PennState

Learning from Ourselves:

Where are we and where can we go in mobile systems security?

Patrick McDaniel, Penn State University

A cautionary tale …

Where are we now …

- September 23, 2008 – May 26th, 2016
  - 7.67 years
  - 242,179,200 seconds
  - 4,036,320 minutes
  - 67,272 hours
  - 2,803 days
  - 400 weeks and 3 days

2008 View : Security and smartphones

- Smartphones: long awaited realization of mobile computing
- Usage model is very different
  - Multi-user single machine to single-user multiple machines
  - Always on, always computing social instrument
  - Enterprise: separate action from geography
- Changing Risk
  - Necessarily contains secrets (financial, personal)
  - Collects sensitive data as a matter of operation
  - Drifts between "*unknown*" environments
  - Highly malleable development practices, largely unknown developers


HOME HOUSE
20 Portman Square
London W1H 9HF

THE RESTAURANT
SRV 80 TABLE 123/1 TIME 22:55

1	HILDON	3.25
1	ARTI SOUP	5.95
1	TIGER PRAWNS	12.50
1	*********	0.00
1	BREAST OF DUCK	17.50
1	WELSH LAMB	18.75
1	DAUPHINOISE	2.75
1	ROAST PARSNIPS	2.75
1	HONORE DE BERTIC	19.50
2	PUDDING TROLLEY	11.90
1	Main Away	0.00
2	CAPPUCCINO	5.90
	SERVICE CHARGE	12.60

NET SALES 100.75 GRAND TOTAL 113.35

THIS IS NOT A VAT RECEIPT

VAT ANALYSIS

RATE %	NET	TAX	TOTAL
17.500	66.37	11.63	78.00
17.500	2.77	0.48	3.25
17.500	16.60	2.90	19.50

SIGNATURE:........................

Home House

PRINT NAME:........................

MEMBERSHIP No:....................

A DISCRETIONARY 12.5% SERVICE CHARGE
HAS BEEN ADDED TO YOUR ACCOUNT

Where are we now …

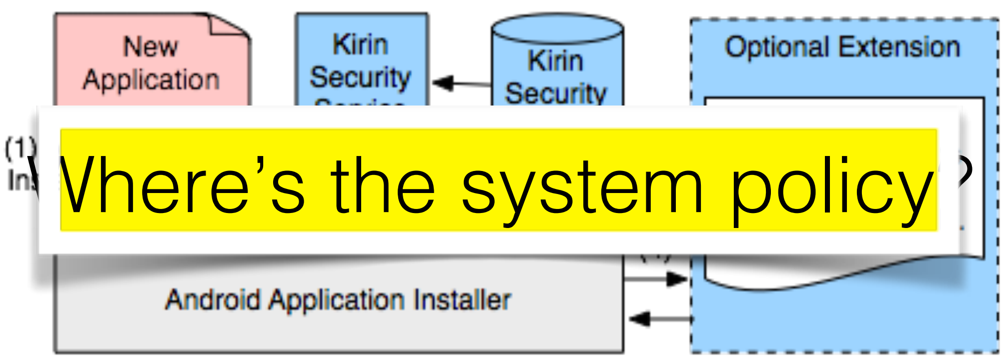
- We are closing in on a decade of research and use of smartphones.
  - What questions have we asked and what have we learned?
  - What questions should we be asking?

Promise: the next four dissertations will be ….

Three questions (2009-2011) …

What do applications ask for?

- *Kirin* certifies applications by vetting policies at install-time (relies on *runtime enforcement*)
- Obvious insight: app config and security policy is an upper bound on runtime behavior.
- Kirin is a modified application installer
  - Apps with unsafe policies are rejected

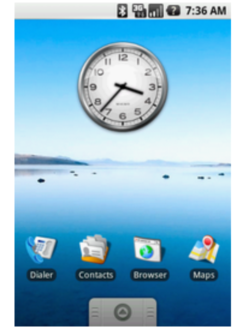


2009

William Enck, Machigar Ongtang, and Patrick McDaniel. On Lightweight Mobile Phone App Certification. *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 235-245, November 2009.

Studying the (early) Market

- Kirin enforces security invariants at install-time




```
restrict permission [ACCESS_FINE_LOCATION, INTERNET]
     and receive    [BOOT_COMPLETE]
```

  - Local evaluation of requested permissions, Intent listeners

Evaluate 311* popular Market apps (Jan 2009)

  - 5 had both dangerous configuration / functionality (1.6%)
  - 5 dangerous configs, but plausable use of permisions (1.6%)

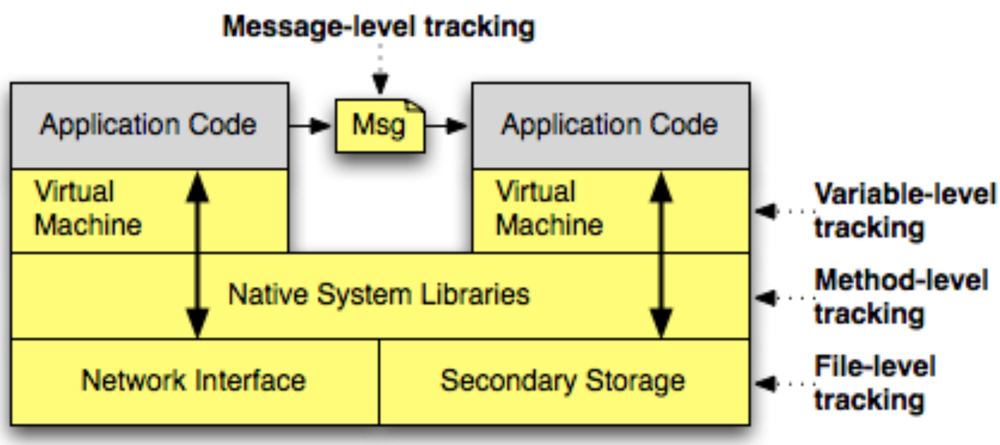
3 apps failed -- (2) *An application must not have the* PHONE_STATE, RECORD_AUDIO, *and* INTERNET *permissions*

(1) *An applicati*
(2) *An applicati*
(3) *An applicati*
(4) *An applicati* *issions*
(5) *An applicati* permissions
(6) *An applicati*
(7) *An applicati*
(8) *An application must not have the* INSTALL_SHORTCUT *and* UNINSTALL_SHORTCUT *permissions*
(9) *An application must not have the* SET_PREFERRED_APPLICATION *permission and receive Intents for the* CALL *action string*

What do the applications do?

2010

- TaintDroid is performs system-wide taint tracking in the Android platform
  1. *VM Layer*: variable tracking throughout Dalvik VM
  2. *Native Layer*: patches state after native method (JNI)
  3. *Binder IPC Layer*: extends tracking between applications
  4. *Storage Layer*: persistent tracking on files



(Firmware mod)

William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth, TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. Communications of the ACM, 57(3), March, 2014.

Findings

- 15 of the 30 applications shared physical location with an ad server (admob.com, ad.qwapi.com, ads.mobclix.com, data.flurry.com)
  - Not trying hard to hide (e.g., AdMob HTTP GET):

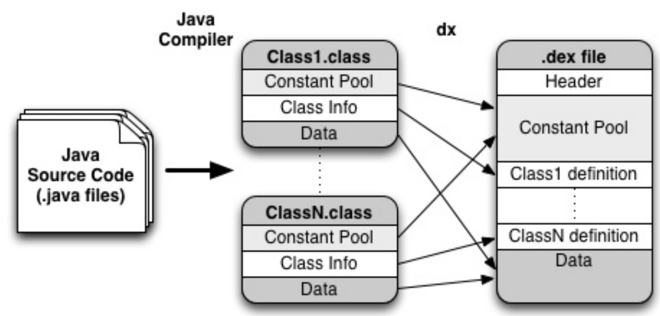
```
...&s=a14a4a93f1e4c68&..&t=062A1CB1D476DE85
B717D9195A6722A9&d%5Bcoord%5D=47.6612278900
00006%2C-122.31589477&...
```

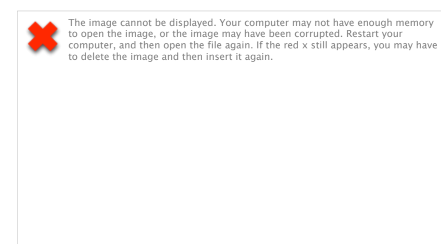
- 7 applications sent device (IMEI) and 2 apps sent phone info (Ph. #, IMSI, ICC-ID) to a remote server without informing the user.

What can the applications do?

2011

- *Static analysis*: look at the possible paths and interaction of data
  - Very, very hard (often undecidable), but community has learned that we can do a lot with small analyses.
- Step 1: decompiler for Android applications (ded)
- Step 2: static source code analysis for both *dangerous functionality* and *vulnerabilities*
  - What data could be exfiltrated from the application?
  - Are developers safely using interfaces?

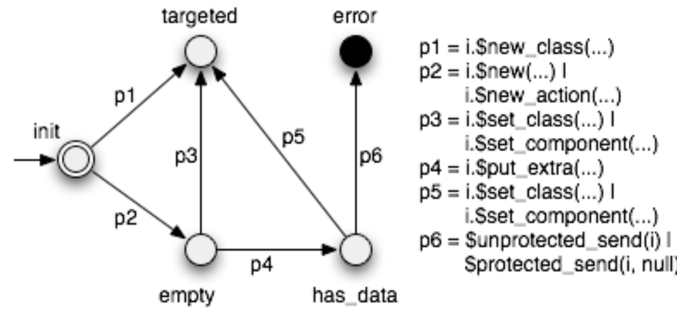




William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A Study of Android Application Security. *Proceedings of the 20th USENIX Security Symposium*, August 2011. San Francisco, CA.

Studying Application Security

- Decompiled top 1,100 apps from Android market: >*21 MLOC*
- Queried for security properties using *program analysis*, followed by *manual inspection* to understand purpose
- Used several types of analysis to design security properties specific to Android using the Fortify SCA framework



Analysis for Dangerous Behavior

Misuse of Phone Identifiers	Data flow analysis
Exposure of Physical Location	Data flow analysis
Abuse of Telephony Services	Semantic analysis
Eavesdropping on Video	Control flow analysis
Eavesdropping on Audio	Structural analysis (+CG)
Botnet Characteristics (Sockets)	Structural analysis
Havesting Installed Applications	Structural analysis

Analysis for Vulnerabilities

Leaking Information to Logs	Data flow analysis
Leaking Information to IPC	Control flow analysis
Unprotected Broadcast Receivers	Control flow analysis
Intent Injection Vulnerabilities	Control flow analysis
Delegation Vulnerabilities	Control flow analysis
Null Checks on IPC Input	Control flow analysis
Password Management*	Data flow analysis
Cryptography Misuse*	Structural analysis
Injection Vulnerabilities*	Data flow analysis

Phone Identifiers

com.avantar.wny - com/avantar/wny/PhoneStats.java

IMEI

```
public String toUrlFormatedString()
{

    StringBuilder $r4;
    if (mURLFormatedParameters == null)
    {
        $r4 = new StringBuilder();
        $r4.append((new StringBuilder("&uuid=")).append(URLEncoder.encode(mUuid)).toString());
        $r4.append((new StringBuilder("&device=")).append(URLEncoder.encode(mModel)).toString());
        $r4.append((new StringBuilder("&platform=")).append(URLEncoder.encode(mOSVersion)).toString());
        $r4.append((new StringBuilder("&ver=")).append(mAppVersion).toString());
        $r4.append((new StringBuilder("&app=")).append(this.getAppName()).toString());
        $r4.append("&returnfmt=json");
        mURLFormatedParameters = $r4.toString();
    }

    return mURLFormatedParameters;
}
```

Tracking

com.froogloid.kring.google.zxing.client.android - Activity_Router.java (Main Activity)

http://kror.keyringapp.com/service.php

```
public void onCreate(Bundle  r1)
{
   ...
   IMEI = ((TelephonyManager) this.getSystemService("phone")).getDeviceId();
    retailerLookupCmd = (new
StringBuilder(String.valueOf(constants.server))).append("identifier=").append(EncodeU
RL.KREncodeURL(IMEI)).append("&command=retailerlookup&retailername=").toString();
    ...
}
```

com.Qunar - net/NetworkTask.java

http://client.qunar.com:80/QSearch

```
public void run()
{
   ...
   r24 = (TelephonyManager) r21.getSystemService("phone");
    url = (new
StringBuilder(String.valueOf(url))).append("&vid=60001001&pid=10010&cid=C1000&uid=").appen
d(r24.getDeviceId()).append("&gid=").append(QConfiguration.mGid).append("&msg=").append(QC
onfiguration.getInstance().mPCStat.toMsgString()).toString();
    ...
}
```

Probing for Permissions

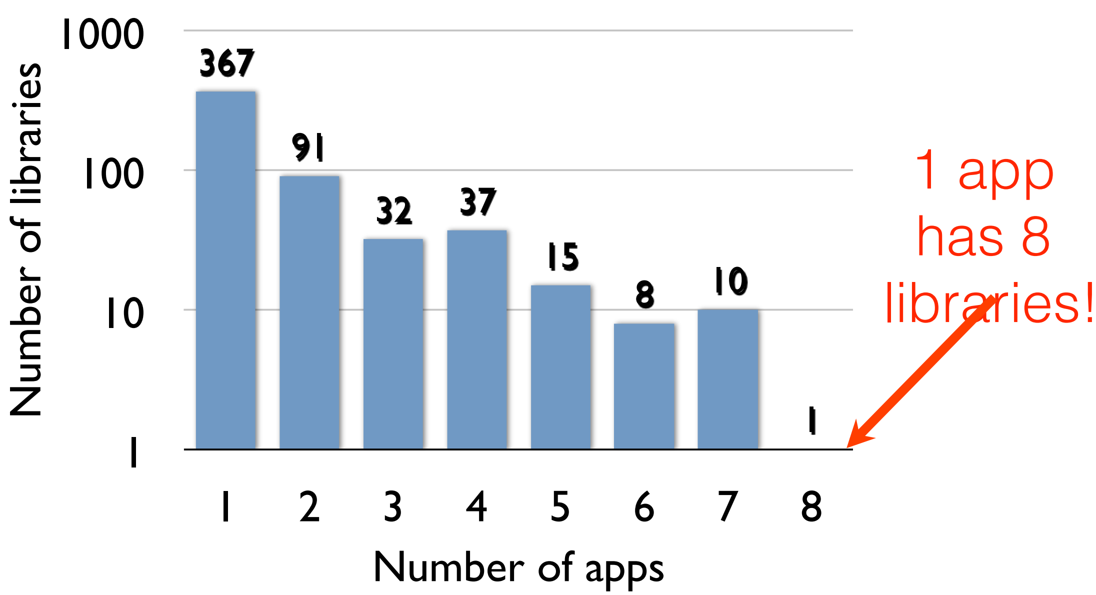
com/casee/adsdk/AdFetcher.java

```java
public static String getDeviceId(Context  r0)
{

    String r1;
    r1 = "";

    label_19:
    {
        if (deviceId != null)
        {
            if (r1.equals(deviceId) == false)
            {
                break label_19;
            }
        }

        if (r0.checkCallingOrSelfPermission("android.permission.READ_PHONE_STATE") == 0)
        {
            deviceId = ((TelephonyManager) r0.getSystemService("phone")).getSubscriberId();
        }
    } //end label_19:

    ...
}
```

Checks before accessing

Ad/Analytics Libraries

- 51% of the apps included an ad or analytics library (many also had custom functionality)
- A few libraries were used most frequently
- Use of phone identifiers and location sometimes configurable by developer



Library Path	# Apps	Obtains
com/admob/android/ads	320	L
com/google/ads	206	-
com/flurry/android	98	-
com/qwapi/adclient/android	74	L, P, E
com/google/android/apps/analytics	67	-
com/adwhirl	60	L
com/mobclix/android/sdk	58	L, E
com/mellennialmedia/android	52	-
com/zestadz/android	10	-
com/admarvel/android/ads	8	-
com/estsoft/adlocal	8	L
com/adfonic/android	5	-
com/vdroid/ads	5	L, E
com/greystripe/android/sdk	4	E
com/medialets	4	L
com/wooboo/adlib_android	4	L, P, I
com/adserver/adview	3	L
com/tapjoy	3	-
com/inmobi/androidsdk	2	E
com/apegroup/ad	1	-
com/casee/adsdk	1	S
com/webtrents/mobile	1	L, E, S, I
Total Unique Apps	**561**	

L = Location; P = Ph#, E = IMEI, S = IMSI, I = ICC-ID

Intent Vulnerabilities

- Similar analysis rules as independently verified by Chin et al. [Mobisys 2011]
- *Leaking information to IPC* - unprotected intent broadcasts are common, occasionally contain sensitive info
- *Unprotected broadcast receivers* - a few apps receive custom action strings w/out protection (lots of "protected bcasts")
- *Intent injection attacks* - 16 apps had potential vulnerabilities
- *Delegating control* - pending intents are tricky to analyze (notification, alarm, and widget APIs) --- no vulns found
- *Null checks on IPC input* - 3925 potential null dereferences in 591 apps (53%) --- most were in activity components

Non-app centric work …

So … then what?

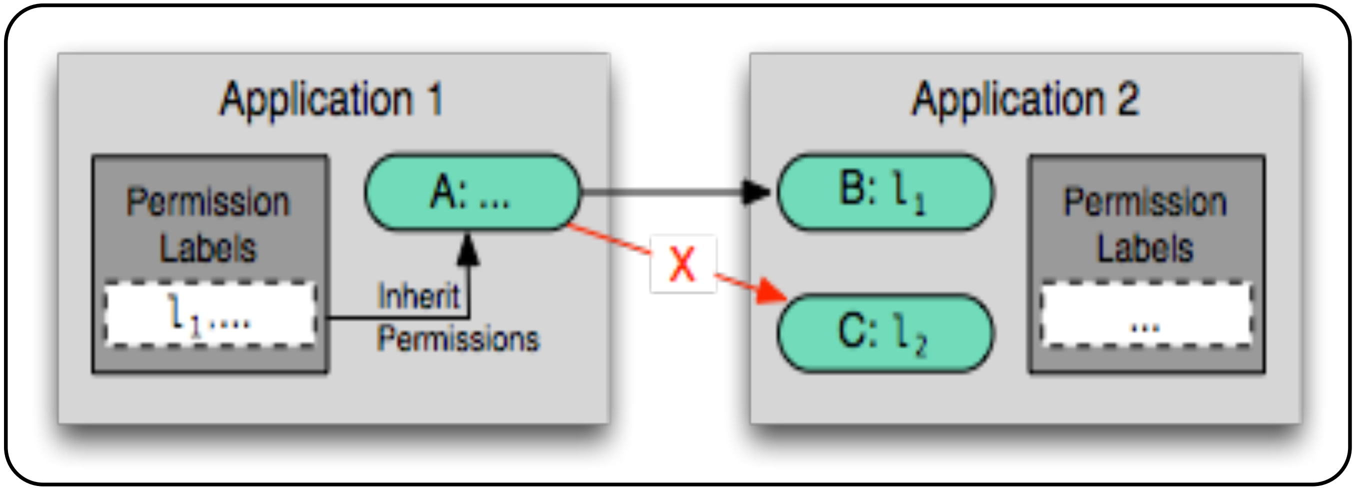
- The community has been working in concert since the early days trying to sort out not just what applications are doing, but how we deal with this new world of security.
- You can distill the non-app centric work into two areas …

Thanks to: Octeau, Enck, Porter Felt, Liu, Roesner, … and hundreds more.

What to do about permissions?

Permissions define security policy …

- Perhaps no subject has spurred more discussion and research than permissions.
  - Understanding permissions
  - Enhancing permissions



- Perhaps define who can do what to whom and when.

What is a permission?

- The existential question: a permission is a statement of a right of an application to use some interface or resource.

```
Application A can use interface/resource P.
```

- In a broader sense, a permission is a (non-negotiable) contract between the application and the user about security relevant actions.

Permission problems

- Permissions are presented largely without the context needed to make an informed decision:

```
Application A can use interface/
resource P (FOR WHAT?).
```

- *Dynamic permissions* in Marshmallow start to address context by providing temporal context, but this still lacks the specificity needed.

Permission problems

- Permissions lack the kinds of clear meaning for people to understand what they mean or what the implications are:

```
Application A can use interface/resource
  P (THAT ENCOMPASSES ..) (FOR WHAT?).
```

- Permission groups start to address context by providing needed semantics (calendar, contacts, location). But …

Permission problems

- The scope of permissions are sometimes too coarse to make informed decisions:

```
    Application A can use interface/resource
P(.REFINEMENT) (THAT ENCOMPASSES ..) (FOR WHAT?).
```

- Consider the calendar permission group. That protects the calendar database, but not controls on the elements of it.

A debate …

android.permission.INTERNET

The myth of the user …

- All of these arguments are true, but rely on a particular interpretation of "*user*".
- The problem is that there is no one single class of user or uniform set of needs for a permission system.
- The current permission system has NOT failed, it has just failed to address all possible user needs and the same time.

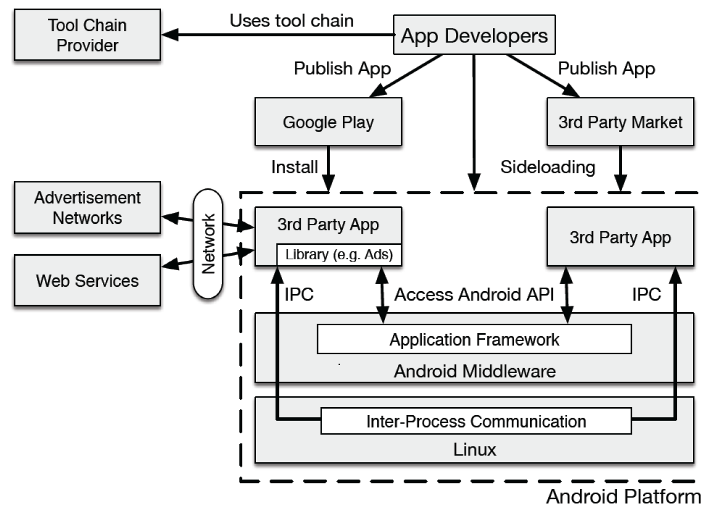
… the emergent system policy

- One of the key challenges of the current permission system is that it leads to an *emergent security policy*.
  - Each application adds something to the aggregate information flow allowed in the system, and therefore alters the security policy.
- *Implication*: Inter-component communication (ICC) analysis is essential to the security of the phone.
- *Challenge*: adding a new application may substantially influence security. Therefore security analysis must be a <u>maintenance</u> process, *not* a certification process.

What about markets?

Application markets

- Markets have changed the software industry.
  - Easy access to consumer market
  - Vender channel (*30% to market*), highly profitable
  - Low barrier to entry
    - Android structure and tools are designed to ease barriers and reduce learning curves
    - There to foster innovation (think 2008-ish)
  - Fast-always available patching



Trivia: 460,00 distinct developers as of Feb 2016

Application market myths …

- Myth: application markets provide security
- Actually, Markets can't provide security
  - They don't know what it means (because it is unknowable for any future context)
  - They can evaluate applications for compliance with proper design usage, and identify over malware … (and they do, but details are sketchy)
  - Even markets could provide security, they could not possibly perform the necessarily expensive analysis for the thousands of applications hitting the market every day

Patrick McDaniel and William Enck, Not So Great Expectations: Why Application Markets Haven't Failed Security. IEEE Security & Privacy Magazine, 8(5):76--78, September/October, 2010.

Application markets myths …

- Myth: markets identify developers and provide transparency of how users and data are part of economy
  - You know where you are getting your software from and how your data is used …
- Actually, Markets don't and can't provide transparency
  - Developer environment and run-time economy is a complex collection of hidden, and fluid relationships
    - App developers, libraries providers, third-party networks, add resellers, all have a role in development and execution
    - Monetization is opaque to the user and market.
- Repackaging: a serious consequence

Stepping back …

Future research

- There are two open areas of research that will define the future of research:
  - Permissions: how do we define and maintain security policy
  - Markets: how do we provide applications to users in a safe way
- Put another way: the next 4 dissertations topics …

Permission research

- Open problems:
  - *Permission structures and definition*: how to we design permission systems that can map to the cognitive models of users (usability) while providing for complete, granular and contextually meaningful mediation?
  - *Separating system and user policy*: How do we trade off system defined policy with user defined policy – note that the sweet spot is likely going to be dependent on the application, user, and environment?

Research in market systems …

- Open problems:
  - *Code provenance*: how can we identify the (a) developers of the application and its parts, (b) identify different parts of the application (app vs. library)
  - *Behavioral disclosure and regulation*: disclose behaviors that have security consequences (SMS premium rate, ad acquisition)

Conclusions

- Android security research is often conflated with application security analysis, but it is much larger.
  - Access control and the way we define it is essential to the future of security research
  - Getting a handle on the applications

Questions?

https://www.patrickmcdaniel.org

mcdaniel@cse.psu.edu

2008 View : Security and smartphones

- Smartphones: long awaited realization of mobile computing
- Usage model is very different
  - Multi-user single machine to single-user multiple machines
  - Always on, always computing social instrument
  - Enterprise: separate action from geography
- Changing Risk
  - Necessarily contains secrets (financial, personal)
  - Collects sensitive data as a matter of operation
  - Drifts seamlessly between "*unknown*" environments
  - Highly malleable development practices, largely unknown developers


HOME HOUSE
20 Portman Square
London W1H 9HF

THE RESTAURANT
SRV 80 TABLE 123/1 TIME 22:55

1	HILDON	3.25
1	ARTI SOUP	5.95
1	TIGER PRAWNS	12.50
1	*********	0.00
1	BREAST OF DUCK	17.50
1	WELSH LAMB	18.75
1	DAUPHINOISE	2.75
1	ROAST PARSNIPS	2.75
1	HONORE DE BERTIC	19.50
2	PUDDING TROLLEY	11.90
1	Main Away	0.00
2	CAPPUCCINO	5.90
	SERVICE CHARGE	12.60

NET SALES 100.75 GRAND TOTAL 113.35

THIS IS NOT A VAT RECEIPT

VAT ANALYSIS

RATE %	NET	TAX	TOTAL
17.500	66.37	11.63	78.00
17.500	2.77	0.48	3.25
17.500	16.60	2.90	19.50

SIGNATURE:......................
Home House
PRINT NAME:......................

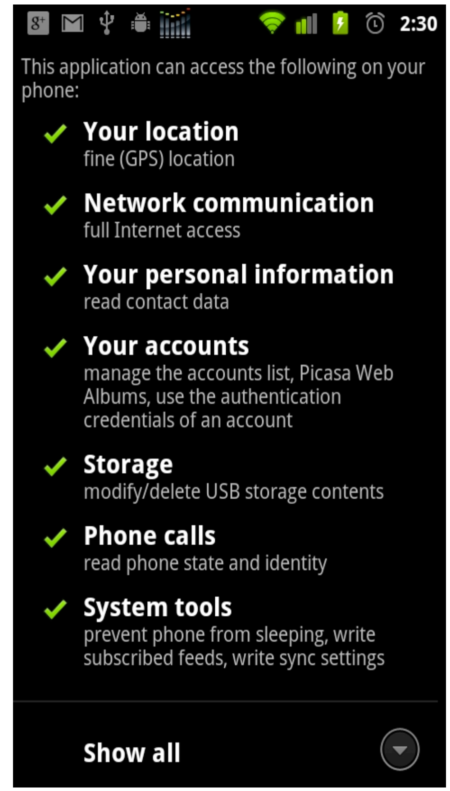
MEMBERSHIP No:....................

A DISCRETIONARY 12.5% SERVICE CHARGE
HAS BEEN ADDED TO YOUR ACCOUNT

Rethinking (host) Security

$$security == permissions \times apps$$
$$security \neq users$$

- *Permissions* define capabilities.
- *Application markets* deliver packaged *applications from* largely unknown sources.
- Users make permission decisions.
- Applications are run within middleware supported sandboxes provided by the OS.

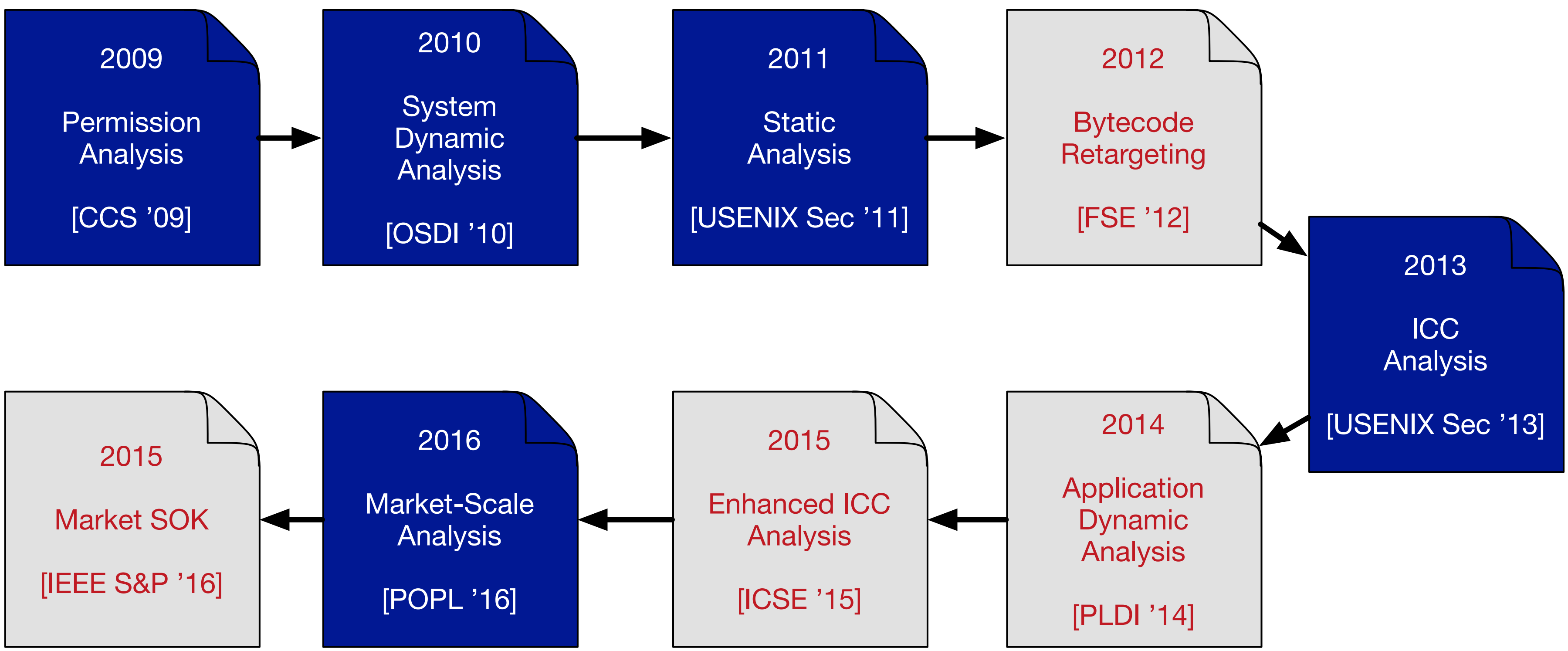


Note: App markets don't (and can't) provide security.

Patrick McDaniel and William Enck, Not So Great Expectations: Why Application Markets Haven't Failed Security. IEEE Security & Privacy Magazine, 8(5):76--78, September/October, 2010.

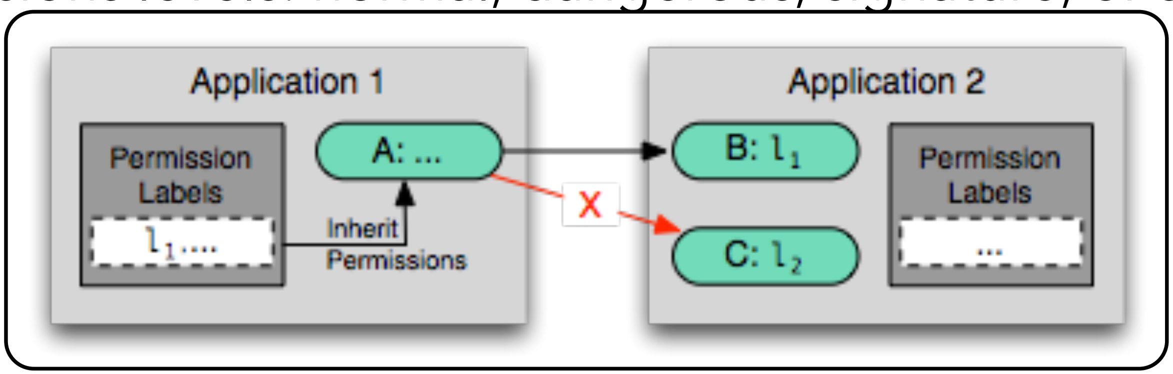
A 8-ishYear Span …

- Evaluating Android Application Security ….



Example: Android Security

- Permissions granted to applications and *never* changed
  - Permissions allow an *application* to accesses a *component*, *API*, ..
  - Runtime decisions look for assigned permissions (*access is granted IFF app A assigned perm X at install*)
  - Permissions levels: normal, dangerous, signature, or system

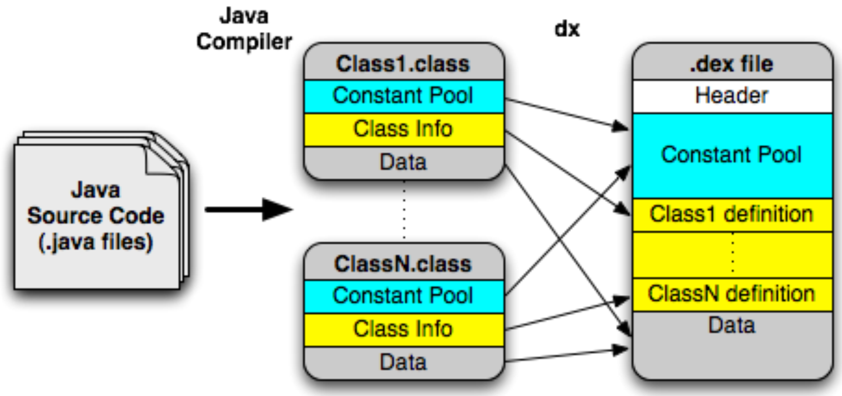


- *Example permissions*: location, phone IDs, microphone, camera, address book, SMS, application "interfaces"

William Enck, Machigar Ongtang, and Patrick McDaniel, *Understanding Android Security*. IEEE Security & Privacy Magazine, 7(1):50--57, January/February, 2009.

Aside: Dalvik EXecutables

- Android applications written in Java, compiled to Java bytecode, and translated into DEX bytecode (Dalvik VM)

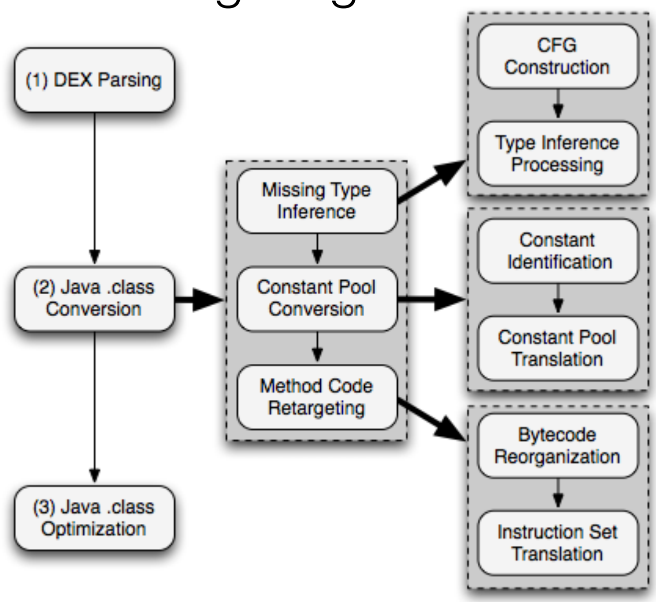


- We want to work with Java (bc), not DEX bytecode
  - There are a lot of existing program analysis tools for Java
  - We want to see what the developer was doing (i.e., confirmation)
- Non-trivial to retarget back to Java: *register vs. stack architecture*, *constant pools*, *ambiguous scalar types*, *null references*, etc.

Getting back to the source

- The *ded* (later *dare*) decompiler
  - Refers to both the entire process and `.dex` ⇒ `.class` retargeting tool
- `ded/dare` recovers logic
- from application package
  - *Retargeting*: type inference, instruction translation, etc
  - *Optimization*: use Soot to optimize Java bytecode
  - *Decompilation/IR*: standard Java decompilation (Soot) or translate to TyDe IR (typed dex in DARE)

Retargeting Process



2011

2012

DARE: Damien Octeau, Somesh Jha, and Patrick McDaniel. Retargeting Android Applications to Java Bytecode. *20th International Symposium on the Foundations of Software Engineering (FSE)*, November 2012. Research Triangle Park, NC. (**best artifact award**).

What can the applications do?

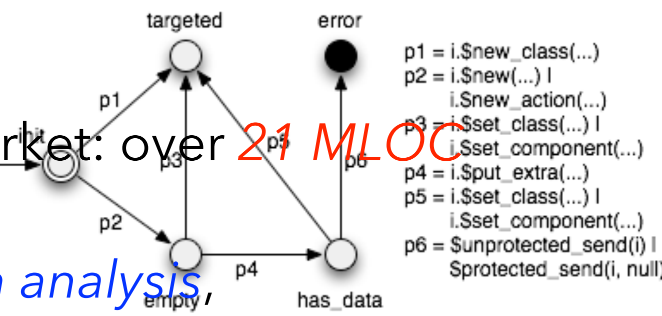
2011

- *Static analysis*: look at the possible paths and interaction of data
  - Very, very hard (often undecidable), but community has learned that we can do a lot with small analyses.
- Step 1: decompiler for Android applications (ded)
- Step 2: static source code analysis for both *dangerous functionality* and *vulnerabilities*
  - What data could be exfiltrated from the application?
  - Are developers safely using interfaces?

William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A Study of Android Application Security. *Proceedings of the 20th USENIX Security Symposium*, August 2011. San Francisco, CA.

Studying Application Security

- Decompiled top 1,100 apps from Android market: over *21 MLOC*
- Queried for security properties using *program analysis*, followed by *manual inspection* to understand purpose
- …sign… …oid



Analysis for Dangerous Behavior

Misuse of Phone Identifiers	Data flow analysis
Exposure of Physical Location	Data flow analysis
Abuse of Telephony Services	Semantic analysis
Eavesdropping on Video	Control flow analysis
Eavesdropping on Audio	Structural analysis (+CG)
Botnet Characteristics (Sockets)	Structural analysis
Havesting Installed Applications	Structural analysis

Also studied inclusion of advertisement and analytics libraries and associated properties

Analysis for Vulnerabilities

Leaking Information to Logs	Data flow analysis
Leaking Information to IPC	Control flow analysis
Unprotected Broadcast Receivers	Control flow analysis
Intent Injection Vulnerabilities	Control flow analysis
Delegation Vulnerabilities	Control flow analysis
Null Checks on IPC Input	Control flow analysis
Password Management*	Data flow analysis
Cryptography Misuse*	Structural analysis
Injection Vulnerabilities*	Data flow analysis

* Included with analysis framework

Phone Identifiers

com.avantar.wny - com/avantar/wny/PhoneStats.java

IMEI

```
public String toUrlFormatedString()
{

    StringBuilder $r4;
    if (mURLFormatedParameters == null)
    {
        $r4 = new StringBuilder();
        $r4.append((new StringBuilder("&uuid=")).append(URLEncoder.encode(mUuid)).toString());
        $r4.append((new StringBuilder("&device=")).append(URLEncoder.encode(mModel)).toString());
        $r4.append((new StringBuilder("&platform=")).append(URLEncoder.encode(mOSVersion)).toString());
        $r4.append((new StringBuilder("&ver=")).append(mAppVersion).toString());
        $r4.append((new StringBuilder("&app=")).append(this.getAppName()).toString());
        $r4.append("&returnfmt=json");
        mURLFormatedParameters = $r4.toString();
    }

    return mURLFormatedParameters;
}
```

Tracking

com.froogloid.kring.google.zxing.client.android - Activity_Router.java (Main Activity)

http://kror.keyringapp.com/service.php

```
public void onCreate(Bundle  r1)
{
   ...
   IMEI = ((TelephonyManager) this.getSystemService("phone")).getDeviceId();
    retailerLookupCmd = (new
StringBuilder(String.valueOf(constants.server))).append("identifier=").append(EncodeU
RL.KREncodeURL(IMEI)).append("&command=retailerlookup&retailername=").toString();
    ...
}
```

com.Qunar - net/NetworkTask.java

http://client.qunar.com:80/QSearch

```
public void run()
{
   ...
   r24 = (TelephonyManager) r21.getSystemService("phone");
    url = (new
StringBuilder(String.valueOf(url))).append("&vid=60001001&pid=10010&cid=C1000&uid=").appen
d(r24.getDeviceId()).append("&gid=").append(QConfiguration.mGid).append("&msg=").append(QC
onfiguration.getInstance().mPCStat.toMsgString()).toString();
    ...
}
```

Registration and Login

com.statefarm.pocketagent - activity/LogInActivity$1.java (Button callback)

IMEI

```
public void onClick(View  r1)
{
    ...
    r7 = Host.getDeviceId(this$0.getApplicationContext());
    LogInActivity.access$1(this$0).setUniqueDeviceID(r7);
    this$0.loginTask = new LogInActivity$LoginTask(this$0, null);
    this$0.showProgressDialog(r2, 2131361798, this$0.loginTask);
    r57 = this$0.loginTask;
    r58 = new LoginTO[1];
    r58[0] = LogInActivity.access$1(this$0);
    r57.execute(r58);
    ...
}
```

How would you feel about a PII to phone database?

Probing for Permissions

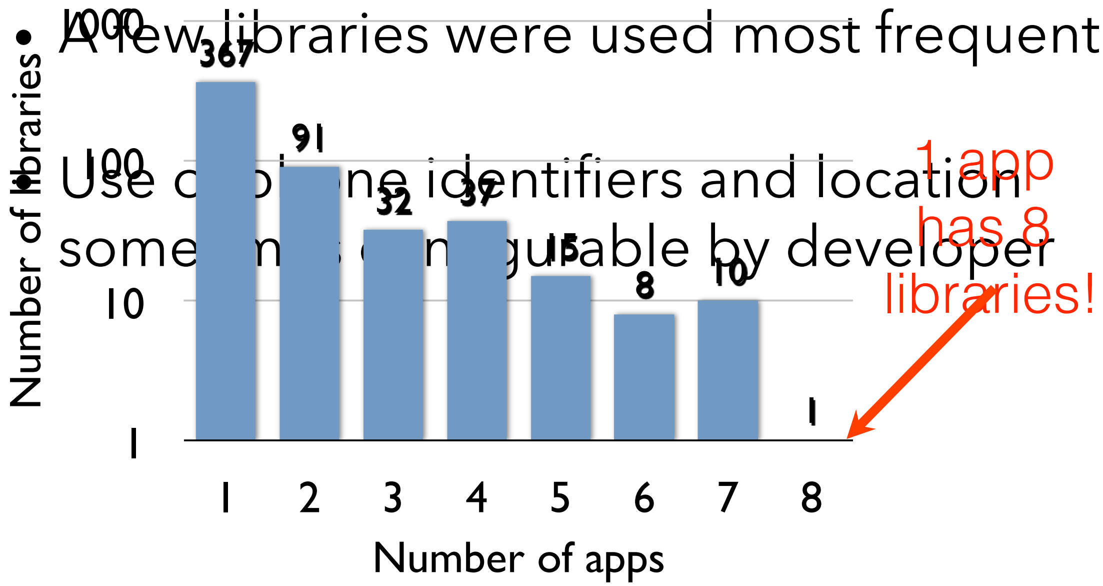
com/casee/adsdk/AdFetcher.java

```
public static String getDeviceId(Context  r0)
{

    String r1;
    r1 = "";

    label_19:
    {
        if (deviceId != null)
        {
            if (r1.equals(deviceId) == false)
            {
                break label_19;
            }
        }

        if (r0.checkCallingOrSelfPermission("android.permission.READ_PHONE_STATE") == 0)
        {
            deviceId = ((TelephonyManager) r0.getSystemService("phone")).getSubscriberId();
        }
    } //end label_19:

    ...
}
```

Checks before accessing

Ad/Analytics Libraries

- 51% of the apps included an ad or analytic library (many also had custom functionality)
- A few libraries were used most frequently
- Use of phone identifiers and location sometimes configurable by developer



Library Path	# Apps	Obtains
com/admob/android/ads	320	L
com/google/ads	206	-
com/flurry/android	98	-
com/qwapi/adclient/android	74	L, P, E
com/google/android/apps/analytics	67	-
com/adwhirl	60	L
com/mobclix/android/sdk	58	L, E
com/mellennialmedia/android	52	-
com/zestadz/android	10	-
com/admarvel/android/ads	8	-
com/estsoft/adlocal	8	L
com/adfonic/android	5	-
com/vdroid/ads	5	L, E
com/greystripe/android/sdk	4	E
com/medialets	4	L
com/wooboo/adlib_android	4	L, P, I
com/adserver/adview	3	L
com/tapjoy	3	-
com/inmobi/androidsdk	2	E
com/apegroup/ad	1	-
com/casee/adsdk	1	S
com/webtrents/mobile	1	L, E, S, I
Total Unique Apps	**561**	

L = Location, P = Ph#, E = IMEI, S = IMSI, I = ICC-ID

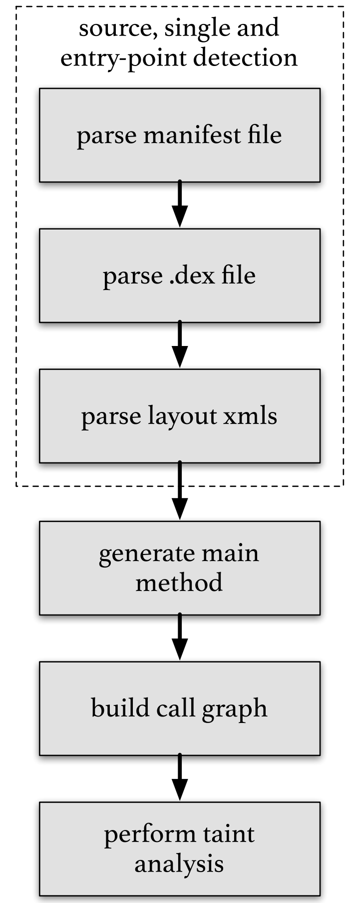
Intent Vulnerabilities

- Similar analysis rules as independently verified by Chin et al. [Mobisys 2011]
- *Leaking information to IPC* - unprotected intent broadcasts are common, occasionally contain sensitive info
- *Unprotected broadcast receivers* - a few apps receive custom action strings w/out protection (lots of "protected bcasts")
- *Intent injection attacks* - 16 apps had potential vulnerabilities
- *Delegating control* - pending intents are tricky to analyze (notification, alarm, and widget APIs) --- no vulns found
- *Null checks on IPC input* - 3925 potential null dereferences in 591 apps (53%) --- most were in activity components

Data Flow (revisited)

2014

- Application analysis is more challenging because of application execution "life-cycle"
  - E.g., component asynchrony, multiple entry points, system events, callbacks …
- FlowDroid is a static taint analysis system that tracks data flow from sources to sinks
  - Approach: identify all entry points construct a dummy main, perform analysis
- Analysis: 93% recall and 86% precision
  - DroidBench (39 hand crafted applications)
- Market or enterprise level analysis
  - Getting back to the certification model of Kirin

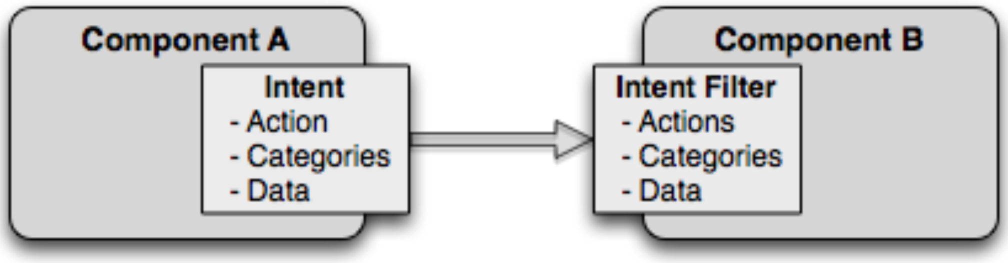


Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. *FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps*. Proc. of the 35th Programming Language Design and Implementation (PLDI), June 2014

How do applications work on concert?

2013

- *Intents* are used to pass information between apps
- IPC (*intra*-) data flows



- ICC Analysis: location of ICC, and data (types, attributes)
  - Soundness: all Intra-Component-Communication (ICC) identified
  - Precision: reduce number of false positives
  - Enable security analysis of ensemble of applications
    - Data flows between components within application
    - Exported flows/interfaces are used by other applications

Damien Octeau, Patrick McDaniel, Somesh Jha, Alexandre Bartel, Eric Bodden, Jacques Klein, and Yves Le Traon. Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis.*Proceedings of the 22th USENIX Security Symposium*, August 2013. Washington, DC.

Analysis Results

- Epicc builds a *model* of ICC
  - Reduce to an *Interprocedural Distributive Environment (IDE)* problem and extract possible Intent values (specifications)
  - Experiment: attempt to recover Intent use in 1200 applications (850 most popular, 350 random applications),
    - Runtime: average 113 seconds per application
  - Entry/exit point analysis
    - All attributes known in about 93% of ICC specifications
    - 56,106 exits points
      - 90% were found to have fixed Intent specification
      - *45% have key-value data*
    - 29,154 entry points
      - About 95% were found to have single Intent Filter specification
      - 8,566 exported components, *5% protected by permissions*

ICC Analysis version 2.0

2015

- IC3: Inter-Component Communication Analysis in Android with COAL
  - More sophisticated two-phase string analysis using flow graph of constraints on string operations
  - Added deeper URI analysis
  - Experiment: analyze ICC for 460 apps using IC3 and Epicc

Precision

	EPICC	IC3
Intents/Filters	69%	86%
URIs	34%	72%
Total	66%	85%

Identified (possible) ICC Flows

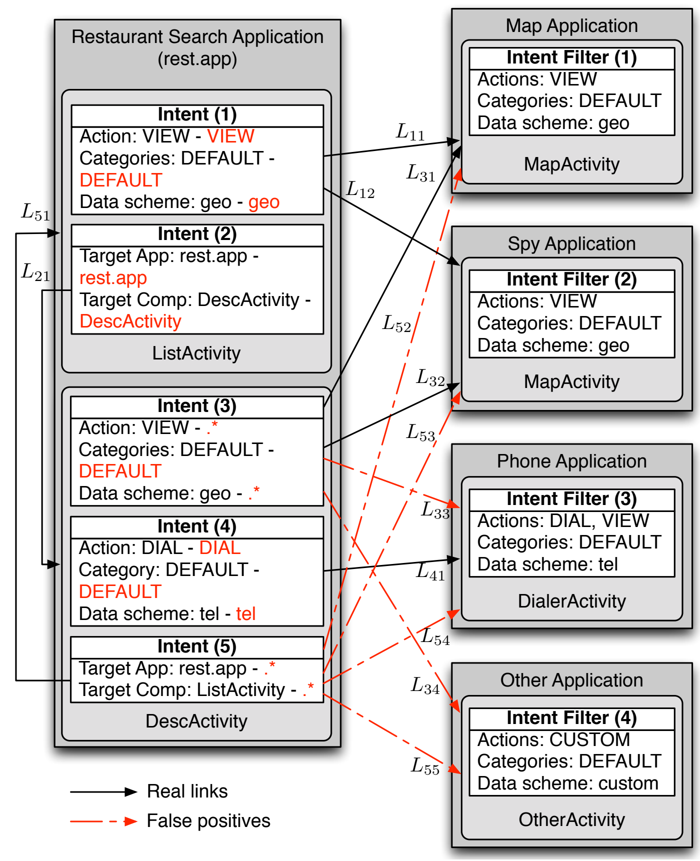
Epicc: 120,817
IC3: 26, 872

Damien Octeau, Daniel Luchaup, Matthew Dering, Somesh Jha, and Patrick McDaniel. Composite Constant Propagation: Application to Android Inter-Component Communication Analysis. Proceedings of the 37th International Conference on Software Engineering (ICSE), May 2015.

Ongoing: Scaling up analysis …

2016

- Static analysis
  - Epicc/EC2: find all Intent values at message-passing program points
  - Small static analysis imprecisions cause explosion in number of links at large scale
  - 600 apps -> 2 million links!
- Two challenges
  - Intent resolution
  - Intent flow ranking



D. Octeau, S. Jha, M. Dering, P.McDaniel, A. Bartel, Li Li, J.Klein, and Yves Le Traon. Combining Static Analysis with Probabilistic Models to Enable Market-Scale Android Inter-Component Analysis. Proceedings of the 43rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), January 2016. St. Petersburg, Florida, USA.

1: Intent Resolution

- Intent resolution – identifying the flows between apps
  - Compute inter-component links in scalable manner
  - Take into account field value regular expressions
  - Our algorithm is based on intersecting sets of Filters that verify several Intent resolution tests
    - Exploits fast matching of constant intent values
  - Runs in time $O(e \cdot |I|)$, where e is small constant
- Experiment
  - Match 10,928 applications
  - Runtime: 8,434 seconds (140 mir

Intent
Application Name
Target Application
Target Component
Uses Permissions
Permission
Type
Action
Categories
Data

Intent Filter
Application Name
Component Name
Permission
Uses Permissions
Exported
Type
Actions
Categories
Data

2: Intent flow ranking

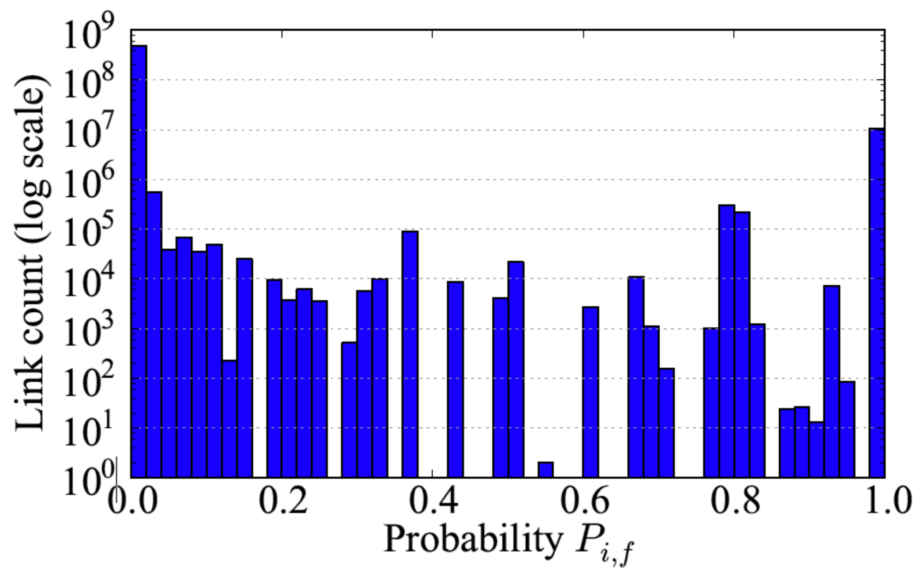
- Intent flow ranking – determining estimated likelihood of flows being “real” by comparing against known flows
- Idea: Intents are highly predictable
  - For example, displaying a map is done by sending Intent with VIEW action and geo scheme (common to applications)
  - Explicit Intents almost always target components within the same application, but often identified as being inter-application
- Approach
  - Estimate the probability of having a given Intent field combination, given the Intents that are known, i.e., to simplify

P(flow) = % known Intent matching specifications matching Intent filter

- Intuition: how similar is potential flow to known flows

Preliminary Results

- 10,928 applications, 489,099,606 potential ICC flows
- 111,254 components, 58,480 Intent filters
- 452,984 Intent values (47% explicit, 53% implicit)
- Key Results
  - 97.3% links Pr() < 0.1
  - 75% explicit links are tagged as inter-applicatio
  - 99.6% of Implicit links are inter-application



(a) Overall distribution of all probability values.

- Take away: vastly reduces the number of links

Conclusions

- The security community has been analyzing mobile applications for almost a decade now ..
  - Research is driven by deep and deeper questions
    - Analysis techniques are getting more sophisticated …
  - Volumes of data are getting larger …
  - Adversarial behavior getting subtler and more costly …
  - Industrial and academic cooperation is very strong …
  - Future is bright for research!