成安装；

4. APK程序运行时，系统优先以先找到的A文件执行，忽略了B，导致漏洞的产生。

修复方式：禁止安装有多个同名ZipEntry的APK文件。

**“9695860”漏洞**

MasterKey漏洞爆出后没多久，国内的“安卓安全小分队”再爆出一个类似的漏洞。这个漏洞非常精巧：利用了Zip local file header在计算时候的一个整形溢出漏洞。

具体原因：

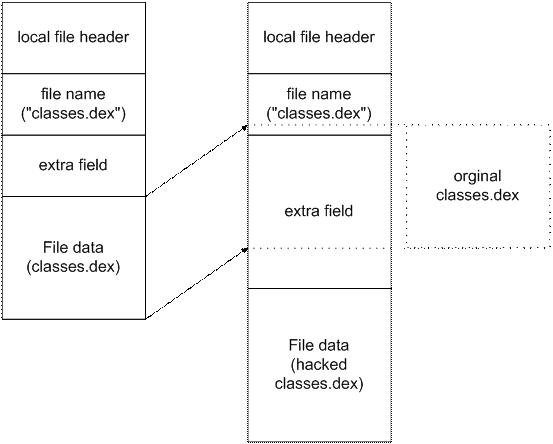
1. 向原有的APK中的classes.dex文件B替换为攻击文件A，并添加一个大小为0xFFFD的extra field；

2. 将原始dex文件B去除头3个字节写入extra field；

3. Android系统在校验签名时使用的是Java代码的short，将0xFFFD以16位带符号整形的方式解析得到-3, 并解析出原始的文件B，Android认为程序APK无修改，正常安装；

4. 系统在执行时使用C代码的uint16，将0xFFFD以16位无符号整形方式，得到攻击文件B。

这个漏洞的精巧之处在于，DEX文件以‘dex’字符串开头，而classes.dex以这个字符串结尾，通过-3的值将这两个内容在文件中重叠起来，因此这也限制了“9695860”漏洞只能对classes.dex进行攻击。



**“9950697”漏洞**

在“9695860”漏洞爆出不久后，APK文件中被发现存在类似的整形溢出漏洞，这个比“9695860”漏洞更容易利用且可以攻击APK中的任意文件。

原因是安卓默认认为Zip中local file header和central directory entry中的文件名长度和和extra的长度是一致的。安装过程中java代码在处理时出现溢出，读取到了正常的文件B，通过校验，APK正常安装。运行过程中，C代码处理时没有溢出，读取到了攻击的文件A。

