Fuzzing Android & iOS

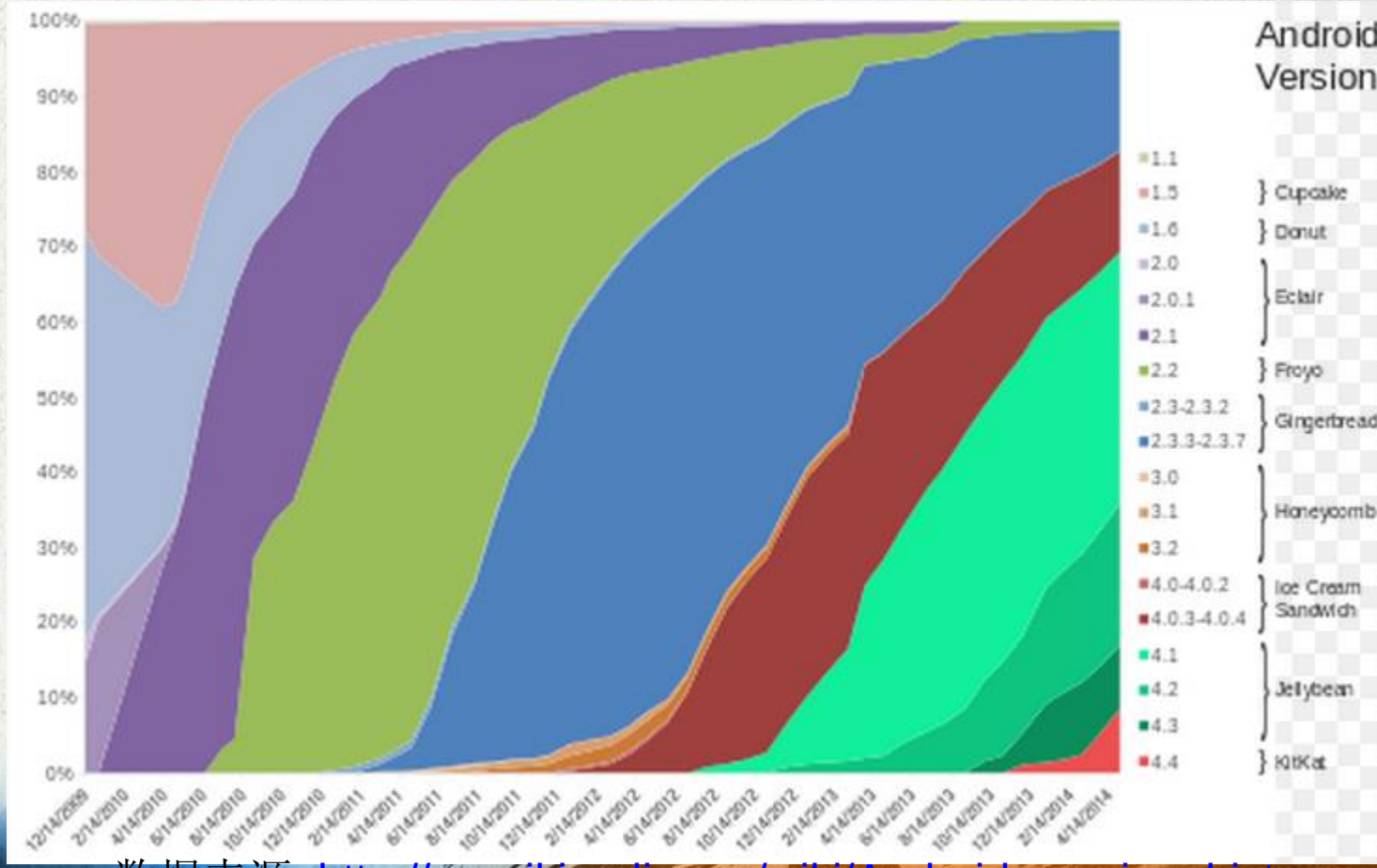
侯浩俊
李小军（OGC557）

Android终端漏洞挖掘

- 漏洞类型与现状
- 不同类型漏洞的挖掘方法
- 现有工具及自动化的局限性

Android漏洞类型与现状

- 碎片化



数据来源: http://en.wikipedia.org/wiki/Android_version_history

Android漏洞类型与现状

- 漏洞源于内核、系统、框架、应用软件

Android漏洞信息库

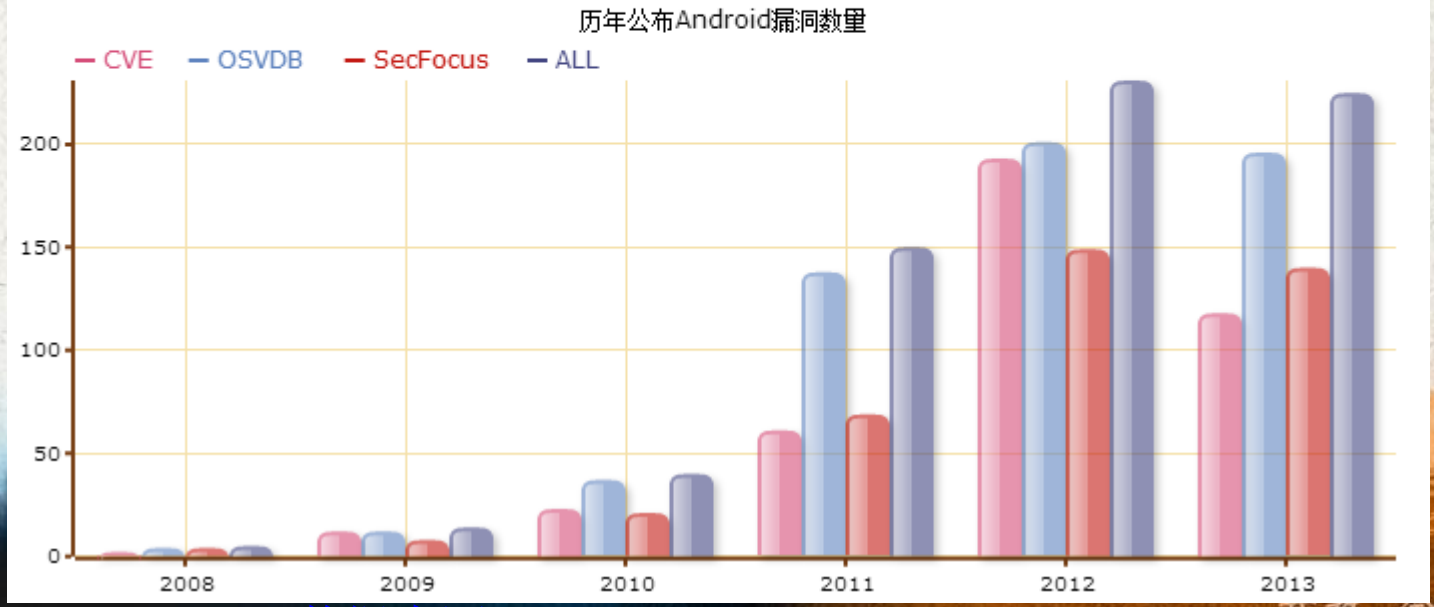
(662项)

662条漏洞信息；775条到OVAL定义的映射；330条到CWE定义的映射

105条原生漏洞；33条框架层漏洞；31条内核层漏洞

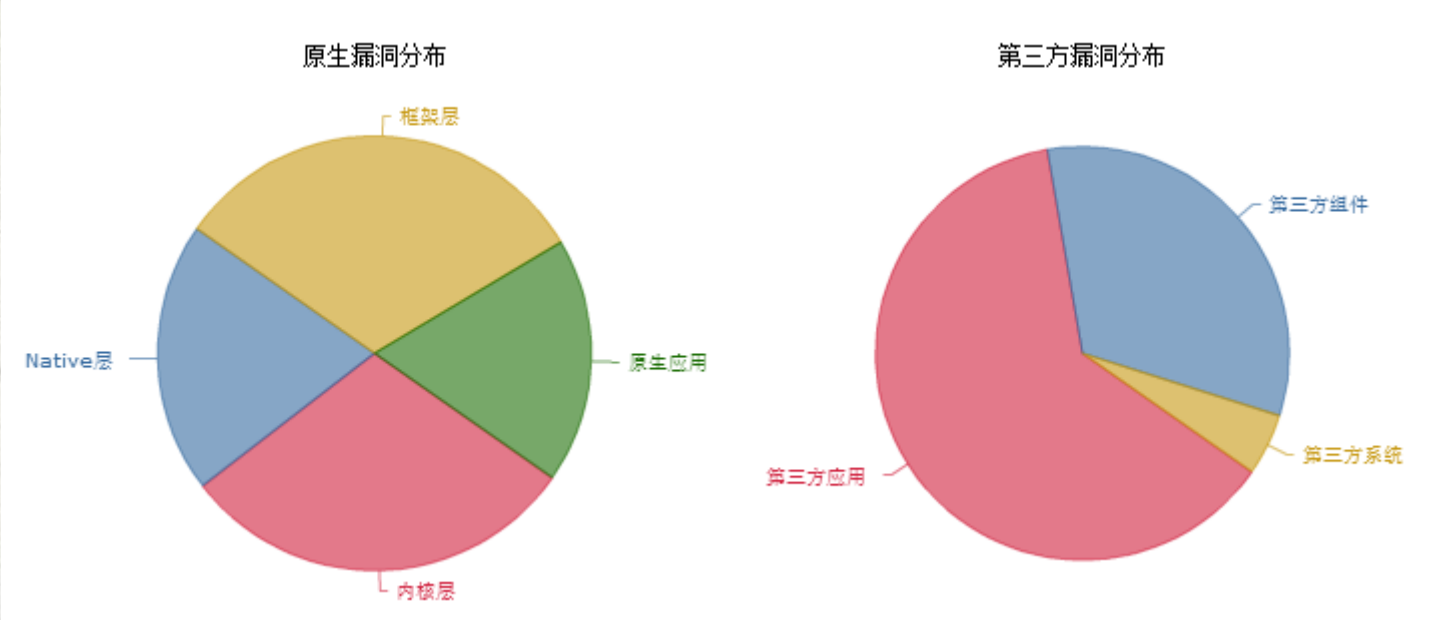
21条Native层漏洞；368条应用层漏洞；19条原生应用层漏洞

349条第三方应用漏洞；182条第三方组件漏洞；27条第三方系统漏洞



数据来源: http://android.scap.org.cn/

Android漏洞类型与现状



数据来源: http://android.scap.org.cn/

Android漏洞类型与现状

- 漏洞类型
- 漏洞种类

不同类型漏洞的挖掘方法

- 漏洞来源
- 抛砖引玉，探讨通用性

1DAY现况

- Android复用PC平台代码
- 1DAY和设备商代码成为提权漏洞的新来源
- 1day提权漏洞和移植
  - CVE-2012-0056 Linux的/proc/pid/mem文件被写入导致本地权限提升漏洞
  - CVE-2013-1773 Linux内核VFAT文件系统实现缓冲区溢出导致本地权限提升漏洞……

权限提升漏洞

- Android多个版本存在很多通过提权漏洞
  - CVE-2009-1185、CVE-2009-2692、CVE-2011-1149、CVE-2011-1823、CVE-2011-3874、setuid、 CVE-2012-0056、CVE-2012-6422、CVE-2013-1773 …

权限提升漏洞的挖掘

- 1day的移植
  - 关注Linux
  - Fuzzing Android上漏洞
- 厂商定制功能和芯片漏洞

权限提升漏洞的挖掘

- 直接测试Android内核
  - 开源部分
  - 闭源部分

eg： Fuzzing system call

Fuzzing system call

- 6个数组

```
//Data type array
unsigned char uCharArr[UCHAR_MAX+1];
signed char sCharArr[(SCHAR_MAX-SCHAR_MIN)+1];

//Random data Array
unsigned char RAN_uCharArr[50][4096];
signed char RAN_sCharArr[50][4096];
unsigned int RAN_uIntArr[1024];
signed int RAN_sIntArr[1024];
```

- 填充

```
for(loop = 0; loop < UCHAR_MAX; loop++){
    uCharArr[loop] = i;
    i++;
}
```

```
for(loop = 0; loop < (SCHAR_MAX-SCHAR_MIN); loop+
    sCharArr[loop] = i;
    i++;
}
```

Fuzzing system call

- [0][255]~[9][255]

```
for(byteLoop = 0; byteLoop < 256; byteLoop++){
    //Store a random char from 0 - 255 per byte
    RAN_uCharArr[loop][byteLoop] = uCharArr[(rand() % 256)];
    RAN_sCharArr[loop][byteLoop] = sCharArr[(rand() % 256)];
}
```

- [10][511]~[19][511]

```
for(byteLoop = 0; byteLoop < 512; byteLoop++){
    //Store a random char from 0 - 255 per byte
    RAN_uCharArr[loop][byteLoop] = uCharArr[(rand() % 256)];
    RAN_sCharArr[loop][byteLoop] = sCharArr[(rand() % 256)];
}
```

- [20][1023]~[29][1023]

```
for(byteLoop = 0; byteLoop < 1024; byteLoop++){
    //Store a random char from 0 - 255 per byte
    RAN_uCharArr[loop][byteLoop] = uCharArr[(rand() % 256)];
    RAN_sCharArr[loop][byteLoop] = sCharArr[(rand() % 256)];
}
```

- [30][2047]~[39][2047]

```
for(byteLoop = 0; byteLoop < 2048; byteLoop++){
    //Store a random char from 0 - 255 per byte
    RAN_uCharArr[loop][byteLoop] = uCharArr[(rand() % 256)];
    RAN_sCharArr[loop][byteLoop] = sCharArr[(rand() % 256)];
}
```

- [40][4095]~[49][4095]

```
for(byteLoop = 0; byteLoop < 4096; byteLoop++){
    //Store a random char from 0 - 255 per byte
    RAN_uCharArr[loop][byteLoop] = uCharArr[(rand() % 256)];
    RAN_sCharArr[loop][byteLoop] = sCharArr[(rand() % 256)];
}
```

Fuzzing system call

```
for(loop = 0; loop < 1024; loop++){
    RAN_uIntArr[loop] = rand() % UINT_MAX;
    RAN_sIntArr[loop] = rand() % INT_MAX + INT_MIN;
}
```

int mkdir(const char *pathname, mode_t mode);

- Fuzz pathname

```
for(loop = 0; loop < UCHAR_MAX; loop++){
    dir_name[0] = uCharArr[loop];
    status = mkdir(dir_name, S_IRWXU | S_IRWXG | S_IXOTH);
    if(status == -1){
        fprintf(stderr, "Status of unsigned char %c at %d: %d\n", uCharArr[loop], loop, errno);
    }else{
        //Remove dir_name when no crash
        rmdir(dir_name);
    }
}
```

Fuzzing system call

```
printf("Fuzz through all signed char data range \n");
for(loop = 0; loop < (SCHAR_MAX-SCHAR_MIN); loop++){
    dir_name[0] = sCharArr[loop];
    status = mkdir(dir_name, S_IRWXU | S_IRWXG | S_IXOTH);
    if(status == -1){
        fprintf(stderr, "Status of signed char %c at %d: %d\n", sCharArr[loop], loop, errno);
    }else{
        //Remove dir_name when no crash
        rmdir(dir_name);
    }
}
```

```
printf("Fuzz through 50 sets of random unsigned char data (256,512,1024,2048,4096) byte \n");
for(loop = 0; loop < 50; loop++){
    status = mkdir(RAN_uCharArr[loop], S_IRWXU | S_IRWXG | S_IXOTH);
    if(status == -1){
        fprintf(stderr, "Status of unsigned char data %s at %d: %d\n", RAN_uCharArr[loop], loop, err
    }else{
        //Remove dir_name when no crash
        rmdir(RAN_uCharArr[loop]);
    }
}
```

```
//Fuzz through 50 sets of random sigend char data (256,512,1024,2048,4096) byte
printf("Fuzz through 50 sets of random signed char data (256,512,1024,2048,4096) byte \n");
for(loop = 0; loop < 50; loop++){
    status = mkdir(RAN_sCharArr[loop], S_IRWXU | S_IRWXG | S_IXOTH);
    if(status == -1){
        fprintf(stderr, "Status of signed char data %s at %d: %d\n", RAN_sCharArr[loop], loop, errno
    }else{
        //Remove dir_name when no crash
        rmdir(RAN_sCharArr[loop]);
```

Fuzzing system call

- Fuzz mode

```
printf("Fuzz through 1024 randomed unsigend int data \n");
for(loop = 0; loop < 1024; loop++){
    status = mkdir(test_dir, RAN_uIntArr[loop]);
    if(status == -1){
        fprintf(stderr, "Status of unsigned int %d : %d\n", RAN_uIntArr[loop], errno);
    }else{
        //Remove dir_name when no crash
        rmdir(test_dir);
    }
}
```

```
//Fuzz through 1024 randomed signed int data
printf("Fuzz through 1024 randomed sigend int data \n");
for(loop = 0; loop < 1024; loop++){
    status = mkdir(test_dir, RAN_sIntArr[loop]);
    if(status == -1){
        fprintf(stderr, "Status of unsigned int %d : %d\n", RAN_sIntArr[loop], errno);
    }else{
        //Remove dir_name when no crash
        rmdir(test_dir);
    }
}
```

逻辑缺陷的提权漏洞

- 个例差异大

- 需要对Android的功能多做尝试，尽可能使用非系统预定的途径完成操作

远程代码执行漏洞

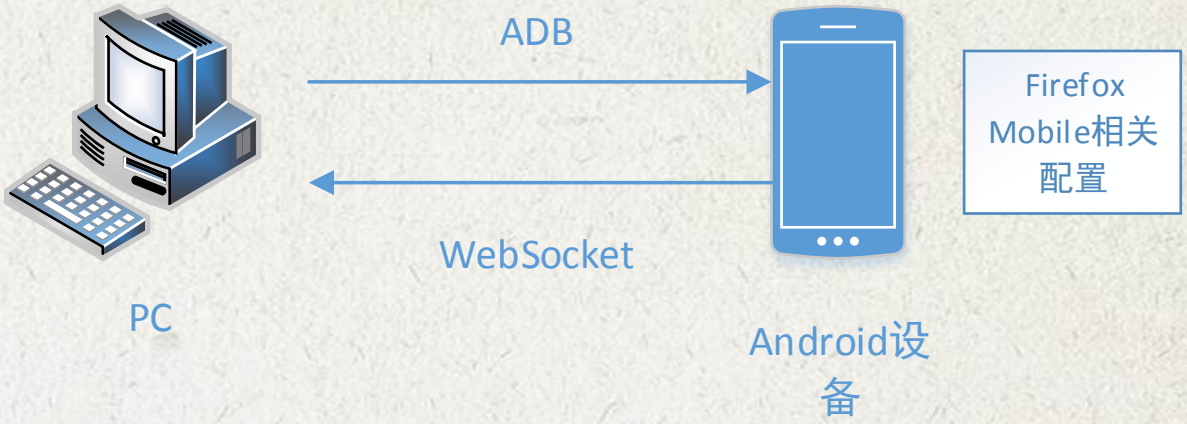
- CVE-2010-1807浮点数漏洞, CVE-2010-1119 Android 2.0/2.1 Webkit Use-After-Free
- Webview的JavaScript接口的代码注入
- Adobe Flash 漏洞移植到终端

远程代码执行漏洞挖掘

- 1day漏洞移植
  - eg：CVE-2010-1119在大量版本触发
- 挖掘移动端的浏览器缺陷

ADBFuzzer

- Target： Firefox Mobile on Android



ADBFuzzer

- Firefox Mobile相关配置

- ADB（pc-> android device）

ADBFuzzer

- WebSocket（android device->pc）
  - Send tests
  - Logs/messages
  - Send commands

ADBFuzzer

- ADB命令
  - mozdevice, DeviceManagerADB
- 关于dump
  - Firefox Mobile支持minidump
  - 分析：minidump_stackwalk、addr2line

BrowserFuzzer

- Target： webkit within the Chrome for Android

- HTML5的新特性
  - Eg:类型数组(Typed Array)

BrowserFuzzer

- Trigger code

```
var arr1 = new Array(0x24924925)
var arr2 = new Float64Array(arr1)
...
```

- Typed Array：
  - Int8Array、Uinit8Array、Int16Array、Uint16Array、Int32Array、Uint32Array、Float32Array、Float64Array

BrowserFuzzer

- Fuzzing

```
def generate_var():
    vtype = random.choce(TYPEDARRAY_TYPES)
    vlen = rand_num()
    return "var arr1 = new %s(%d);" % (vtype, vlen)
```

```
def generate_assignmen():
    vtype = random.choice(TYPEDARRAY_TYPES)
    return "var arr2 = new %s(arr1);" % (vtype)
```

BrowserFuzzer

- Pc端
  - Web服务
  - 构造js样本
  - 通过adb 向目标发起http访问请求
  - 分析结果
    - Logcat
    - Debuggerd

      -- find‘SIGSEGV’

Android组件的Fuzzing

- Activity、Service、Broadcast Receiver、Cotent Provider
- 额外权限
- 能力泄露
- …

权限策略，权限检查

Intent fuzzing

- Intent构造
  - Action
    - 隐式： AndroidManifest.xml , <intent-filter>...</intent-filter>
    - 显式：区分不同的发送者
      const strings —> 标识action → 潜在的Actions
  - Data：
    - 隐式：<intent-filter>、URL
      根据Action的类型， Fuzz uri(tel:、http:、mailto:、content:等)
  - Extras
    - putExtras(key,value)、getXXXXExtras()
    - ASOP, Hook关键API； 闭源ROMs,字节码插桩

Intent fuzzing

- Pc端： Client
  - 构造Intent
  - 处理Server的反馈信息，进一步Fuzzing
  - 分析结果

- 移动端： Server
  - 向组件发送Intent
  - 收集中间结果（hook关键API的结果）

Intent fuzzing

- hook关键API

  getIntExtras(…)、getStringExtras(…)、getShortExtra(…)、getShortArrayExtra(…)…

  ActivityManagerService

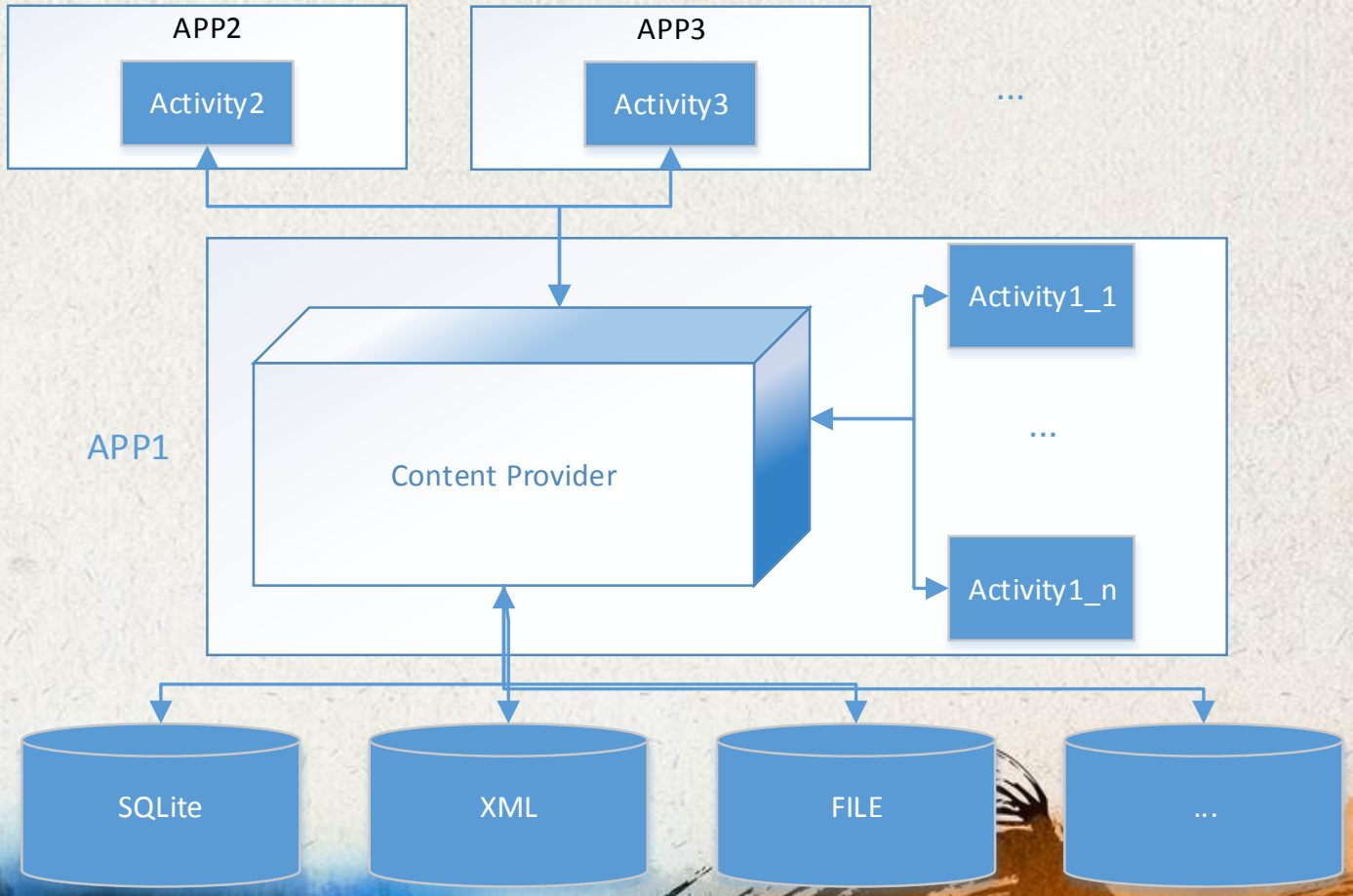
  checkPermission(String permission,int pid,int uid)

```
public int checkPermission(String permission, int pid, int uid) {
    if (permission == null) {
        return PackageManager.PERMISSION_DENIED;
    }
    return checkComponentPermission(permission, pid, UserHandle.getAppId(uid), -1, true);
}
```

Content Provider Fuzzing

- 任然是权限设置不当
  - 用户隐私泄露
    - 未检查访问者权限
    - 结合SQLite，接口参数输入不当导致SQL注入或路径遍历
  - 数据被污染
    - 应用数据或被配置被篡改，导致不可预期后果。

Content Provider Fuzzing



Content Provider Fuzzing

- Fuzzing方法：
  - Provider是否对外暴露
  - URI接口是否存在SQL注入、路径遍历
  - SQLite
    - 获取表结构
    - 污染数据
    - Hook SQLite APIs 监控
    - 触发数据库操作

Content Provider Fuzzing

- 思路类似Fuzzing Intent
- pc端：Client
- 移动端：Server

其它类型漏洞的挖掘

- 短信导致系统DOS
  - Fuzzing the Phone in your Phone
- 蓝牙通信栈的测试
  - Fuzzing Bluetooth:Crash-testing bluetooth-enabled devices
- GSM通信模块的测试
  - MobiDeke: Fuzzing the GSM Protocol Stack
- NFC通信模块的测试
  - Vulnerability Analysis and Attacks on NFC-enabled Mobile Phones

现有工具

- 学术界半自动化挖掘
  - ComDroid、CHEX、DroidChecker、Woodpecker、MalloDroid、ContentScope、…
- 开源挖掘工具
  - Drozer、Intent Sniffer、Intent Fuzzer、ASEF、AFE、…

自动化挖掘的局限性

- 代码和框架
- UI交互
- 业务关联性

NEXT

iOS 终端漏洞挖掘

iOS 终端漏洞挖掘

- 漏洞类型与现状
- iOS平台漏洞挖掘

漏洞类型与现状

- 与Android系统类似，都运行在ARM架构的终端上，源自OSX，与Linux有很多共性
- 完全闭源的操作系统，其研究门槛较Android高很多，目前iOS的304个漏洞中Webkit占据主要成分。
- 与Android不同，绝大部分漏洞没有公开的漏洞利用代码

漏洞类型与现状

- 漏洞包括：信息泄露，内存破坏，缓冲区溢出，沙盒逃逸，权限提升…
- 可以测试的对象包括：Safari浏览器，Appstore，短信，蓝牙，无线通信，内核
- 同样存在1day的移植问题，及通过挖掘Mac OSX系统下的漏洞，PC端的Safari漏洞等，然后移植测试用例尝试在iOS设备重现

漏洞类型与现状

- 重点讨论仅在iOS端触发的高危漏洞（远程代码执行，内核权限提升），需要已经越狱的iOS设备搭建挖掘平台
- 如屏幕锁绕过，浏览器的XSS等漏洞不在讨论范围
- 虽然闭源，但由于设备总类精简，所以相关调试分析手段也很多

Safari浏览器代码执行漏洞

- 由于Safari相比用户开发的程序而言，权限较高，且其漏洞可以远程触发，所以是漏洞挖掘的重点对象
- 这些漏洞源于Webkit解析HTML，不同文件格式时（PDF，JPEG，PPT，XLS）导致缓冲区溢出等

Safari浏览器代码执行漏洞

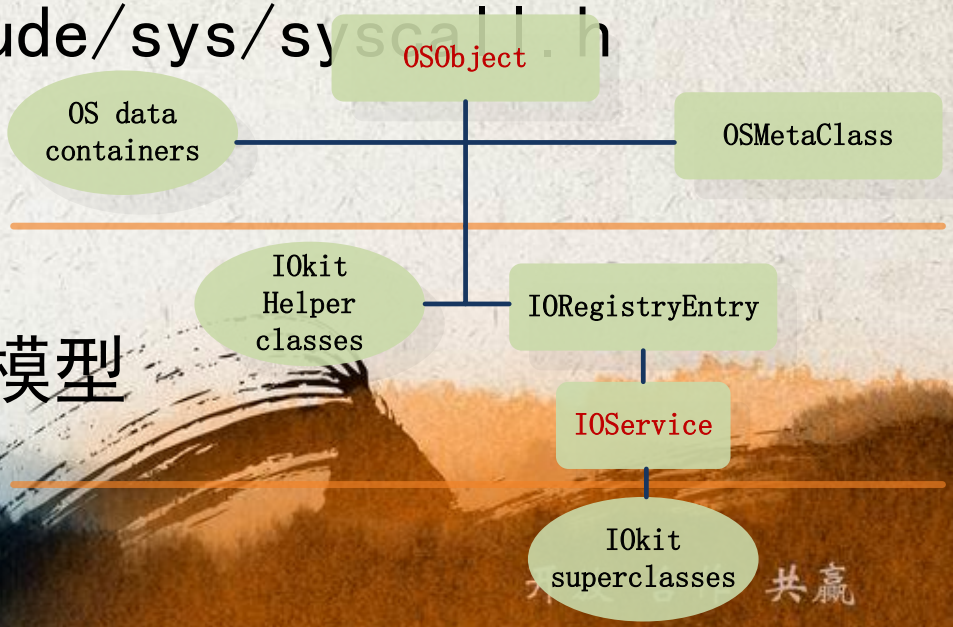
- 挖掘方法
  - 问wushi

iOS内核

- iOS内核基础
- OSX早于iOS，iOS应该是在OSX的基础上开发出来的，OSX内核从BSD内核基础上开发出来(XNU开源)
  - Mach，底层抽象Low level abstraction of kernel
  - BSD，高层抽象High level abstraction of kernel
  - IOKit，Apple内核扩展基础框架

iOS内核

- BSD
  - 实现 File System, Socket等
  - 导出的POSIX API处于用户空间和内核之间的基本接口
  - 调用编号 /usr/include/sys/sy
- IOKit
  - 内核扩展
  - 基于面向对象的编程模型



iOS内核

- Mach API

InP->Head.msgh_request_port = host;

- InP->Head.msgh_reply_port = mig_get_reply_port();
- InP->Head.msgh_id = 200;
- msg_result = mach_msg(&InP->Head, MACH_SEND_MSG|MACH_RCV_MSG|MACH_MSG_OPTION_NONE, (mach_msg_size_t)sizeof(Request), (mach_msg_size_t)sizeof(Reply), InP->Head.msgh_reply_port, MACH_MSG_TIMEOUT_NONE, MACH_PORT_NULL);

iOS内核漏洞

- 挖掘方法
  - 读XNU代码
  - 逆向
  - fuzz

Fuzz iokit

- 新手怎么入门
  - 从osx下手
  - IOServiceMatching(“IOReportHub”);
  - IOServiceOpen(service, mach_task_self(), connnection_type, &connection);
  - selector=3;
  - input[0]=0x4444;
  - IOConnectCallMethod(connection, selector,input, 1, 0, 0, NULL,NULL, NULL, NULL);
  - CVE-2014-1355和多个还没出补丁的

Fuzz iokit

- 如何深入
  - Ioreg 找所有模块
  - Fuzz 各种参数
  - IOConnectCallMethod(
  - mach_port_t connection, // In
  - uint32_t selector, // In
  - const uint64_t *input, // In
  - uint32_t inputCnt, // In
  - const void *inputStruct, // In
  - size_t inputStructCnt, // In
  - uint64_t *output, // Out
  - uint32_t *outputCnt, // In/Out
  - void *outputStruct, // Out
  - size_t *outputStructCnt) // In/Out

其他Fuzz 方向

- Syscall
- Ioctl
- Mach api
- 等等
- **Ole Henry Halvorsen& Douglas Clarke** OS X and iOS Kernel Programming

无贼

- 漏洞的用处
  - 卖钱-树人se
  - 越狱-pangu
  - CVE

天下无贼
2014电子商务安全技术峰会

请支持

开放 合作 共赢

END

Q&A 谢 谢！

开放 合作 共赢