安卓软件漏洞修复与检测技术研究

张源 博士
复旦大学系统软件与安全实验室

软件漏洞

- 攻击目的：privilege
  - 突破沙箱、访问控制等防护机制
  - Know something you are not allowed to know
  - Do something you are not allowed to do
- Code Injection/Reuse Attack?
  - 使用受害者的身份运行攻击者指定的代码逻辑
  - 二进制：利用内存读写漏洞
    - Buffer overflow, Format string, User-after-free, Information disclosure, etc
  - Web: Cross Site Scripting, SQL Injection, etc
- Insecure privileged interface
  - 利用没有保护好的具有特权的接口
  - e.g Cross Site Request Forgery

安卓软件漏洞

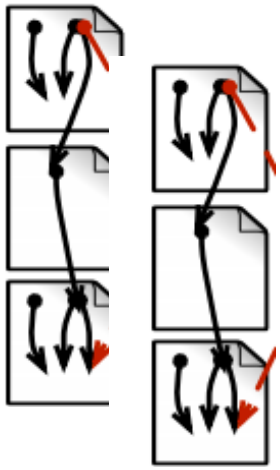
- Code Injection/Reuse is Hard!
  - Java is sandboxed, memory-safe
  - Android app is not a server
    - TRUE until wormhole?
  - Limited injection/reuse attack cases
    - Java代码加载漏洞
    - WebView导致js代码注入
- Capability Leak (Privilege Escalation)
  - 权限泄露漏洞
    - Confused deputy
    - Component hijacking
  - Content Provider漏洞
- Bad security practice
  - Insecure SSL
  - Insecure Crypto
  - Vulnerability? or Risk?

主题：漏洞防护

代码分析
漏洞特征描述

漏洞检测

漏洞特征报告



根据漏洞细节修复漏洞

漏洞修复

如何修复漏洞？

- 人工修复
  - 理解漏洞？
  - 如何修改？
  - 确保没有引入新漏洞？
- 自动化程序重写
  - 如何使所有版本更新？
- 有没有更好的选择？
  - Policy-driven的漏洞修复
  - 系统级的漏洞修复支持

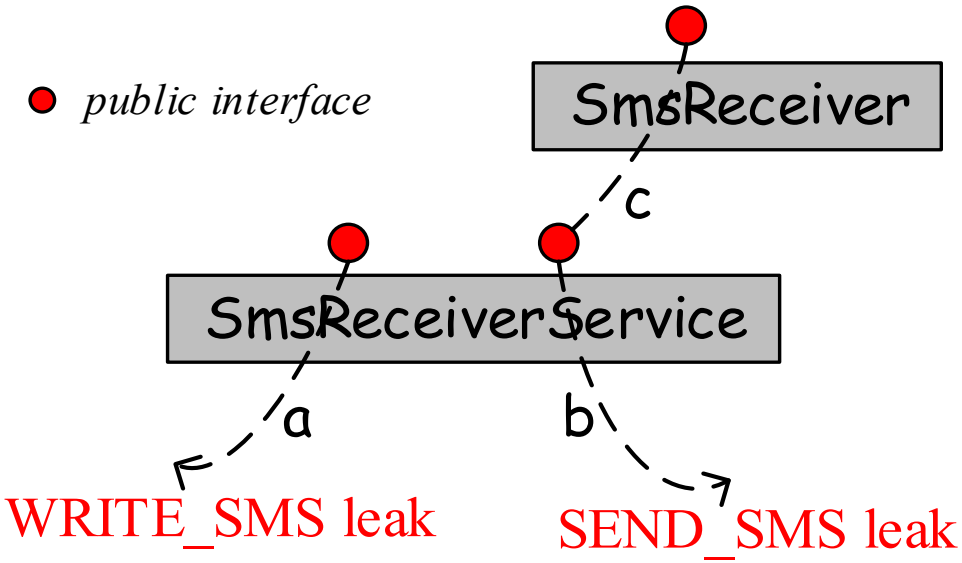
权限泄露漏洞

```
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

权限泄露漏洞分析



- 漏洞利用程序
  - 三条漏洞路径：a、b、c->b
  - 编写测试程序，均可成功利用这些漏洞

权限泄露漏洞分析

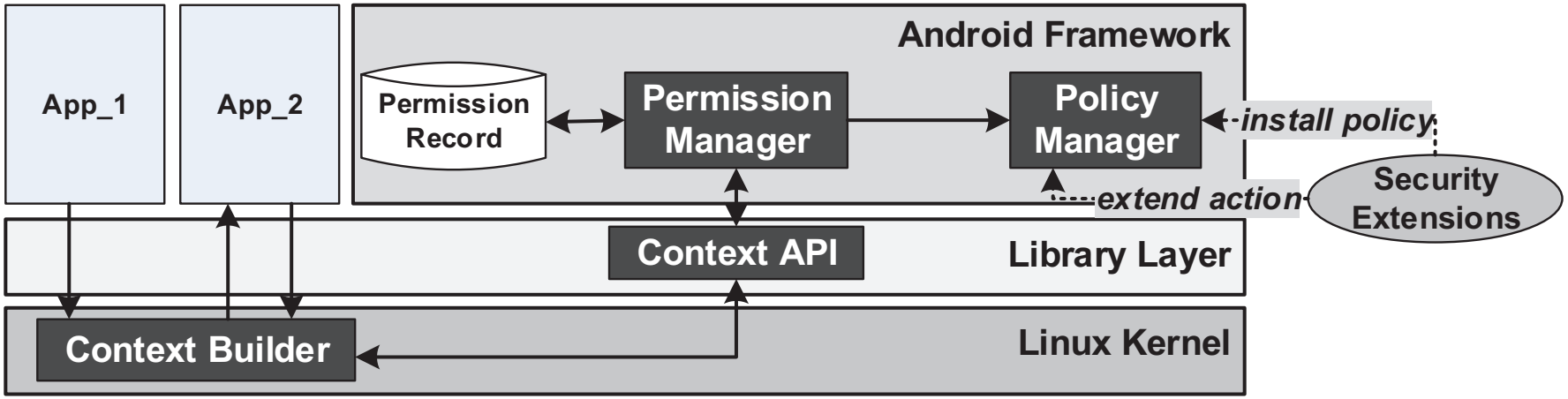
- Why permission leak?
  - intra/inter component interaction share the same communication channel
  - components may be designed for both **public use** and **internal use**

- Key to fix permission leak
  - Identify vulnerable path
  - Differ public use or internal use

FineDroid

- Key Idea
  - Permission Request <--> Application Context
  - **New Granularity**
    - **At the level of application execution context**
- Application Execution Context
  - Intra-application context, *mark the execution flow inside the app*
  - Inter-application context, *mark the IPC behavior among interacted apps*
- Policy-driven Permission Framework
  - Unified Permission Interception
  - Extensible Policy Framework

系统设计



- *Building and Propagating Context along with application execution*
- *Intercept all permission requests with system-wide application execution context*
- *Query policy framework to handle permission request in a context-sensitive manner*

核心模块

- Context Builder
  - Intra-application context builder
  - Inter-application context builder
- Context Propagator
  - Intent-level, Thread-level, Event-level
- Policy-driven Permission Framework
  - Permission Interception
  - Extensible Policy Framework

Intra-application Context Builder

- *Mark the execution flow inside the app*
  - *using Calling Context to monitor internal flow*
  - *Probabilistic Calling Context (PCC)*
    - *An integer birthmark based on all the functions in the flow*
    - *Can be calculated with a recursive expression*

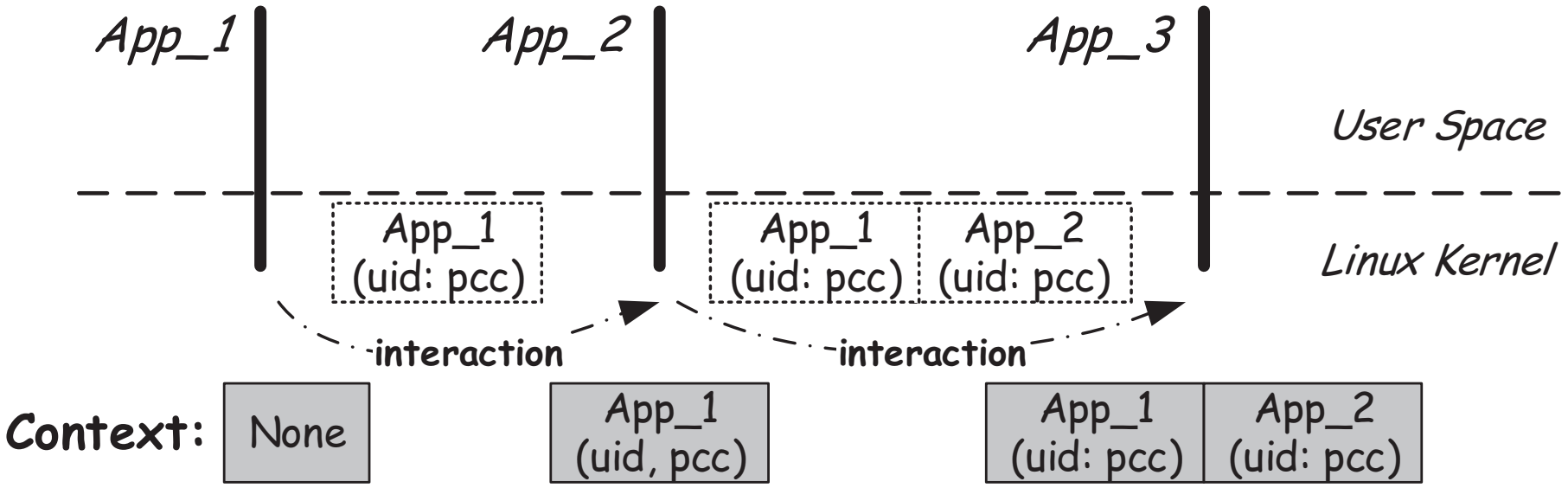
$$pcc = 3 * pcc' + cs$$

where pcc' *is PCC value of caller and* cs *is the birthmark of call site*

    - *Call-site birthmark: offset of the invoke-site in DEX*

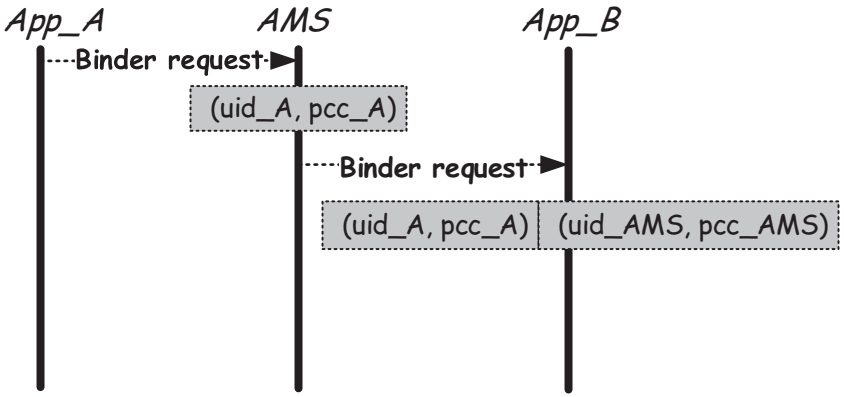
Inter-application Context Builder

- *Mark the IPC behavior of interacted apps*
  - *Using Binder to track IPC behaviors*
  - *Inter-Context: **UID + PCC***

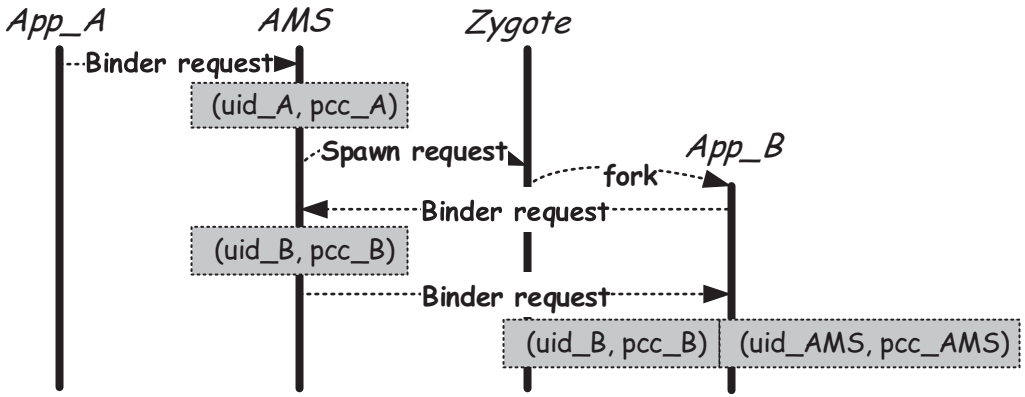


Context Propagating

- Complex application dynamics may break the context propagating
- **Intent-level**



(a) When App_B is started



(b) When App_B is not started

- **Solution:** bind context to Intent object

Context Propagating

- **Thread-level**
  - New thread should inherit the context of its parent thread
  - New resumed thread should inherit the context of the initiator thread
- **Event-level**
  - Async behaviors via callbacks
  - e.g., OnClickListener, OnLocationListener
  - Callbacks should inherit the context of its register

Policy-driven Permission Framework

- Permission Interception
  - **Goal:** *intercepts all permission use and manages the requests in a unified point*
  - Kernel-Enforced Permission
    - relies on UID/GID isolation
    - redirects the permission requests to framework
  - Android-Enforced Permission
    - relies on PMS to check permission
    - queries policy manager to make a decision

Policy-driven Permission Framework

- Policy Language
  - Declarative language to specify rules

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

  - Implemented in XML format
    - Tags: policy, uid-selector, uid-context, pcc-selector, method-sig, or, and, not
    - *Details can be found in the paper*
- Policy Matching
  - Find a policy that best match the current context

修复Mms中的权限泄露漏洞

- How to Fix?
  - With inter-application context, differ ***public use*** or ***internal use***
  - *With intra-application context, accurately describe the vulnerable flow*
  - e.g, Fix SEND_SMS permission leak in SmsReceiver

```
<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>
```

修复规则自动生成

- 基于漏洞检测工具
  - 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

原型系统

- Android 4.1.1, *Nexus Prime and emulators*

- *Retrofitted Modules*
  - *Linux Kernel*
    - *for intra/inter context building*
    - *for Kernel permission interception*
  - *Android Framework*
    - *for context propagating*
    - *for policy management*
  - ***No modification to apps***

性能测试

- Overall Performance
  - less than *2%*, in Linpack, AnTuTu, CaffeineMark3

性能测试

- Permission Requesting Handling

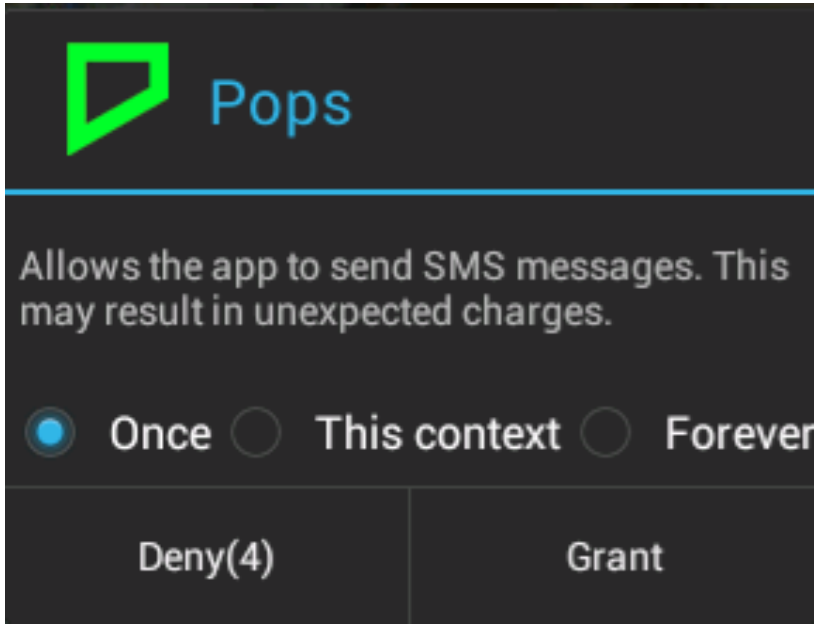
Permission Type	Origin Android	FineDroid w/o Context	FineDroid w/ Context
Socket(KEP)	0.14ms	2.16ms Δ2.02ms	2.18ms Δ0.02ms
IMEI(AEP)	0.62ms	0.69ms Δ0.06ms	1.09ms Δ0.40ms

- Policy Matching

Permission Type	FineDroid w/o Policy	FineDroid w/ Policy	Overhead
Socket(KEP)	2.18 ms	3.06 ms	0.88 ms
IMEI(AEP)	1.09 ms	1.99 ms	0.90 ms

FineDroid的其他应用场景

- 一、细粒度的权限授予机制
  - 程序中可能会有多个地方使用同一个权限
  - 程序为粒度的权限决策是否足够灵活？
  - 重打包（恶意）应用程序？
  - 安全性 vs 可用性



FineDroid的其他应用场景

- 二、进程间栈自省技术
  - 不同的代码模块具有不同的权限配置
  - E.g. 地理位置权限
    - 广告库需要用到
    - 程序自身也需要用到
    - **可否只将权限授予给程序自身的代码？**
    - 采用包含Flurry广告插件的应用程序Stock Watch进行实验

```
...
<fine-permission android:package="com.flurry.android">
  <deny android:permission="android.permission.ACCESS_FINE_LOCATION" />
  <deny android:permission="android.permission.ACCESS_COARSE_LOCATION" />
</fine-permission>
...
```

FineDroid的其他应用场景

- 二、进程间栈自省技术
  - 不同的代码模块具有不同的权限配置
  - E.g. 地理位置权限
    - 广告库需要用到
    - 程序自身也需要用到
    - **可否只将权限授予给程序自身的代码？**
    - 采用包含Flurry广告插件的应用程序Stock Watch进行实验

```
com.snapwork.finance(10053:914896)  android.permission.ACCESS_FINE_LOCATION   DENIED
      com.flurry.android.e-e-LL-42
      com.flurry.android.e-a-VLLZ-3
      com.flurry.android.v-run-V-21
```

    - 原程序使用该权限不受影响
  - FlexDroid: In-app Privilege Separation[NDSS 2016]

漏洞检测

- 基于静态分析的漏洞检测
  - Automatically check app behaviors at runtime
    - Check attacker-controlled input will influence privileged actions
  - Conservative
    - Value not determined at static-phase
    - Path not determined at static-phase
    - …
- 常用的分析手段
  - CFG
    - 单个函数的行为和逻辑分析
  - Call Graph
    - 所有可能的执行流
  - Data Dependency Graph
    - 在执行流的基础上所有可能的数据依赖情况

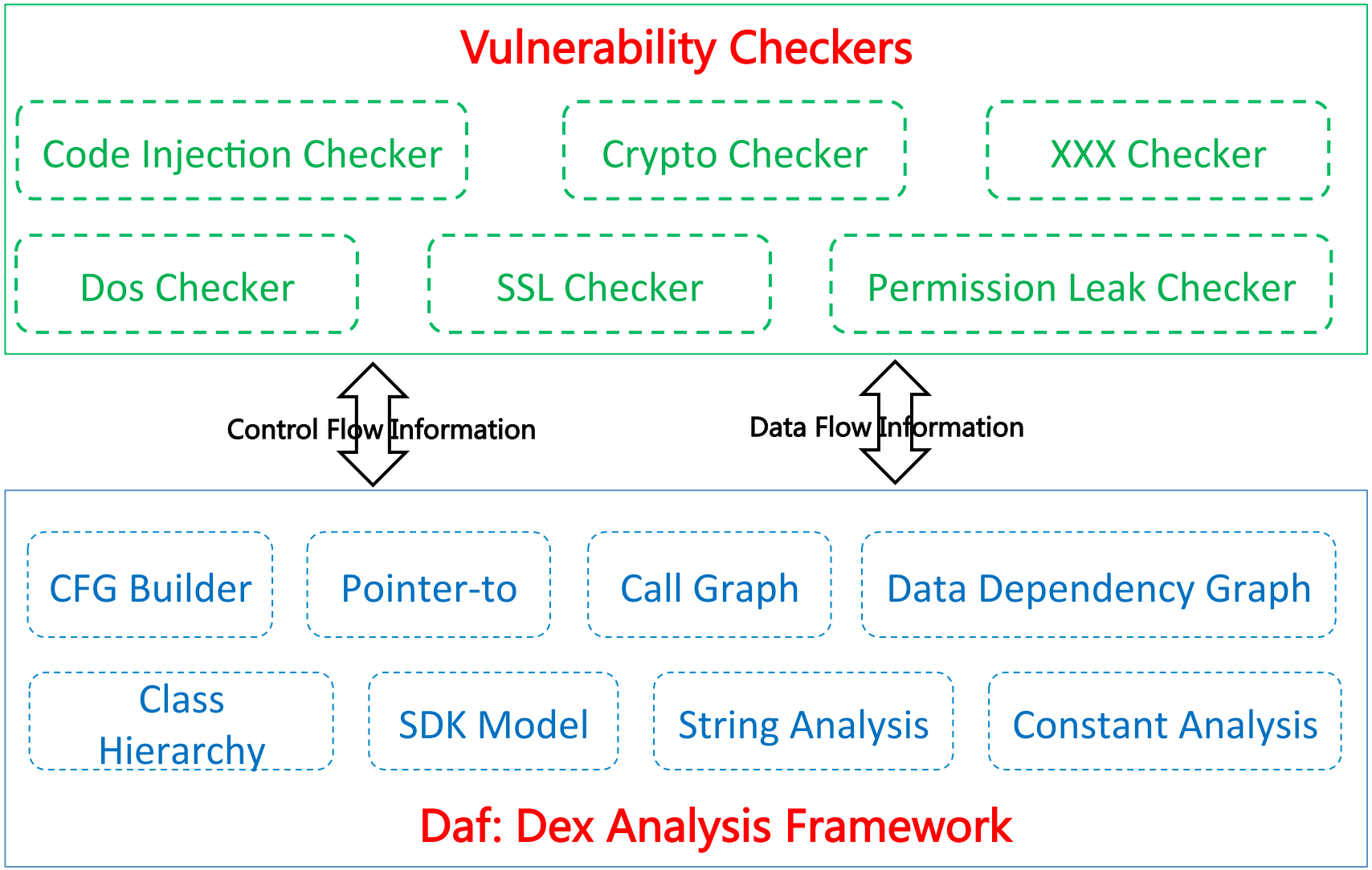
现有分析工具

- FlowDroid [PLDI 2014]
  - Soot: Java Analysis Framework
  - SPARK: point-to analysis
  - Dexpler: dex->jimple
  - IDE/IFDS-based Information Flow Analysis
  - context-, flow-, field-, object-sensitive and lifecycle-aware
  - Heavyweight, not designed for Android
- Amandroid [CCS 2014]
  - Sireum: static analysis, symbolic execution, etc
  - context-, flow-sensitive and handles ICC
  - written in Scala
  - Hard to study, limited community support

Daf: Dex Analysis Framework

- Overview
  - Directly analyze dex, build from dexlib
  - Highly customizable
  - written in Java*, still under development*
  - context-, flow-, field-, object-sensitive
- Modules
  - Point-to analysis
  - Call Graph
  - Data Dependency Graph
  - String analysis
  - Constant analysis
  - Exception analysis
  - ICC Analysis (to do)

漏洞检测框架


Vulnerability Checkers
Code Injection Checker
Crypto Checker
XXX Checker
Dos Checker
SSL Checker
Permission Leak Checker
Control Flow Information
Data Flow Information
CFG Builder
Pointer-to
Call Graph
Data Dependency Graph
Class Hierarchy
SDK Model
String Analysis
Constant Analysis
Daf: Dex Analysis Framework

以DosChecker为例

- Input-Validation Flaws in Components
  - Intent-related APIs may cause exceptions
  - NullPointerException

```
Intent i = new Intent();
if (i.getAction().equals("TestForNullPointerException")) {
    Log.d("TAG", "Test for Android Refuse Service Bug");
}
```

  - ClassCastException

```
Intent i = getIntent();
String test = (String)i.getSerializableExtra("serializable_key");
```

  - ClassNotFoundException

```
Intent i = getIntent();
i.getSerializableExtra("serializable_key");
```

检测方法

- 静态检测
  - Exception分析
    - 识别会触发Dos异常的语句（Intent相关API）
    - 判断是否有try-catch块
  - Intent分析
    - 判断出现异常的点是否为攻击者可以控制的输入
- 动态测试
  - String分析
    - Intent.setAction(String)
    - Intent.putExtra(String, Object)
  - Intent注入器
    - 发送Intent到目标程序组件
    - 检测程序组件是否Crash

案例分析-平安口袋银行

- SSL缺陷
  - ISF 2014
  - 登陆信息泄露
  - 转账信息泄露
  - 用户信息泄露
  - 转账交易泄露/劫持

案例分析-中国电信翼支付

- 支付授权缺陷
  - GeekPwn 2015 优胜奖
  - 任意账户金额远程盗刷
  - 自动遍历受害账户进行攻击

总结

FineDroid安全框架
- 程序上下文跟踪
- Policy-driven漏洞修复

漏洞修复

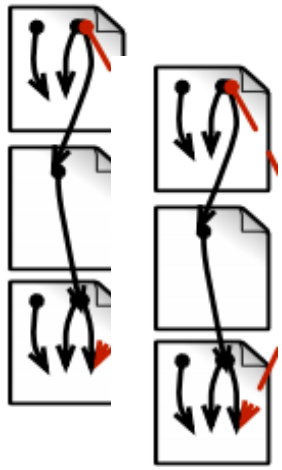
根据漏洞细节修复漏洞

代码分析
漏洞特征描述

Daf漏洞检测框架
- 基于DEX的静态分析
- Plugin-based漏洞检测

漏洞检测

漏洞特征报告



Q&A