目录

[第一章 绪论 3](#_Toc388512258)

[1.1 研究背景 3](#_Toc388512259)

[1.2 Root漏洞现状 3](#_Toc388512260)

[1.3 手机Root后的影响 3](#_Toc388512261)

[1.3.1 为什么android不开放Root权限？ 3](#_Toc388512262)

[1.3.2 Root后的优点 4](#_Toc388512263)

[1.3.3 Root后的危害 4](#_Toc388512264)

[第二章 Android Root原理分析 6](#_Toc388512265)

[2.1 Linux Root定义 6](#_Toc388512266)

[2.2 Android Root原理 6](#_Toc388512267)

[第三章 原生android系统Root漏洞 7](#_Toc388512268)

[3.1 sendpage漏洞分析 7](#_Toc388512269)

[3.1.1 sendpage漏洞 7](#_Toc388512270)

[3.1.2 漏洞原理 8](#_Toc388512271)

[3.1.3 漏洞的利用 8](#_Toc388512272)

[3.2 udev漏洞分析 9](#_Toc388512273)

[3.2.1 udev漏洞 9](#_Toc388512274)

[3.2.2 udev漏洞原理 10](#_Toc388512275)

[3.2.3 udev漏洞的利用 10](#_Toc388512276)

[3.3 adb setuid漏洞 12](#_Toc388512277)

[3.3.1 adb setuid漏洞信息 12](#_Toc388512278)

[3.3.2 adb setuid漏洞原理 12](#_Toc388512279)

[3.3.3 adb setuid漏洞的利用 12](#_Toc388512280)

[3.3.4 adb setuid漏洞的修复 13](#_Toc388512281)

[3.4 zergRush漏洞分析 14](#_Toc388512282)

[3.4.1 zergRush漏洞 14](#_Toc388512283)

[3.4.2 漏洞原理 14](#_Toc388512284)

[3.4.3 漏洞利用 14](#_Toc388512285)

[3.5 psneuter漏洞分析 15](#_Toc388512286)

[3.5.1 psneuter漏洞 15](#_Toc388512287)

[3.5.2 psneuter漏洞原理 15](#_Toc388512288)

[3.5.3 psneuter漏洞利用 16](#_Toc388512289)

[3.6 GingerBreak漏洞分析 16](#_Toc388512290)

[3.6.1 GingerBreak漏洞 16](#_Toc388512291)

[3.6.2 GingerBreak漏洞原理 16](#_Toc388512292)

[3.6.4 GingerBreak漏洞的利用 17](#_Toc388512293)

[3.6.5漏洞的修复 18](#_Toc388512294)

[3.7 mempodroid漏洞分析 18](#_Toc388512295)

[3.7.1 mempodroid漏洞 18](#_Toc388512296)

[3.7.2 mempodroid漏洞原理 18](#_Toc388512297)

[3.7.3 mempodroid漏洞利用 19](#_Toc388512298)

[3.7.4 mempodroid漏洞的修复 19](#_Toc388512299)

[3.8 libperf\_event漏洞分析 20](#_Toc388512300)

[3.8.1 libperf\_event漏洞 20](#_Toc388512301)

[3.8.2 libperf\_event漏洞原理 20](#_Toc388512302)

[3.8.3 libperf\_event漏洞利用 21](#_Toc388512303)

[3.8.4 libperf\_event漏洞的修复 22](#_Toc388512304)

[第四章 厂商定制系统漏洞 22](#_Toc388512305)

[4.1 TacoRoot漏洞 23](#_Toc388512306)

[4.1.1 TacoRoot简介 23](#_Toc388512307)

[4.1.2 TacoRoot原理 23](#_Toc388512308)

[4.2 BurritoRoot漏洞 23](#_Toc388512309)

[4.2.1 burroitoRoot漏洞简介 23](#_Toc388512310)

[4.2.2 BurroitoRoot漏洞原理 23](#_Toc388512311)

[4.3 LGPwn漏洞 23](#_Toc388512312)

[4.3.1 LGPwn漏洞简介 23](#_Toc388512313)

[4.3.2 LGPwn漏洞原理 24](#_Toc388512314)

[4.4 Motorola Razr ICS漏洞 25](#_Toc388512315)

[4.4.1 Motorola Razr ICS漏洞简介 25](#_Toc388512316)

[4.4.2 Motorola Razr ICS漏洞原理 25](#_Toc388512317)

[4.5 HTC One X AT&T Root漏洞 26](#_Toc388512318)

[4.5.1 HTC One X AT&T Root漏洞简介 26](#_Toc388512319)

[4.5.2 HTC One X AT&T Root漏洞原理 26](#_Toc388512320)

[4.6 Rooting the Sony Tablet S 27](#_Toc388512321)

[4.6.1 漏洞简介 27](#_Toc388512322)

[4.6.2 漏洞原理 27](#_Toc388512323)

[第五章 总结 28](#_Toc388512324)

[5.1漏洞挖掘分析技术 28](#_Toc388512325)

[5.1.1 手工测试 29](#_Toc388512326)

[5.1.2 Fuzzing技术 29](#_Toc388512327)

[5.1.3 二进制比对技术 30](#_Toc388512328)

[5.1.4 静态分析 31](#_Toc388512329)

[5.2 漏洞总结 31](#_Toc388512330)

[5.3漏洞发现步骤 32](#_Toc388512331)

[参考文献 32](#_Toc388512332)

# 第一章 绪论

## 1.1 研究背景

科技的发展有时候是一把双刃剑，手机的功能越来越强大，给人们带来更便利和更丰富的信息服务以及良好的用户体验的同时，随之而来的是更加严重的安全保密隐患。手机中一般存储了大量涉及用户隐私的数据，特别是用户的通讯录，短信息，通话记录，照片等，另外，只能手机的许多功能和服务是涉及用户资费的，如用户的网络流量，手机通话等，与用户的经济利益直接相关。用户的手机一旦被植入病毒或者而已软件，就可能会造成用户私密信息的泄露，一些吸费软件可能会自动打电话或者发短信，在用户不知情的情况下定制一些用户用不到的付费增值业务，造成用户的手机话费剧增，手机的安全问题日益严重，给手机用户带来了很多困扰。

## 1.2 Root漏洞现状

由于Google以Apache免费开源许可证的授权方式，发布的Android的源代码，所以任何人都可以修改代码，定制Android系统。因此，有些Root漏洞只出现在某些机型上。据不完全统计，目前已发现的Android漏洞大概有40多个,其中原生系统漏洞大概有10个左右，如CVE-2009-1185 Exploid ，CVE-2011-1823 Gingerbreak， CVE-2012-0056 Mempodroid CVE-2009-2692 Wunderbar， CVE-2011-3874 ZergRush，Zimperlich / Zygote setuid这些漏洞一般具有通用性，其它为厂商定制代码所产的漏洞，不具有通用性

## 1.3 手机Root后的影响

### 1.3.1 为什么android不开放Root权限？

Android操作系统是一个多用户的Linux系统中，每个应用程序是不同的用户

默认情况下，系统分配的每个应用程序的唯一的Linux用户ID（该ID用于仅由系统和未知的应用程序）。该系统设置在应用程序中的所有文件的权限，以便只有分配给该应用程序的用户ID都可以访问它们。

每个进程都有自己的虚拟机（VM），所以应用程序的代码在隔离从其他应用程序运行

默认情况下，每个应用程序都在它自己的Linux进程中运行。 Android的启动过程中，当需要执行任何应用程序的组件，然后关闭该进程，当它不再需要时或系统必须恢复内存供其他应用程序

总而言之，android系统会为每个app分配一个用户id，使该app只能在该用户的权限下运行，以保证其安全性

### 1.3.2 Root后的优点

现在很多人买了android手机之后，就考虑如何刷机了，那Root手机有那些好处了？

1.删除系统一些自带软件。在root一起一些系统软件在软件管理里面没有卸载选项。但是root之后，可以配合一些文件管理软件直接删除手机内存里的系统应用文件。这样可以节省手机内存空间

2.自由定制手机的主题和字体，

3.管理开机自启动软件，开机会有部分应用在后台自动运行或者联网后自动运行，这不仅拖慢我们开机时间，也会让设备多出很多软件推送信息，所以我们可以通过第三方的软件来阻止屏蔽这些程序。

4.将应用安装到SD卡，由于android默认安装位置是内存上，root后可以将程序安装到sd卡中，节约手机内存

### 1.3.3 Root后的危害

Root之后虽然给用户带来了诸多优点，但也留下了不少隐患。隐私泄露，信息泄露等都会给带来很多困扰。

1.静默安装

拥有root权限的病毒可以完成静默安装应用A自身捆绑一个应用B（病毒），用B可以偷窥隐私，扣话费等。当用户发现自己中毒后，可能会删除应用A，但是却不敢删除应用B，因为它伪装为系统应用，普通用户无法知道应用B到底是做什么的，只能刷机，注意：刷机的时候如果不病毒也备份了，那就悲剧了。另一种变体：A并不直接捆绑病毒，而是在用户安装了之后，在后台偷偷下载病毒，

2.广告拦截

有些软件有广告拦截功能，就是说，当你的手机中有人弹出广告通知时候，你打开广告拦截功能，就能知道是哪个应用弹出的广告，然后把这个应用杀死，。病毒也可以做到这一点，比如在凌晨，A应用下载一个应用B(病毒)，然后检查通知栏，如果发现了有关应用B的通知，就将改通知清理掉，这样病毒就悄无声息得安装到了用户的手机

3.篡改superuser（获得一次root权限即可永久获得root权限）

病毒申请root权限的时候superuser会弹出对话框，见下图1.3。没有root，病毒就会受到很大限制。但是如果用户不小心（病毒申请root权限的时候用户基本

上不知道这是个病毒）给了病毒一次root权限

图1.3 应用申请Root权限



用户以为它只给病毒B一次root权限，下次需要的时候，还会经过这里。其实病毒B可以利用这一次root权限而永久获得root权限，所以，没事应该去spueruser中看看，是不是有不该有的应用常驻在里面，把它从列表中删除，最好也把它从手机中删除

 4.禁止开机启动与停用软件

病毒如果拥有了root权限，那么它就可以禁止其他软件开机启动，甚至停用其他软件

android系统自带的停用软件的功能在android4.1才开始有，而且只有系统应用才能使用停用功能，见下图



图1.4 强行停止其他应用的运行

停用的软件在桌面应用（或者叫home、launcher）中是看不到的，就好像根本没有安装一样，只有到手机中设置那里才能看到，而且在列表中没有任何标识，需要查看详细信息才行（就是得到上面的这个页面中查看）

5.静音拍照和录音

android系统中要想拍照，必须提供一个窗口才行，但是对于对于具有root权限的应用，可以在后台实现静音拍照，窃取用户隐私

1.4

Root自己的手机虽然可以带来很多方面，但是留下了很多的隐患。

# 第二章 Android Root原理分析

## 2.1 Linux Root定义

root存在于UNIX系统（如AIX、BSD等）和类UNIX系统（如Debian、Redhat、Ubuntu以及Android系统等各个发行版的Linux）中，超级用户一般命名为root，相当于Windows系统中的Administrator。root是系统中唯一的超级用户，具有系统中所有的权限，如启动或停止一个进程，删除或增加用户，增加或者禁用硬件等等

## 2.2 Android Root原理

在root过的手机上面获得root权限的代码如下：

|  |
| --- |
| Process process = Runtime.getRuntime().exec("su");  DataOutputStream os = new DataOutputStream(process.getOutputStream());  os.writeBytes("mount -oremount,rw /dev/block/mtdblock3 /system\n");  os.writeBytes("busybox cp /data/data/com.koushikdutta.superuser/su /system/bin/su\n");  os.writeBytes("busybox chown 0:0 /system/bin/su\n");  os.writeBytes("chmod 4755 /system/bin/su\n");  os.writeBytes("exit\n");  os.flush(); |

从上面代码我们可以看到首先要运行su程序，所以root的秘密都在su程序中

Andoid的内核是linux,在linux下获取root权限的时候是执行sudo或者su,接下来系统会提示你输入root用户的密码，密码正确就获取了root权限。Android本身就不想让你获取root权限，大部分手机出厂的时候根本就没有su这个程序。所以你想获得android的root权限，第一步就是要把编译好的su文件拷贝到android手机的/system/bin或者/system/xbin目录下，接下来你就可以在android手机的adb shell或串口下输入su,但android里的 su和linux里su是不一样的,android里的su不是靠验证密码，而是看你原来的权限是什么。意思是如果你是root,那你可以通过su切换到别的用户，但是如果你是root以外的用户，就不能切换回root,会提示你permission denied.

su.c的一部分：

|  |
| --- |
| */*\* Until we have something better, only root and the shell can use su. \*/  myuid = getuid();  if (myuid != AID\_ROOT && myuid != AID\_SHELL) {  fprintf(stderr,"su: uid %d not allowed to su\n", myuid);  return 1;  }//当uid不是AID\_ROOT和AID\_SHELL,就报错 |

可以看出只允许getuid()为AID\_ROOT和AID\_SHELL的进程可以使用su进行登陆。   
也就是说，Android系统默认的su程序只能root和shell可以用运行su，这个是安全的。如果把这个限制拿掉，就是root破解了，获取了root权限。

linux下的文件一般有四种权限，r代表可读，w代表可写，x代表可执行，-代表没有该权限，但是有的文件比较特殊，它的执行权限标志位是一个s,代表当任何一个用户执行该文件的时候,都拥有该文件所有者的权限，如果这个文件的所有者是root,那么，不管谁执行它，都是以root身份执行的

所以，要root手机，需要把一个所有者是root的su拷贝到android手机上，并把su的权限标志位置成-rwsr-xr-x。它需要这两行代码

|  |
| --- |
| cp /data/tmp/su /system/bin/ #copy su 到/system/分区 chown root:root su #su的所有者置成root chmod 4775 /system/bin/su #把su置成-rwsr-xr-x |

但是执行上面的每一行代码都需要root权限。只有有root权限的情况下才能执行上面两行代码，而这两行代码就是为了让你获得root权限的，这是一个逻辑闭环，那么如何打破这个逻辑闭环呢？一个办法就是找一个本身已经有root权限的进程来启动我上面的两行代码，那我这两行代码一启动就是root权限，就可以顺利执行了。

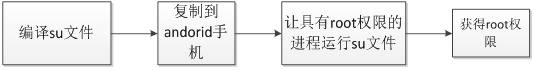


图2.2 获得Root的过程

# 第三章 原生android系统Root漏洞

Root漏洞不是与生俱来的，这是全世界优秀的计算机黑客不懈努力的成果，每个漏洞都是他们孜孜不倦研究的成果。

## 3.1 sendpage漏洞分析

### 3.1.1 sendpage漏洞

发现时间：2009年

发现者：zinx

编号：CVE-2009-2692

漏洞影响范围：android2.1及android2.1以下

漏洞简介：sock\_sendpage()的空指针解引用。因为sock\_sendpage没有对socket\_file\_ops结构的sendpage字段做指针检查，有些模块不具备sendpage功能，初始时赋为NULL，这样，没有做检查的sock\_sendpage有可能直接调用空指针而导致出错并提升权限！

### 3.1.2 漏洞原理

一个本地文件通过socket发送出去，过程：打开文件fd和一个socket,然后循环的从文件中read数据，并将读取的数据发送到socket中，这样，每次读写我们都需要两次系统调用，并且数据被从内核拷贝到用户空间(read),再从用户空间拷贝到内核(send),而sendfile就将这个过程封装在一个系统调用中，避免了多次系统调用，和数据在内核空间和用户空间之间大量拷贝。

/\*in 只能是普通文件， out 是 socket\*/

ssize\_t sendfile(int out\_fd, int in\_fd, off\_t \*offset, size\_t count);

写数据到out\_fd过程会调用sock\_sendpage()函数

sock\_sendpage的代码如下:

|  |
| --- |
| struct socket \*sock; int flags; sock = file->private\_data; flags = !(file->f\_flags & O\_NONBLOCK) ? 0 : MSG\_DONTWAIT; if (more)      flags |= MSG\_MORE; return sock->ops->sendpage(sock, page, offset, size, flags); |

调用sock->ops->sendpage 之前没有判断这个函数指针是否为NULL而导致错误并提升权限

### 3.1.3 漏洞的利用

攻击流程图:

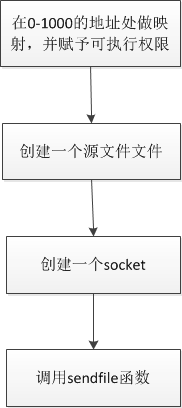


图3.1 sendpage漏洞攻击流程

步骤：

1. 调用mmap()函数，将0-1000的地址做了映射，并赋予可执行属性
2. 在映射的0地址上写下JMP到恶意代码的指令
3. 创建一个临时文件，用作源文件
4. 创建一个socket，注意其类型为PF\_PPPOX
5. sendfile 函数的调用

unlink(template);

|  |
| --- |
| ftruncate(fdin, PAGE\_SIZE);  sendfile(fdout, fdin, NULL, PAGE\_SIZE); //调用一个值为NULL的函数指针 |

sendfile将在系统调用中，cpu进入内核态，触发对0地址的调用，所以，只需要把恶意代码映射到0地址处

3.1.4 漏洞的修补

在调用之前，先判断函数指针是否为空，如果为空，就返回一个错误

## 3.2 udev漏洞分析

### 3.2.1 udev漏洞

发现时间：2010年

发现者:” The Android Exploid Crew”小组

编号：CVE-2010-EASY

漏洞影响范围：android2.3及以下

### 3.2.2 udev漏洞原理

由于udev对热插拔消息检测不严，恶意程序可以发送伪热插拔消息，让内核执行自身恶意代码，而内核由于没有检查校区发送是内核还是用户，就执行了恶意程序，从而取得root权限

注：热拔插即带电插拔，热插拔功能就是允许用户在不关闭系统，不切断电源的情况下取出和更换损坏的硬盘、电源或[板卡](http://baike.baidu.com/view/559655.htm)等部件，从而提高了系统对灾难的及时恢复能力、扩展性和灵活性等，在安卓系统中如wifi的打开和关闭，内存卡(sd卡)的拔出和插入等都属于热拔插

vold工作机制：

  kernel检测到硬件事件，然后广播出去，Native层的init进程通知vold(root权限)去处理该广播信息，vold再通知JNI层的MountService,其与Java应用层交互

  vold进程可以将用户态数据copy到内核，由内核写入设备文件（恶意程序的依赖）

  vold进程应该只处理kernel发送的device的NETLINK的socket消息，但实际上并未检测NETLINK的socket消息的来源，

这样可以广播add device的socket信息，触发硬件处理事件，将恶意代码传入kernel，由其写入设备文件

### 3.2.3 udev漏洞的利用

步骤：

1. 初始化要发送的数据
2. 构建一个NETLINK的套接

|  |
| --- |
| if ((sock = socket(PF\_NETLINK, SOCK\_DGRAM, NETLINK\_KOBJECT\_UEVENT)) < 0)          die("[-] socket"); |

3.创建要热拔插的文件

4.建立一个data文件，为指向系统的hotplug的符号链接

 symlink("/proc/sys/kernel/hotplug", "data");

5.构建发送给内核的消息，内容为进行热拔插

|  |
| --- |
| snprintf(buf, sizeof(buf), "ACTION=add%cDEVPATH=/..%s%c"                   "SUBSYSTEM=firmware%c"                   "FIRMWARE=../../..%s/hotplug%c", 0, basedir, 0, 0, basedir, 0);          printf("[+] sending add message ...\n"); |

6.发送热拔插消息，触发漏洞

构建NETLINK套接字

创建要热拔插文件件

构建热拔插内容，发送给内核

触发漏洞

图3.2 udev漏洞攻击流程

创建一个NETLINK的套接字，并构建热拔插的信息，里面还含有恶意代码发送给内核，由于内核没有检查消息的发送者是内核还是用户，就以root权限执行了代码，这样就获得了权限提升

3.2.4 udev漏洞的修复

只需要给system/core/init/devices.c文件打个补丁就可以了，具体内容如下：

|  |
| --- |
| static int open\_uevent\_socket(void)   {     setsockopt(s, SOL\_SOCKET, SO\_PASSCRED, &on, sizeof(on));  //在open\_uevent\_socket对套接字增加一个选项 SO\_PASSCRED，这样可以让套接字增加一个认证，让接收者可以知道发送者的uid和gid  }    void handle\_device\_fd(int fd)  {     for(;;) {         char msg[UEVENT\_MSG\_LEN+2];         char cred\_msg[CMSG\_SPACE(sizeof(struct ucred))];         struct iovec iov = {msg, sizeof(msg)};         struct sockaddr\_nl snl;         struct msghdr hdr = {&snl, sizeof(snl), &iov, 1, cred\_msg, sizeof(cred\_msg), 0};         ssize\_t n = recvmsg(fd, &hdr, 0);         if (n <= 0) {             break;         }       while((n = recv(fd, msg, UEVENT\_MSG\_LEN, 0)) > 0) {         struct uevent uevent;         if ((snl.nl\_groups != 1) || (snl.nl\_pid != 0)) {             /\* 如果不是内核的多播信息则抛弃 \*/             continue;         }         struct cmsghdr \* cmsg = CMSG\_FIRSTHDR(&hdr);         if (cmsg == NULL || cmsg->cmsg\_type != SCM\_CREDENTIALS) {             /\* 如果发送者的认证没有则抛弃 \*/             continue;         }         struct ucred \* cred = (struct ucred \*)CMSG\_DATA(cmsg);         if (cred->uid != 0) {             /\* 消息不是来自于root用户则抛弃 \*/             continue;         }  }  } |

## 3.3 adb setuid漏洞

### 3.3.1 adb setuid漏洞信息

发现时间：2010年

发现者:” The Android Exploid Crew”小组

漏洞编号：无

影响范围：android4.0以下

### 3.3.2 adb setuid漏洞原理

adb.c中的代码是以root权限运行，以完成部分初始化工作，初始化完成之后，会调用下面代码及setuid()将用户从root切换回shell,但是，setuid()在shell用户进程数达到上限1024时，会失败，而系统没有检查setuid函数的返回值，因此adb.c继续以root身份运行，没有报错

android\_src/system/core/adb/adb.c

|  |
| --- |
| /\* then switch user and group to "shell" \*/  setgid(AID\_SHELL);  setuid(AID\_SHELL);//没有检查返回值 |

|  |
| --- |
|  |
| 3.3.3 adb setuid漏洞的利用 步骤：  1、在Android的shell用户下，制造大量的僵尸进程，直至达到shell用户的进程数上限RLIMIT\_NPROC；2、kill当前系统中的adb进程，并再次占据其进程位置以保持达到上限；  3、系统会在一段时间后重启一个adb进程，该进程最初是root用户，在完成少许初始化工作后，调用setuid()切换至shell用户；  4、此时shell用户的进程数已经达到上限，所以setuid()失败，返回-1，并且用户更换没有完成，adb还是root权限；  5、adb没有检查setuid()的返回值，继续后续的工作，因此产生了一个具有root权限的adb进程，可被用于与用户的下一步交互。  搜索adb进程pid  fork一个新进程  在子进程中继续不断fork  杀掉adb进程，并占据留下的进程空位  漏洞触发  图3.3 setuid漏洞攻击流程  不断fork子进程(其实是孙进程)，然后退出，产生僵尸进程，使得当前shell用户的pid数达到上限，然后kill掉adbd进程，然后再迅速fork一个，使之仍然达到上限，这样将无法再为当前用户创建进程过段时间，init进程会检测到当前没有adbd进程，会运行adb.c，利用漏洞得到root权限的adbd进程 3.3.4 adb setuid漏洞的修复  |  | | --- | | /\* then switch user and group to "shell"  if (setgid(AID\_SHELL) != 0) {  exit(1);  }  if (setuid(AID\_SHELL) != 0) {  exit(1);} | |

增加一个判断，如果返回值不为0，就退出

## 3.4 zergRush漏洞分析

### 3.4.1 zergRush漏洞

发现时间:2011年

发现者：Revolutionary工具开发小组

编号：CVE-2011-3874

漏洞影响范围：android3.0及以下

### 3.4.2 漏洞原理

具有root权限的vold进程使用了libsysutils.so库，该库的一个函数存在栈溢出，栈变量argv[FrameworkListener::CMD\_ARGS\_MAX]由于允许的最大下标为16，如果我们特意传送超过 16 个空格分割的字符串，函数就会溢出，因此可以在root权限执行输入的shellcode

注：Shellcode实际是一段代码（也可以是填充数据），是用来发送到服务器利用特定漏洞的代码，一般可以获取权限

栈溢出原理：不顾堆栈中分配的局部[数据块](http://baike.baidu.com/view/702806.htm)大小，向该数据块写入了过多的数据，导致数据越界，结果覆盖了老的堆栈数据。 或者解释为 在长字符串中嵌入一段代码，并将过程的返回地址覆盖为这段代码的地址，这样当过程返回时，程序就转而开始执行这段自编的代码

### 3.4.3 漏洞利用

步骤：

1. 计算出vold的堆地址

2. 查到system调用的地址

3. 尝试出栈缓冲区大小

4. 通过崩溃产生的调试信息，取得栈地址和栈结构信息

5. 在libc.so中找寻跳板指令

6. 根据缓冲区大小、栈结构和上述各种地址，构造出有效的shellcode来，发送到vold

7. shellcode在vold中以root权限运行，它通过system调用运行该利用程序的一个副本boomsh

8. 程序副本boomsh以root权限运行时，会置上shell程序的S位，并设置系统属性ro.kernel.qemu

9. 结束掉adb，后续开启的adb进程将具有root权限

利用了android中几个特殊之处：

1. vold的溢出会在adb logcat中输出调试信息，这些信息说明了其内存结构，而其他程序可以读取到这些信息；

2. 在ARM架构下，跳板指令有了更多的选择，ret2libc的攻击也可能更容易实现

3. adb的降低权限过程又一次被利用。

3.5.3 zergRush漏洞的修复

在/system/core/libsysutils/src/FrameworkListener.cpp中的

1.  函数onDataAvailable()对read(c->getSocket(), buffer,sizeof(buffer))加了TEMP\_FAILURE\_RETRY宏的判断

2.  函数dispatchCommand()在给局部变量argv传值时候，对长度进行判断,若超过limit，则跳转退出。

## 3.5 psneuter漏洞分析

### 3.5.1 psneuter漏洞

发现时间：2011年

发现者：scotty2

漏洞编号：[CVE-2011-1149](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-1149)

漏洞影响范围：android2.3及以下

### 3.5.2 psneuter漏洞原理

ADB 是 Android 手机中的一个调试服务 是由 ROOT [程序](http://www.xuebuyuan.com/)派生的，

看 adb.c 中有一样一段[代码](http://www.xuebuyuan.com/)

|  |
| --- |
| property\_get("ro.kernel.qemu", value, "");      if (strcmp(value, "1") != 0) {          property\_get("ro.secure", value, "");          if (strcmp(value, "1") == 0) {              // don't run as root if ro.secure is set...              secure = 1;              // ... except we allow running as root in userdebug builds if the              // service.adb.root property has been set by the "adb root" command              property\_get("ro.debuggable", value, "");              if (strcmp(value, "1") == 0) {                  property\_get("service.adb.root", value, "");                  if (strcmp(value, "1") == 0) {                      secure = 0;                  }              }          }      } |

只要让 secure 不等于1，就是以 root 运行，否则就是以 shell 运行,secure 默认值是为 0 的，看上面代码，property\_get 是 Android 中属性服务，用的是共享内存，因为在这里没有加上合适的判断，可以让这片内存设定不可读，按上面的代码逻辑，ADB 也就成 ROOT 用户了。所以只需要找到这片内存，并将secure的值设为0

### 3.5.3 psneuter漏洞利用

步骤：

1. 查找环境变量 ANDROID\_PROPERTY\_WORKSPACE，记录的是共享内存的句柄，及内存大小
2. 调用了一个 ioctl 命令 ioctl(fd, ASHMEM\_SET\_PROT\_MASK, 0)，会导致这个共享内存不可读
3. 杀死当前的adbd进程，等待init进程重启adbd进程。
4. 具有root权限的adbd重启，触发漏洞

3.5.4 漏洞的修复

## 3.6 GingerBreak漏洞分析

### 3.6.1 GingerBreak漏洞

发现时间：2011年

发现者: “The Android Exploid Crew”小组

编号: [CVE-2011-1823](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-1823)

影响范围：android3.0及以下

### 3.6.2 GingerBreak漏洞原理

  利用了android的/system/vold/DirectVolume.cpp中handlePartitionAdded()函数的漏洞

|  |
| --- |
| void DirectVolume::handlePartitionAdded(const char \*devpath, NetlinkEvent \*evt) {  int major = atoi(evt->findParam("MAJOR"));  int minor = atoi(evt->findParam("MINOR"));  int part\_num;  const char \*tmp = evt->findParam("PARTN");  if (tmp) {  part\_num = atoi(tmp);  } else {SLOGW("Kernel block uevent missing 'PARTN'");  part\_num = 1;  }  if (part\_num > mDiskNumParts) {  mDiskNumParts = part\_num;  }  if (part\_num > MAX\_PARTITIONS) {  //没有判断part\_num的正负攻击点，如果part\_num小于1  SLOGE("Dv:partAdd: ignoring part\_num = %d (max: %d)\n", part\_num, MAX\_PARTITIONS);  } |

通过代码修改/system/bin/vold程序的GOT表项，将strcmp()、atoi()等函数的地址为system()函数的地址，然后触发调用strcmp()或atoi()来达到执行system()的目的，而后者真正被执行后会为我们来带的Root Shell，在修改完函数地址后，就要考虑如何触发，使用了NET\_LINK进行通信，通过发送热插拔消息让void中的strcmp()或atoi()被调用，但不同的Android系统版本可能操作起来有所不同，于是，需要手工构造消息。

### 3.6.4 GingerBreak漏洞的利用

步骤：

修改/system/bin/vold程序的GOT表项，将strcmp()、atoi()等函数的地址修改为system()函数的地址，代码片段

|  |
| --- |
| vold.pid= found;   vold.found= 1;   if(vold.system)     return;     ptr= find\_symbol("system");  vold.system= (uint32\_t)ptr; |

手工构造热拔插消息，不同的android版本，构造方式不一样

|  |
| --- |
| if(honeycomb) { //针对android3.0                n= snprintf(buf, sizeof(buf), "@/foo%cACTION=add%cSUBSYSTEM=block%c"                                 "SEQNUM=%s%cDEVPATH=%s%c"                                   "MAJOR=%s%cMINOR=%s%cDEVTYPE=%s%cPARTN=1",                                   0, 0, 0, bsh, 0, bsh,0, bsh, 0, bsh, 0, bsh, 0);         }else if (froyo) {//针对android2.0                n= snprintf(buf, sizeof(buf), "@/foo%cACTION=add%cSUBSYSTEM=block%c"                                  "DEVPATH=%s%c"                                "MAJOR=179%cMINOR=%d%cDEVTYPE=harder%cPARTN=1",                                   0, 0, 0, bsh, 0, 0,vold.system, 0, 0);         }else {                n= snprintf(buf, sizeof(buf), "%s;@%s%cACTION=%s%cSUBSYSTEM=%s%c"                                 "SEQNUM=%s%cDEVPATH=%s%c"                                "MAJOR=179%cMINOR=%d%cDEVTYPE=harder%cPARTN=1",                                   bsh, bsh, 0, bsh, 0,bsh, 0, bsh, 0, bsh, 0, 0, vold.system, 0, 0); |

使用NET\_LINK进行通信，使vold中的strcmp()或者atoi()被调用

### 3.6.5漏洞的修复

|  |
| --- |
| if (part\_num > MAX\_PARTITIONS || part\_num < 1) {         SLOGE("Invalid 'PARTN' value");        return;    } |

增加一个判断，当part\_num<1时，报错，这样不会出现任何调试信息

## 3.7 [mempodroid漏洞分析](http://blog.csdn.net/mldxs/article/details/14497719)

### 3.7.1 mempodroid漏洞

发现时间：2012年

发现者：zx2c4

编号：CVE-2012-0056

### 3.7.2 mempodroid漏洞原理

破解过程使用到了suid程序。suid并不是一个程序，而是可执行文件的一种属性。当你执行一个带有suid属性的程序时，在执行suid程序期间，你启动的进程的user将被临时改为suid程序的owner，进程将拥有程序owner所拥有的权限。这一特性经常用于让普通用户临时获得root权限。在linux系统中，很多功能是需要root权限才能使用的。用户如果想要用到这些功能，可以有两个办法：一是使用root用户登录。但是很可能你没有root密码。就算有，这样做也不安全（误操作是致命的）；二是执行一个owner是root的suid程序。这样就可以在不使用root用户登录的情况下，被允许使用一些需要root权限的功能，既方便又安全。

漏洞的原理是利用系统中具体s属性的程序通过自修改程序的内存，执行Shellcode达到获得Root权限的目的，完成修改进程内存的动作前需要解决两个问题：

1.  系统只允许$pid进程或者$pid的调试进程对/proc/$pid/mem文件进行写入。

2.  系统会检查打开/poc/$pid/mem的程序的self\_exec\_id是否与当前运行的程序相同，一个进程使用exec()后self\_exec\_id会自动加一，以此来保护内存不会被别的程序修改。

解决第一个问题很简单，可以直接打开自己进程的内存即可，第二个问题就难办了，因为进程打开自己时self\_exec\_id已经加一了，zx2c4使用子进程来巧妙的解决了这个问题，首先fork()子进程来保存进程的mem文件到CMSG\_DATA,然后父进程使用dup(2)创建2号fd，接着dup2(mem,2)将mem的内容dup2给2号fd，这时2号fd指向了/poc/$pid/mem的fd，下一步是构造参数args，调用"/system/bin/run-as"来执行Exploit

### 3.7.3 mempodroid漏洞利用

步骤：

直接打开自己进程的内存

首先fork()子进程来保存进程的mem文件到CMSG\_DATA,然后父进程使用dup(2)创建2号fd，接着dup2(mem,2)将mem的内容dup2给2号fd，这时2号fd指向了/poc/$pid/mem的fd

构造参数args，调用"/system/bin/run-as"来执行Exploit

### 3.7.4 mempodroid漏洞的修复

移除系统中具体s属性的程序通过自修改程序的内存的权限

## 3.8 libperf\_event漏洞分析

### 3.8.1 libperf\_event漏洞

发现时间：2013年

发现者：hiikezoe

编号：CVE-2013-2094

影响版本：android4.0及以下

### 3.8.2 libperf\_event漏洞原理

该漏洞位于[kernel/events/core.c](http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/kernel/events/core.c?id=8176cced706b5e5d15887584150764894e94e02f)文件中的perf\_swevent\_init函数。当使用syscall打开perf\_event的file descriptor时，该函数被调用。

在受影响的版本中，core.c关键部分内容如下：

|  |
| --- |
| static int perf\_swevent\_init(struct perf\_event \*event)  {  int event\_id = event->attr.config; //dangerous!  /\*  \* ....omit some code...  \*/  if (event\_id >= PERF\_COUNT\_SW\_MAX)  return -ENOENT;/\*\* .........omit........\*/  } |

由于将event\_id定义为了有符号型整数，而后面又只检查了event\_id的上界，所以当值为负时，即可越过检查，继续执行后面的代码

|  |
| --- |
| if (!event->parent) {  int err;  err = swevent\_hlist\_get(event);  if (err)  return err;  atomic\_inc(&perf\_swevent\_enabled[event\_id]);  event->destroy = sw\_perf\_event\_destroy;  } |

其中，static\_key\_slow\_inc函数将某地址内容做inc。当使用负数值的event\_id调用时，perf\_swevent\_enabled[event\_id]将指向内核空间，如，在Cent OS 64bit系统中，当event\_id = -1调用时，perf\_swevent\_enabled[event\_id]的地址为:0xfffffffffffffffe \* 4 + 0xffffffff81f360c0 == 0xFFFFFFFF81F360B8，该地址指向内核空间。

而当file descriptor关闭时，由于event\_id定义为u64型，所以关闭时做dec时，地址指向的是用户空间。所以当我们用负数值调用perf\_event\_open时，将可以在perf\_swevent\_enabled地址后面任意地址写入数据。

### 3.8.3 libperf\_event漏洞利用

但由于手机使用的ARM架构和x64架构下，linux内核代码有所不同，该漏洞的利用方式也有所不同。

x64架构下，该漏洞比较容易利用，需要的信息更少，exploit执行更快。方法简单来说是修改IDT中 int 4 的中断向量。x64的linux中，Offset hight bits和Offset middle bits是分别保存的，而将 int 4 的高位 0xffffffff 加1之后，中断向量将指向用户空间。而将shellcode放置在用户空间的这段内存上，即可获得系统最高权限。

具体来说，

1.使用-1和-2调用，得到数组的偏移值。

2.使用-1和-2调用，在file descriptor被关闭后，将使 0x38000000 附近的内存中某两个值减1，通过搜索可以确定perf\_swevent\_enabled的编译值，并保存起来方便利用。

3.将shellcode放入适当的用户空间内存中。使用 sidt 指令可以获得IDT的地址，然后使用 mmap 将 shellcode 放入适当的地址。使 IDT 中的中断向量高位+1之后落在shellcode上。

4.修改shellcode将 shellcode 中的 MARKER 修改为实际的 uid 和 gid 的保存地址。这样当shellcode运行时，即可找到 task\_struct 和 thread\_info 的地址，从而直接修改 real\_cred 值，使当前进程获得 root 权限。

5.修改 IDT 从 1 中得到的信息，和IDT 地址做计算，用计算好的负数值调用 perf\_event\_open ，可以使 int 4 的中断向量的高位 +1，从而指向shellcode。

6.int 0x4 触发shellcode

7.在shellcode中提权，并弹shell修改 real\_cred 获得root权限，并修复被修改的 IDT 和其他部分内存，

最后execl("/bin/bash", "-sh", NULL);得到root权限的shell

但是在ARM linux下，IDT的结构如下

|  |
| --- |
| struct \_idt\_entry {  unsigned short base\_lo;  unsigned short sel;  unsigned char unused;  unsigned char flags;  unsigned short base\_hi;  } \_\_attribute\_\_((packed)); |

注意到base\_hi是short型，而perf\_swevent\_enabled是int型的数组，所以无法将地址修改base\_hi到用户空间地址。

所以在ARM下我们没有修改IDT，而是在linux内核中寻找合适的函数指针。反复调用perf\_event\_open，将函数指针的值加到用户能控制的地址，然后调用函数。ptmx\_fops结构中的fsync函数指针被初始化为0，且不需要特殊的权限即可调用。

利用的基本思路是：

|  |
| --- |
| /\* file\_operations 结构fsync函数的偏移为56 \*/  int target = pmtx\_ops + 56;  int payload = -((perf\_table - target)/4)  struct perf\_event\_attr event\_attr;  event\_attr.config = payload;  ...  /\* 反复调用将fsync的值修改为合适的值 \*/  syscall(\_\_NR\_perf\_event\_open, &event\_attr, 0, -1, -1, 0);  ...  /\* 调用 \*/  int ptmx = open("/dev/ptmx", O\_RDWR);  fsync(ptmx); |

但是，linux系统中单个进程能够打开的file descriptor数量是有限制的，所以需要fork出足够多的进程，反复修改。

### 3.8.4 libperf\_event漏洞的修复

漏洞产生的原因是event\_id定义为了有符号型整数，而后面又只检查了event\_id的上界，所以当值为负时，即可越过检查，继续执行后面的代码，因此在[kernel/events/core.c](http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/kernel/events/core.c?id=8176cced706b5e5d15887584150764894e94e02f)文件中的perf\_swevent\_init函数中检查下event\_id的下界，如果为负，报错，这样就不会越过检查，执行后面的代码了

# 第四章 厂商定制系统漏洞

很多手机厂商为了突出自己的特点，对Android系统进行了定制。Samsung、Motolora、LG、ZTE、Huawei、Sony……等厂商的设备都曾出现问题 。



缺陷产生的原因大都为重要系统目录或文件的权限配置不当、自己添加的系统服务以过高的权限运行、定制的硬件驱动存在各类编码漏洞、写入文件没有考虑符号链接

## 4.1 TacoRoot漏洞

### 4.1.1 TacoRoot简介

发现时间：2011年

发现者：Justin Case/jcase

针对机型：HTC

### 4.1.2 TacoRoot原理

HTC手机中有可以恢复并且具有可写权限的日志，所以，如果将这日志删除，系统发现该日志不存在。会重启恢复一份日志，由于，日志具有可写权限，所以将日志中的ro.kernel.qemu的值设为1，然后再次重启。这个参数让手机认为，启动的是模拟器，并以root权限运行adbd。但是这个过程中，由于手机不稳定，可能造成手机无法重启

## 4.2 BurritoRoot漏洞

### 4.2.1 burroitoRoot漏洞简介

发现时间：2012年

发现者：Justin Case/jcase

针对机型：Kindle Fire(固件在6.2.1以下)

### 4.2.2 BurroitoRoot漏洞原理

在/sbin/adbd中"service.root.amazon.allow"的二进制属性，用来检测adbd是否能以root权限运行，正常情况下，它只能被系统或者具有root权限的进程所修改。在Services.jar中，有一个广播接收器com.lab126.services.EasterEggReceiver，它能够修改"service.root.amazon.allow"的二进制值，而且它能接受任何应用发出的广播。当这个值被修改，手机就被root了

## 4.3 LGPwn漏洞

### 4.3.1 LGPwn漏洞简介

发现时间：2013年

发现者：Justin Case/jcase

针对机型：LG-E971 LG Optimus G LG-E973 LG Optimus G LG-E975 LG Optimus G LG-E975K LG Optimus G等

### 4.3.2 LGPwn漏洞原理

Backup和Spritebud是由Sprite Software公司开发的应用备份/恢复系统，内置在部分LG Android智能手机上。但是Spritebud 1.3.24和Backup 2.5.4105在代码实现上存在本地权限提 升漏洞，本地攻击者可利用此漏洞获取受影响设备的root权限。  
 Spritebud后台程序是由init脚本启动并以root用户运行，以unix套接字形式监听来自"Backup"应用的指令。本地攻击者通过特制的备份，就可以写入和更改任何文件的权限和所有权。

   特制备份数据包包含exploit程序需要恢复的数据，备份数据包括一个的50MB虚拟文件a，a文件用于增加我们的exploit窗口事件，  
su二进制文件b，脚本文件c用户安装su，和一个文本文件d，d文件内容为c文件的路径。 所有文件都属于该exploit应用程序，且这些文件都是rwxr-xr-x。所有的文件以字母顺序恢 复。整个备份压缩后约2MB。在此备份中的结构如下：

|  |
| --- |
| drwxrwxrwx u0\_a114 u0\_a114 2013-05-28 20:13 files ./files:  -rwxr-xr-x u0\_a114 u0\_a114 52428800 2013-05-22 20:06 a  -rwxr-xr-x u0\_a114 u0\_a114 91992 2013-05-22 20:07 b  -rwxr-xr-x u0\_a114 u0\_a114 251 2013-05-22 20:12 c  -rwxr-xr-x u0\_a114 u0\_a114 42 2013-05-22 20:07 d |

在恢复之前，首先运行exploit应用程序运行时，观看的过程和等待。  
在恢复过程中，spritebud守护进程首先创建files目录，并设置它的权限和所有者。然后它解压缩并还原a的文件（50MB的虚拟文件）。在恢复a文件期间，我们的exploit应用程序就可以创建文件d(其中包含我们的脚本文件c的完整路径)的符号链接到/sys/kernel/uevent\_helper，。恢复文件d后，我们的路径被写入到uevent\_helper。当一个热插拔事件发生（每隔几秒钟发生一次），包含在uevent\_helper路径文件被内核执行。我们的脚本执行c执行并安装的su二进制文件b。

## 4.4 Motorola Razr ICS漏洞

### 4.4.1 Motorola Razr ICS漏洞简介

发现时间：2012年

发现者：bliss

影响设备： Motorola Razr

### 4.4.2 Motorola Razr ICS漏洞原理

Motorola Razr ICS手机存在/data/local/12m目录，该目录由init进程在设备启动时创建并chown到shell用户组，且赋予shell用户组对该目录的rwx权限。因此我们可以通过adb shell创建/data/local/12m到/data目录的软链接，这样设备重启后，init进程对/data/local/12m目录的所有操作就会转换为对/data目录的操作，shell用户组就具有了对/data目录的rwx权限，我们可以通过修改/data/local.prop配置文件来使adb shell在设备重启后以root身份运行。

  Motorola Razr ICS 手机启动过程中，init进程通过读取init.rc配置文件完成初始化任务，init.rc文件存在如下内容：

|  |
| --- |
| mkdir /data/local 0771 mot\_tcmd shell    mkdir /data/local/tmp 0771 mot\_tcmd shell    mkdir /data/local/12m 0771 mot\_tcmd shell    mkdir /data/local/12m/batch 0771 mot\_tcmd shell   chown mot\_tcmd shell /data/local     chown mot\_tcmd shell /data/local/12m    chown mot\_tcmd shell /data/local/12m/batch   chown mot\_tcmd shell /data/local/tmp |

    init.rc相关语法：

       mkdir <path> [mode] [owner] [group]  
          Create a directory at <path>, optionally with the given mode, owner, and  
          group. If not provided, the directory is created with permissions 755 and  
          owned by the root user and root group.

       chown <owner> <group> <path>  
          Change file owner and group.

    可见/data/local目录对shell可写，那我们可以利用shell在该目录下进行ln -s链接操作（操作前先备份原文件或文件夹）：

|  |
| --- |
| adb shell    mv /data/local/12m /data/local/12m.bak    ln -s /data /data/local/12m |

重启手机后系统会执行init.rc，对/data/local/12m的操作实际执行是对/data目录的操作，这样/data目录属性被修改为：

|  |
| --- |
| drwxrwx–x mot\_tcmd     shell              2013-12-20 10:58 data |

  接下来修改/data/local.prop文件进行修改：

|  |
| --- |
| rm /data/local/12m    mv /data/local/12m.bak /data/local/12m    mv /data/local.prop /data/local.prop.bak    echo “ro.sys.atvc\_allow\_all\_adb=1″ >> /data/local.prop |

   (“ro.sys.atvc\_allow\_all\_adb”是Motorola系统特有的设置，置1后重启，adb即拥有root权限。)

# 4.5 HTC One X AT&T Root漏洞

### 4.5.1 HTC One X AT&T Root漏洞简介

发现时间：2012年

发现者：Justin Case/jcase

影响设备：HTC One X AT&T 1.85固件及更早的版本上

### 4.5.2 HTC One X AT&T Root漏洞原理

在HTC One X AT&T 1.85固件及更早的版本上，预装了一款ATT Ready2Go应用，该程序以system用户运行，具有设置wifi、导入联系人、更换壁纸和安装应用程序等功能，我们可以利用此程序存在的漏洞获取设备Root权限。漏洞出现在Ready2Go安装应用程序的流程上。

Ready2Go安装应用完整程序流程如下：

1. 第一次通过Read2Go安装应用时，Read2Go建立下载目录“/data/install”，并且运行

|  |
| --- |
| chmod 777 /data/install |

赋予所有用户rwx权限。

2、当用户通过该程序安装应用程序时，Read2Go会把应用程序下载到“/data/install”目录，并且运行

|  |
| --- |
| chmod 666 /data/install/<apkfilename> |

3、下载完成后，Read2Go开始安装下载的程序，安装成功后删除下载的apk文件。

  利用该漏洞获取Root权限完整流程如下：

1. 查看“/data/install”目录是否存在，如不存在，可以通过Ready2Go下载安装任意应用程序的方式来创建该目录。

2、挑选一个已知包名的应用程序，如“com.att.android.markthespot.apk”,然后通过以下命令建立软连接到“/data/local.prop”文件。

|  |
| --- |
| *adb shell ls -s /data/local.prop /data/install/com.att.android.markthespot.apk* |

3、通过Ready2Go下载挑选的应用程序，由于 /data/install/com.att.android.markthespot.apk是/data/local.prop文件的软连接，Ready2Go对下载文件的操作的作用在了/data/local.prop文件(Read2Go具有system权限，可以操作/data/local.prop文件)，/data/local.prop就被修改为全局可读写。  
4、通过adb reboot命令打断Read2Go的安装流程，阻断Read2Go安装成功后的删除 apk操作。  
5、设备重新启动后，/data/local.prop就被遗留为全局可读写的。

|  |
| --- |
| *adb shell “echo ‘ro.kernel.qemu=1′ > /data/local.prop”*  *adb reboot* |

6、设备重新启动后，由于ro.kernel.qemu=1，所以adb不降权，这样adb shell连接设备就得到了以root身份运行的shell。  
（设备启动过程中，系统加载/data/local.prop配置文件设置系统属性，adb最初以root运行，之后调用setuid()降低权限。降权之前，会判断系统属性ro.kernel.qemu，如果该属性位1，则不降权。）

## 4.6 Rooting the Sony Tablet S

### 4.6.1 漏洞简介

发现时间：2012年

发现者：Dan Rosenburg

影响设备：Sony Tablet S

### 4.6.2 漏洞原理

索尼Tablet S的具有root运行的名为日志服务nfx\_log\_service，该服务维护了Android的logcat缓冲区中的磁盘上的备份/log目录。 此目录是组可写和组“日志”，这是授予adb shell，所以我们可以在这个目录中修改文件。这个日志服务系统使用Kernel.txt,AndroidMail.txt,androiEvent.txt和androidRadio.txt作为文件名，通过符号链接替换这些文件，它可能会导致日志服务要么创建一个包含相应的日志缓冲区中的内容的新文件，或者如果符号链接是发生在启动的时候，追加相应的日志缓冲到现有文件。 这些文件被创建为root：root 0644，所以一旦被创建，文件内容将无法被修改。

Android包管理器有一种象。 包内有数据目录在/data/data/[package]/ 。 普通包有lib在数据目录的本地库的目录。 然而，系统封装（如通过谷歌或OEM安装的，在这种情况下，索尼）预计不会有这个目录中的库。 如果有任何系统包在它的文件/data/data/[package]/lib/目录中，这些文件是由包管理器删除开机。 有趣的是，这包括下面的符号链接。 因此，如果我们能以某种方式导致系统封装的数据文件夹包含一个符号链接在/data/data/[package]/lib指向/data ，然后将/data目录将被删除开机，删除/data/local.prop ，并允许我们创建一个新的，获得root。

Android系统包含一种’run-as’的程序，使ADB shell假设被标记为“可调试”的任何应用程序的权限。 这是通过检查来确定/data/system/packages.list文件，其中包含包，它们的UID时，可调试标志，并且它们的数据的目录列表。 对于这个漏洞，可以挑选任何系统的应用 – 比如选择com.google.android.location。

可以通过创建一个符号链接到漏洞/log/AndroidRadio.txt指向/data/system/packages.list ，重新启动设备，并运行一个程序，记录packages.list对于被标记为假的应用程序行可调试具有相同的UID本系统的应用。 例如，使用：

com.pwn.me 10026 1 /data/data​​/ com.google.android.location

在这种情况下，10026是com.google.android.location的UID。 不同的设备可能会有所不同。 此时，记录服务附加此行packages.list ，可以在“运行为com.pwn.me”，在这一点上，由于有系统包的权限。 然后，可以更换系统包的lib有一个符号链接到目录/data ，重启设备，再次利用记录器来创建一个新的/data/local.prop包含行“ro.kernel.qemu = 1”，并再次重新启动，此时adb shell作为root运行，并且可以获得root权限

# 第五章 总结

开源代码大概每1000行就有一个缺陷，而android系统源码有几百万行，所以还有很多漏洞隐藏在其中。在前面我们分析了8个原生系统中产生的漏洞和6个厂商定制代码产生的漏洞，通过总结这些漏洞规律，我们可以通过这些规律来指导我们去发现其他的漏洞

## 5.1漏洞挖掘分析技术

漏洞挖掘是一个多种漏洞挖掘分析技术相互结合、共同使用和优势互补的过程。目前漏洞挖掘分析技术有多种，主要包括手工测试技术（manual testing）、Fuzzing技术、比对和二进制比对技术（Diff and BinDiff）、静态分析技术（static analysis）、动态分析技术（runtime analysis）等。

### 5.1.1 手工测试

手工测试是通过客户端或服务器访问目标服务，手工向目标程序发送特殊的数据，包括有效的和无效的输入，观察目标的状态、对各种输入的反应，根据结果来发现问题的漏洞检测技术。如zergRush漏洞就需要通过崩溃产生的调试信息，取得栈地址和栈结构信息。

### 5.1.2 Fuzzing技术

Fuzzing是一种基于缺陷注入的自动软件测试技术，它利用黑盒测试的思想，使用大量半有效的数据作为应用程序的输入，以程序是否出现异常为标志，来发现应用程序中可能存在的安全漏洞。所谓半有效的数据是指对应用程序来说，文件的必要标识部分和大部分数据是有效的，这样应用程序就会认为这是一个有效的数据，但同时该数据的其他部分是无效的，这样应用程序在处理该数据时就有可能发生错误，这种错误能够导致应用程序的崩溃或者触发相应的安全漏洞。Fuzzing技术是利用Fuzzer工具通过完全随机的或精心构造一定的输入来实现的。显示了Fuzzing技术的通用测试过程：

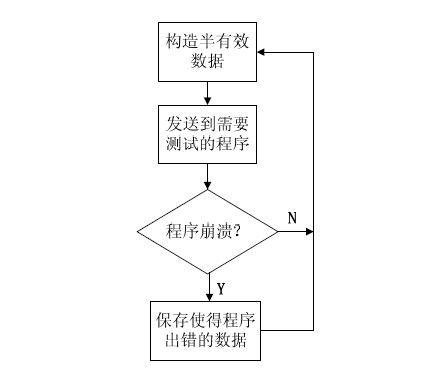


图5.1 Fuzzing测试过程

Fuzzing测试通常以大小相关的部分、字符串、标志字符串开始或结束的二进制块等为重点，使用边界值附近的值对目标进行测试。Fuzzing技术可以用于检测多种安全漏洞，包括：缓冲区溢出漏洞、整型溢出漏洞、格式化串漏洞、竞争条件漏洞、SQL注入、跨站点脚本、远程命令执行、文件系统攻击、信息泄露等。目前公布的安全漏洞中有许多都是使用Fuzzing技术检测发现的，并且有许多Fuzzing工具可以用于测试应用程序的安全性。与其它技术相比，Fuzzing技术具有思想简单，容易理解、从发现漏洞到漏洞重现容易、不存在误报的优点，当然它也具有黑盒测试的全部缺点，而且它有不通用，构造测试周期长等问题。

### 5.1.3 二进制比对技术

   二进制比对技术又可称为补丁比对技术，它主要是被用以挖掘“已知”的漏洞，因此在一定意义上也可被认为是一种漏洞分析技术。由于安全公告中一般都不指明漏洞的确切位置和成因，使得漏洞的有效利用比较困难。但漏洞一般都有相应的补丁，所以可以通过比较补丁前后的二进制文件，确定漏洞的位置和成因。

  补丁比对技术对于安全人员及黑客都是非常有用的技术，通过补丁比对分析，定位漏洞代码，再加以数据流分析，最后可以得到漏洞利用的攻击代码。有经验的安全专家或黑客甚至能在很短的时间内就完成这个漏洞挖掘与利用的过程。

补丁比对技术有很多，简单的比较方法有二进制字节比较和二进制文件反汇编后的文本比较，前者只适用于若干字节变化的比较；而后者缺乏对程序逻辑的理解，没有语义分析，适用于小文件和少量的变化。这两种方法都不适合文件修改较多的情况。较复杂的方法还有Tobb Sabin提出的基于指令相似性的图形化比较和Halvar Flake提出的结构化二进制比较，前者可以发现文件中一些非结构化的变化，如缓冲区大小的改变等，并且图形化显示比较直观。其不足是受编译器优化的影响较大，且不能在两个文件中自动发现大量比较的起始点。后者注重二进制可执行文件在结构上的变化，从而在一定程度上消除了编译器优化对分析二进制文件所带来的影响，但这种方法不能发现非结构的变化。

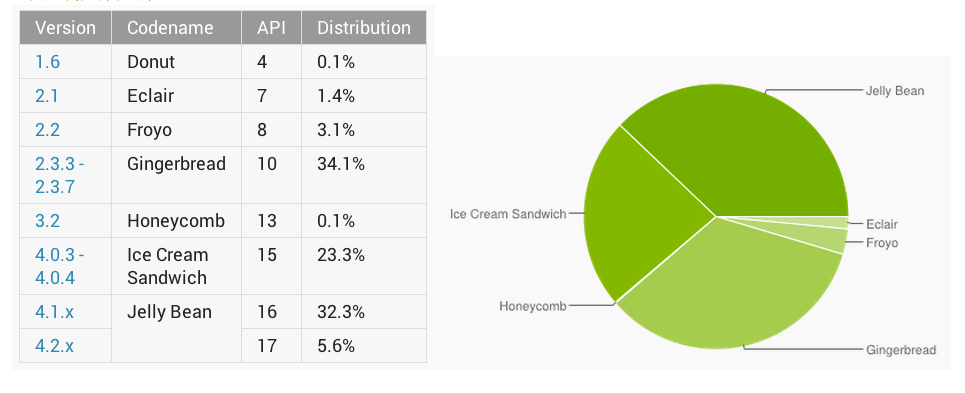


图5.2 Android版本分布

见图5.2，目前Android 4.x以上的版本总比已经超过了60%，其中4.1.x的设备占比量最高，另外之前一直占比很高的2.3版本已经从之前的50%多下降到了34%

由于Android的更新机制问题，系统的碎片化比较严重，所以通过二进制对比技术发现的漏洞，依然有使用价值。

### 5.1.4 静态分析

静态分析是通过词法、语法、语义分析检测程序中潜在的安全问题，发现安全漏洞的，其基本思想方法也是对程序源程序的静态扫描分析，故也归类为静态检测分析。静态分析重点检查函数调用及返回状态，特别是未进行边界检查或边界检查不正确的函数调用（如strcpy，strcat等可能造成缓冲区溢出的函数）、由用户提供输入的函数、在用户缓冲区进行指针运算的程序等。如adbd漏洞就是没有对进程的上限进行检查造成的提权漏洞

## 5.2 漏洞总结

sendpage漏洞就是利用具有Root权限的函数sock\_sendpage没有对socket\_file\_ops结构的sendpage字段做指针检查，将sendpage字段初始化为NULL即触发了改漏洞。

udev漏洞是由于udev对热插拔消息检测不严，恶意程序可以发送伪热插拔消息，让内核执行自身恶意代码，而内核由于没有检查校区发送是内核还是用户，就执行了恶意程序，从而取得root权限。

adb漏洞是adb.c中的代码是以root权限运行，以完成部分初始化工作，初始化完成之后，会调用下面代码及setuid()将用户从root切换回shell,但是，setuid()在shell用户进程数达到上限1024时，会失败，而系统没有检查setuid函数的返回值，因此adb.c继续以root身份运行，从而获得root权限。

zergRush具有root权限的vold进程使用了libsysutils.so库，该库的一个函数存在栈溢出，栈变量argv[FrameworkListener::CMD\_ARGS\_MAX]由于允许的最大下标为16，如果我们特意传送超过 16 个空格分割的字符串，函数就会溢出，因此可以在root权限执行输入的shellcode从而获得root权限

## 5.3漏洞发现步骤

这是几个漏洞一键Root手机软件中最常用的几个漏洞，说明这几个漏洞最常见，最好利用然而怎么发现它们了。

1. 寻找具有root权限进程所对应的代码。如udev漏洞和zerRush漏洞是利用的vold进程，adb漏洞和psneuter漏洞利用的是adbd进程。这两个进程都是具有Root权限的
2. 分析其中的函数，看是否有未进行边界检查或边界检查不正确的缓冲区溢出漏洞、整型溢出漏洞、格式化串漏洞。如sendpage就是没有对关键字段做检查，导致函数调用空指针，造成错误。adb漏洞是没有检查setuid的返回值。zergRush是利用一个函数存在栈溢出。
3. 根据漏洞类型构造恶意代码，利用该漏洞获得root权限

对于厂商定制代码产生的漏洞，基本都是通过各种手段修改/data/local.prop中“ro.kernel.qemu ”的值，将其设置为1，然后重启的adb shell作为root运行，并且可以获得root权限。

**致谢**

本论文是在张玉清教授和周文明学长的精心指导下完成的。从论文的立题到最终完成，他们都给予了极大的关怀和帮助，并提出了宝贵的意见，值此论文结束之际，我以诚挚的心情向他们表示衷心的感谢，感谢他们在这半年时间里对我的亲切关怀，热情鼓励和悉心指导

# 参考文献

[1]唐宇敬,Android平台下软件安全漏洞挖掘方法研究[D],河北工程大学,2013.

[2]肖梓航,Android平台的漏洞回顾[R]，台湾黑客年会HITCON2013,2013

[3]jduck,quine, Vuln Exploit List[EB/OL], https://github.com/droidsec/droidsec.github.io/wiki/Vuln-Exploit-List,2014

[4]