反思：移动平台应用软件行为管控机制

Rethinking Behavior Control on Mobile Systems

张源

复旦大学系统软件与安全实验室

个人/团队介绍

- 复旦-系统软件与安全实验室
  - 负责人：杨珉 教授（国家973项目首席科学家）
  - 主要研究兴趣：移动安全
- 主讲人：
  - 张源　博士、实验室副主任
  - 研究方向：操作系统、编译和运行时环境、程序分析技术
  - 负责 *复旦大学移动应用安全攻防团队*

平安银行　陆金所　平安好医生　翼支付　500彩票　银多资本　借贷宝　平安WIFI

我们做些什么？

2011年

安卓系统研发及性能优化研究
Java语言编译优化及动静态分析研究

用户隐私保护

- 2012年，在国内首次披露安卓应用中的隐私泄露问题，发表于《瞭望》
- 2013年、2014年协助央视3.15晚会检测安卓应用的隐私问题

恶意软件对抗

- 2013年，被顶级安全会议CCS录用两篇关于安卓恶意软件的论文
- 2015年，获准立项973项目，专注于移动恶意软件的检测和控制理论研究

软件漏洞与缺陷

- 移动应用软件渗透测试体系
- 针对安卓APP的高效率高精度分析工具

本次报告内容

- 行为管控机制介绍
- 行为管控机制反思
- 一些想法
- FineDroid系统

移动平台软件行为管控机制

- 软件行为管控
  - **目的**：安全性
  - 管理程序的执行、控制程序的行为

移动平台软件行为管控机制

- 传统行为管控机制
  - **针对某个(些)程序**，从程序外部控制其能操作的资源
  - 例如：SELinux、iptables
- 规则样例
  - 允许init类型对unlabeled类型的filesystem进行mount的操作

```
allow init unlabeled:filesystem mount;
```

  - 允许user_t对bin_t类型的file进行除了write setattr ioctl相关的操作

```
allow user_t bin_t:file ~{write setattr ioctl};
```

  - 在WIFI下丢弃uid为10042的数据包

```
iptables -A OUTPUT -o wlan+ -m owner --uid-owner 10042 -j DROP
```

移动平台软件行为管控机制

- 移动应用软件新特征
  - 重交互：移动应用程序处于复杂的交互执行环境中
  - 模块化：移动应用程序内部含有多个利益体的代码

我们看到的问题

- **程序交互中的攻击**
  - 处于交互中的应用程序是否值得相信？
- 一些案例
  - 案例一: 系统预制应用中存在众多权限泄露漏洞，可导致系统重启、程序崩溃、强制关机 [CCS 2013]
  - 案例二: 应用中的数据存储接口被利用，导致用户私密信息泄露、核心业务数据被修改 [NDSS 2013]
  - 案例三: WebView中的url注入攻击

我们看到的问题

- **嵌入代码中的攻击**
  - 打包在同一个应用程序的代码是否都可以相信？
- 程序内部的SDK可以做什么
  - 可以获得APP的哪些敏感数据？
    - by Pluto [NDSS 2016]
    - 通过分析医疗、教育、健康、生活类应用的私有目录下数据
    - 发现APP中的SDK可以读取到用户的性别、电话、年龄、疾病等信息
  - 是否可以攻击程序逻辑？
    - Hook程序代码
    - Hook框架层代码

反思

- **现有软件构造流程是否是一种倒退？**
  - 从软件工程的角度来看
    - 基于组件开发、不重复造轮子、提高复用率
  - 从项目开发的角度来看
    - Time-to-release、保障上线时间、快速出原型
  - 从产品理念的角度来看
    - 业务聚焦、领域细分、单点突破
  - 生产力决定生成关系
    - 移动平台降低了开发者门槛、竞争白热化
    - 移动平台为了**更好的连接** => 移动平台产品的连接
    - 代表着一种更强的生产力：具有各自利益的主体进行协作

反思

- **为何现有行为管控机制无法保护移动APP的安全？**
  - 现有安卓行为管控机制
    - 基于UID隔离各个应用程序
    - 基于UID控制每一个程序可以使用的权限/资源
  - 问题分析
    - **应用程序交互不敏感（Inter-application In-sensitive）**

```
String url = intent.getStringExtra("load_url");
webview.loadUrl(url);
```

怎么判断url来自于开发者还是外部攻击者?

    - **应用程序内部不敏感（Intra-application In-sensitive）**

```
open("/data/data/com.XX/shared_prefs/..")
```

怎么判断访问请求来自于APP还是SDK?

一些相关工作

- 孤立的解决问题
  - IPC Inspection [USENIX Security 2011]
    - 基于进程调用栈自省技术限制程序在交互状态下能够使用的权限集合
  - TrustDroid, XManDroid [NDSS 2012]
    - 基于预定义规则阻止有风险的程序交互
  - Compac [CODASPY 2014]
    - 跟踪程序内部不同package之间的行为交互，限制不同package能够使用的权限
  - FlexDroid [NDSS 2016]
    - 基于跨进程的程序内栈自省技术识别程序内部执行状态，限制程序SDK能够使用的权限

我们的想法：通用化的精细行为管控

- 构造适用于移动平台的新型软件行为管控模型
  - 之前的做法：将行为能力绑定在特定的程序(UID)上
    - 普通API：所有APP都可以调用，例如动态加载、数据库访问
    - 敏感API：APP被授予相应权限，例如访问网络、发送短信
    - 二元组：*<app, behavior>*
  - **我们的思路**
    - **三元组：*<app, context, behavior>***
    - 描述一个APP在处于什么样的上下文中可以做什么
  - Context是什么?
    - 地理位置？设备充电中？设备连接WIFI？
    - android.content.Context？

我们的想法：通用化的精细行为管控

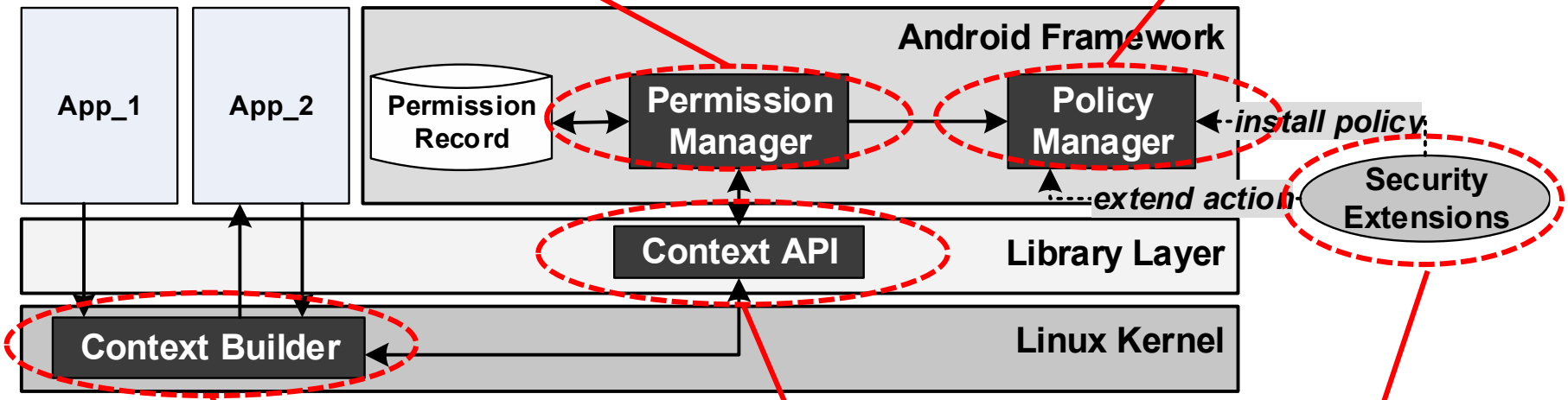
- System-wide Application Execution Context
  - 应用程序交互上下文
    - 刻画程序处于的复杂外部执行流
  - 应用程序内部上下文
    - 刻画程序内部精确的执行流

- Application-context-as-a-Service
  - 系统内建的功能，跟踪程序执行上下文
  - 将程序执行上下文作为一种系统服务
    - 应用程序可以根据自己的上下文审查函数的调用
    - 系统可以根据程序的上下文审查API的调用

FineDroid系统设计

- **以权限为例**，对程序权限使用行为进行管控
  - 根据应用程序上下文，对程序权限使用行为进行管控
  - 区分程序在不同上下文下的权限使用行为
- Application-context-as-a-Service
  - 专有API用于获取当前执行流所处于的程序上下文
  - **目标：**跟踪程序交互，识别程序内部执行流
- Policy-driven的方式管控程序权限使用行为
  - Out-of-the-box，行为管控逻辑独立于APP的实现
  - Easy-to-update，规则即时下发，易扩展

系统架构



在系统层次拦截所有的权限使用行为，包括内核级和框架级

Policy-driven的方式管控权限使用行为

跟踪应用程序上下文，伴随程序的执行，对程序透明

支持application-context-as-a-service

基于policy扩展管控逻辑

Context Builder模块

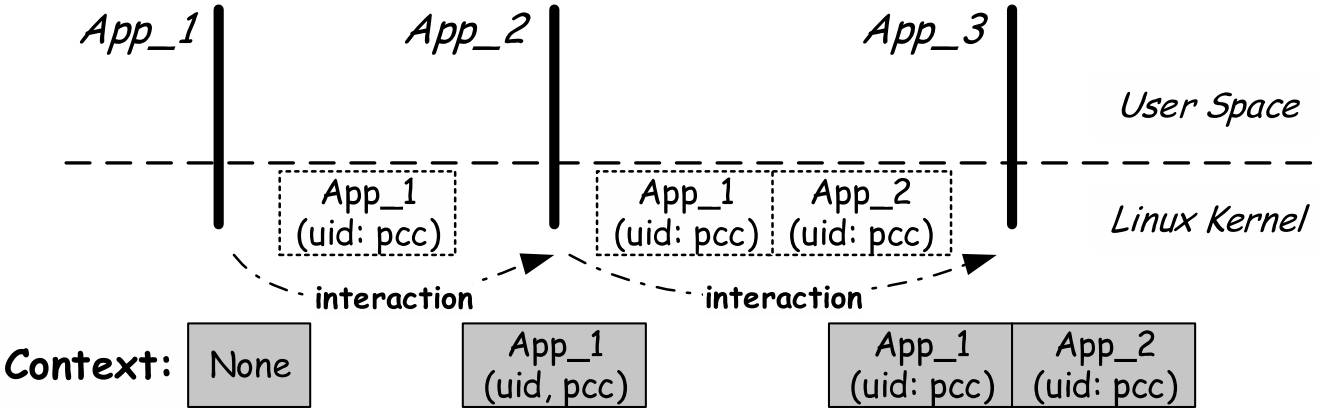
- 应用程序**内部**上下文
  - 如何标记程序内部的执行状态？
    - 使用Calling Stack来标示内部执行流
  - 如何进行有效的标记？
    - Stack Hash
    - 存储的效率、传播的效率、确定性
  - Probabilistic Calling Context (PCC)
    - 用一个32/64位的整数表示一个Stack
    - 通过递归的方式在stack unwind过程中计算

$$\boldsymbol{pcc} = \mathbf{3} * \boldsymbol{pcc}' + \boldsymbol{cs}, where\ \boldsymbol{pcc}'\ is\ PCC\ value\ of\ caller\ and\ \boldsymbol{cs}\ is\ the\ birthmark\ of\ call\ site$$

    - cs值，Call-site birthmark: 指令在DEX文件中的偏移

Context Builder模块

- 应用程序**交互**上下文
  - 基于Binder的进程交互机制，支持对调用者进程的跟踪
  - 无法回溯整个调用链上的所有进程
- 扩展Binder协议
  - 维护完整的进程间交互信息
  - 每一个进程需要记录的信息：UID+PCC



Context Builder模块

- 应用程序上下文**传播**
  - Binder级的跟踪只能维护进程间*直接调用*时的上下文
  - 如何在***间接的程序交互***下维护应用程序上下文？
- 有哪些间接的执行流？
  - 组件交互
    - 组件交互都通过AMS路由，跨程序的组件无法直接调用
  - 程序内线程交互
    - 程序内线程交互可能不通过Binder进行
  - 事件交互
    - Callback执行的上下文与Callback注册时的上下文不同

Context API模块

- 获取当前执行流处于的应用程序上下文信息
  - 调用链上进程的基本信息
    - 存储于内核之中，由Binder驱动维护
    - static final native int[] getUidChain()
    - static final native int[] getPccChain()
  - 程序内部上下文详细信息，包含所有stack上函数信息
    - static final PccInfo getPccSig(int uid, int pcc)
    - Stack上的每一个函数的签名信息抽象为一个四元组
    - *<class_name, method_name, method_sig, callsite_offset>*

Permission Manager模块

- 目标：在所有的权限使用点进行拦截，汇总到统一的一个点进行管控
- 两种权限
  - 框架级权限
    - 依赖于PackageManagerService，根据程序安装时用户授予的权限进行管控
    - **处理方法**：调用Policy Manager模块路由权限决策
  - 内核级权限
    - 依赖于Linux内核的UID/GID机制进行保护，例如网络、文件系统
    - **处理方法**：识别特定的UID/GID，通过进程间交互将最终决策交给Policy Manager

Policy Manager模块

- 输入：app, app_context, permission
- 输出：deny/grant/...
- 规则引擎

policy := <*action*> <*app*> <*permission*> <*context*>
action := *grant* | *deny* | ...

  - 基于XML表述和存储
  - 规则匹配：选择一条最能准确表述当前应用程序上下文的规则，通过cache提高匹配效率
  - 扩展性：action可以扩展，规则可以运行时增删改

Policy Manager模块

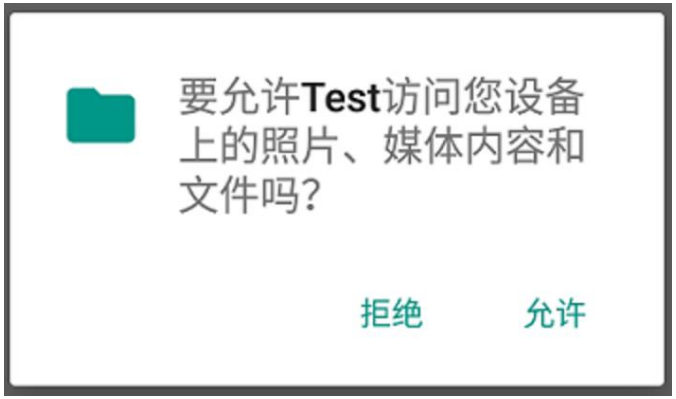
- 一个例子

```
<policy action="deny" app="com.android.mms" permission="SEND_SMS" >
  <uid-selector selector="strictcontains" >
    <uid-context uid="^com.android.mms" pcc="*" />
    <uid-context uid="com.android.mms" />
      <pcc-selector selector="contains" >
        <method-sig className="com.android.mms.transaction.SmsReceiver"
                    methodName="beginStartingService" />
      </pcc-selector>
    </uid-context>
  </uid-selector>
</policy>
```

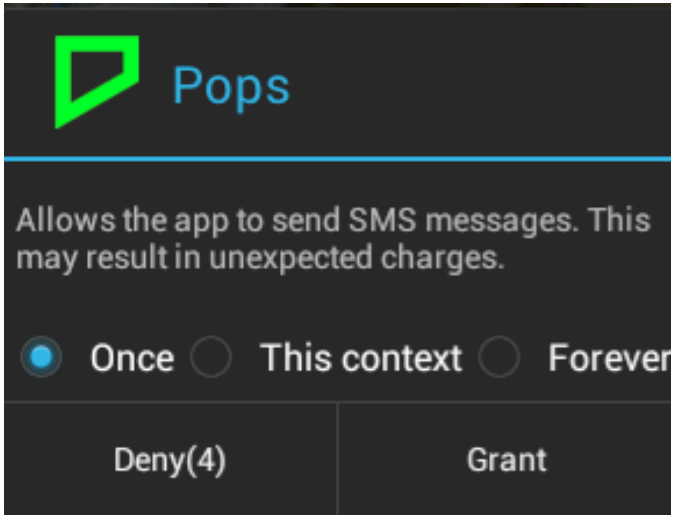
XML Tag	用途
uid-selector	表述一个程序交互链的特征
uid-context	表述一个程序上下文的特征
pcc-selector	表述一个程序内部函数调用链的特征
method-sig	表述一个程序内部函数调用的特征

Security Extensions模块

- 案例一：APP权限管理
  - 权限动态管控



  - 权限即时管控



Security Extensions模块

- 案例二：权限泄露漏洞修复
  - 权限泄露漏洞
    - APP_A存在一个供外部调用的**接口X**
    - APP_A的敏感行为能力通过**X**被APP_B使用，导致能力泄露

权限泄露漏洞的一个例子

```
public class SmsReceiverService extends Service {
    public int onStartCommand(Intent intent, int flags, int startId) {
        ... ...
      Message msg = mServiceHandler.obtainMessage();
      msg.arg1 = startId;
      msg.obj = intent;
      mServiceHandler.sendMessage(msg);
        ... ...
    }

  private final class ServiceHandler extends Handler {
    public void handleMessage(Message msg) {
        Intent intent = (Intent)msg.obj;
        if (intent != null) {
            String action = intent.getAction();
                ... ...
            } else if (SMS_RECEIVED_ACTION.equals(action)) {
                handleSmsReceived(intent, error);
            }
                ... ...
        }
    }
  }
  private void handleSmsReceived(Intent intent, int error) {
    SmsMessage[] msgs = Intents.getMessagesFromIntent(intent);
    String format = intent.getStringExtra("format");
    Uri messageUri = insertMessage(this, msgs, error, format);
    if (messageUri != null) {
        long threadId = MessagingNotification.getSmsThreadId(this, messageUri);
        MessagingNotification.blockingUpdateNewMessageIndicator(this, threadId, false);
    }
  }
}
```

AOSP中Mms程序存在的WRITE_SMS权限泄露漏洞

基于规则的漏洞修复

- 如何基于FineDroid修复权限泄露漏洞？
  - 基于程序间交互上下文，可以有效区分一个当前的执行流是来自于**内部程序**还是**外部程序**
  - 基于程序内部上下文，可以精确识别当前执行流**是否处于漏洞被利用的路径**上
  - 例子：修复SmsReceiver中的 SEND_SMS 权限泄露

```
<policy action="deny" app="com.android.mms" permission="SEND_SMS" >
  <uid-selector selector="strictcontains" >
    <uid-context uid="^com.android.mms" pcc="*" />
    <uid-context uid="com.android.mms" />
      <pcc-selector selector="contains" >
        <method-sig className="com.android.mms.transaction.SmsReceiver"
                    methodName="beginStartingService" />
      </pcc-selector>
    </uid-context>
  </uid-selector>
</policy>
```

修复规则的自动生成

- 基于漏洞检测工具，自动生成修复规则
  - CHEX [CCS 2012]
  - 对检测日志分析，提取函数调用路径和组件信息

应用名	泄露路径数	生成规则数
com.gmail.traveldevel.android.vlc.app-131	2	2
com.froogloid.kring.google.zxing.client.android-67	24	24
de.cellular.tagesschau-5	361	361
com.akbur.mathsworkout-92	2	2
com.appspot.swisscodemonkeys.paintfx-4	2	2
com.androidfu.torrents-26	1	1
com.espn.score_center-141	6	6
com.espn.score_center-142	6	6
fr.pb.tvflash-9	2	2
hu.tagsoft.ttorrent.lite-15	8	8
总计	414	414

Application-context-as-a-service

- 应用程序上下文的应用场景
  - 当前Component是被外部程序调用还是内部程序调用？
  - 我自身的一些私有接口会不会被外部程序调用？
  - 我集成的SDK会不会发短信？会不会读取我的私有文件？
  - 我集成的SDK会不会搜集我的用户的信息？
  - 我集成的SDK有没有可能攻击我？
  - 等

Take-away

- 传统软件行为管控机制在移动平台的挑战
  - 应用程序交互不敏感
  - 应用程序内部不敏感
- Application-context-as-a-service
  - 抽象程序执行流上下文信息，并且作为系统内建接口
  - 根据程序执行流的上下文进行行为管控
- Policy-driven的行为管控框架
  - Out-of-the-box，行为管控逻辑独立于APP的实现
  - Easy-to-update，规则即时下发，易扩展

Q&A