A Study of Android Application Security

William Enck, Damien Octeau, Patrick McDaniel,
and Swarat Chaudhuri

USENIX Security Symposium
August 2011

New Dominant Player

Android takes almost 50% share of worldwide smart phone market

- iOS becomes second largest smart phone platform

Palo Alto, Singapore and Reading (UK) - Monday, 1 August 2011

canalys
Insight. Innovation. Impact.

New Dominant Player

Android takes almost 50% share of worldwide smart phone market

- iOS becomes second largest smart phone platform

Palo Alto, Singapore and Reading (UK) - Monday, 1 August 2011

canalys
Insight. Innovation. Impact.

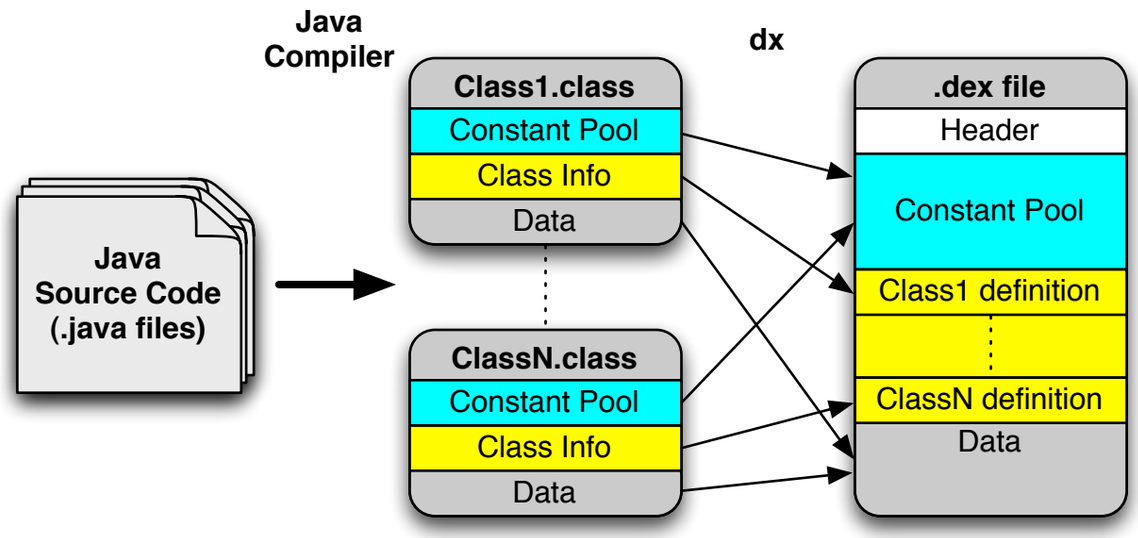
- Nobody is looking at all the apps (250K and growing)
- What would you look for if you did?

Studying Applications

- Goal: *Study a breadth of security properties in a large set of popular Android smartphone applications.*
- How do you get the applications and source code?
  - How to retrieve application packages (.apk files)?
  - How to retrieving application source code?
- How do you go about studying the source code?
  - What do you look for?
  - How do you look for it?
  - How do you know what's actually there?

Dalvik EXecutables

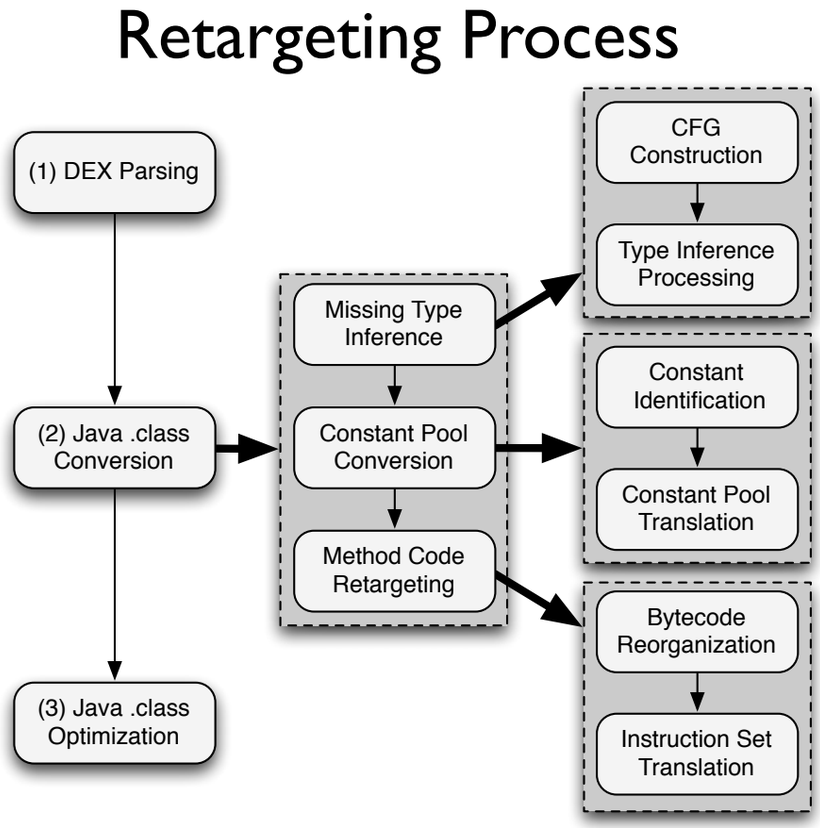
- Android applications written Java, compiled to Java bytecode, and translated into DEX bytecode (Dalvik VM)



- We want to work with Java, 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* decompiler
  - Refers to both the entire process and the `.dex` ⇒ `.class` retargeting tool
  - Multi-stage process with many sub-stages
  - http://siis.cse.psu.edu/ded



- ded recovers source code from application package
  - *Retargeting*: type inference, instruction translation, etc
  - *Optimization*: use Soot to re-optimize for Java bytecode
  - *Decompilation*: standard Java decompilation (Soot)

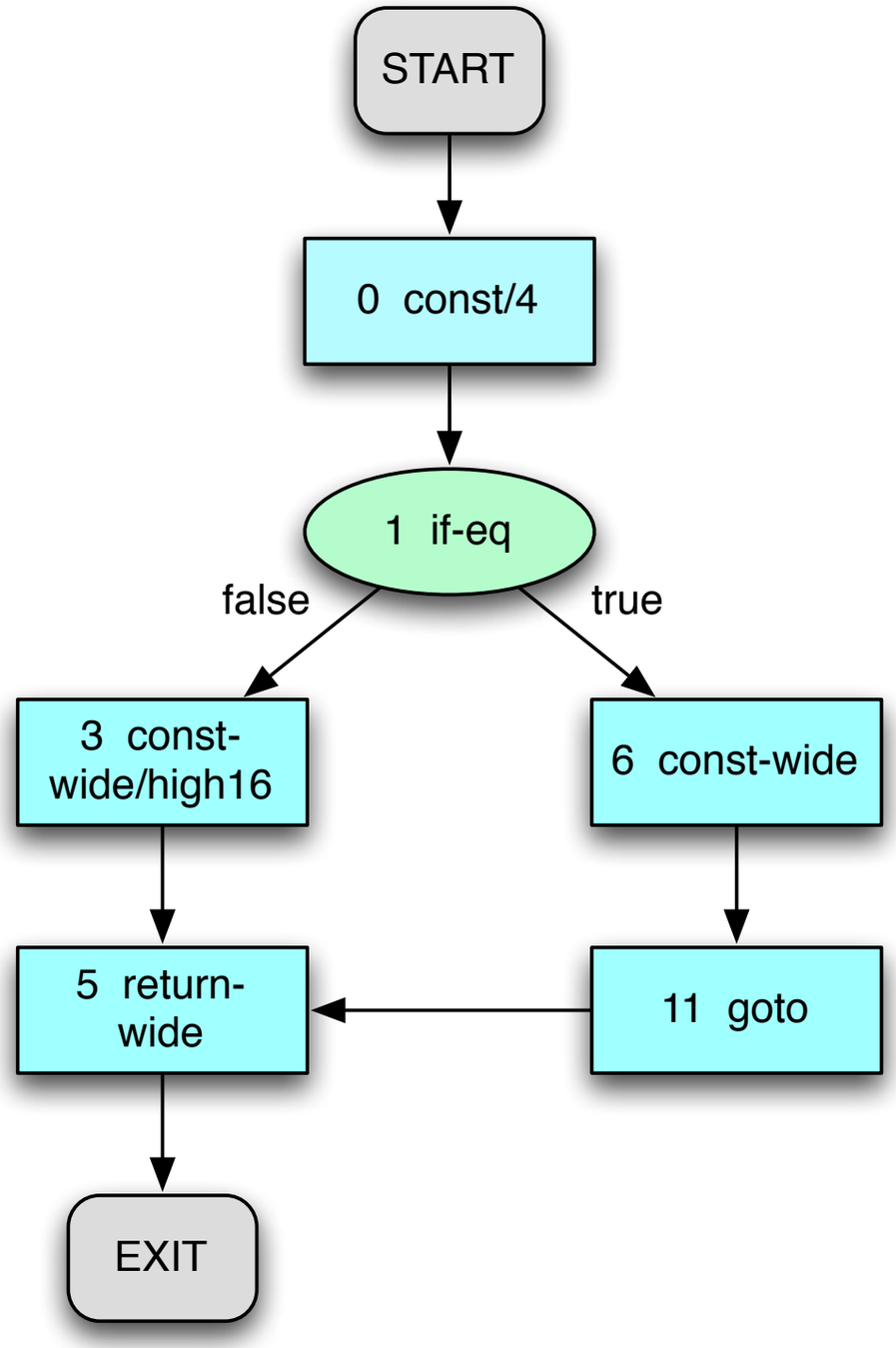
Type Inference

```
double return_a_double(int a) {
    if(a != 1)
        return 2.5;
    else
        return 1.2;
}
```

Source code

```
double return_a_double(int)
  0:  const/4 v0,1
  1:  if-eq v3,v0,6
  3:  const-wide/high16 v0,16388
  5:  return-wide v0
  6:  const-wide v0,4608083138725491507
  11: goto 5
```

DEX bytecode



CFG

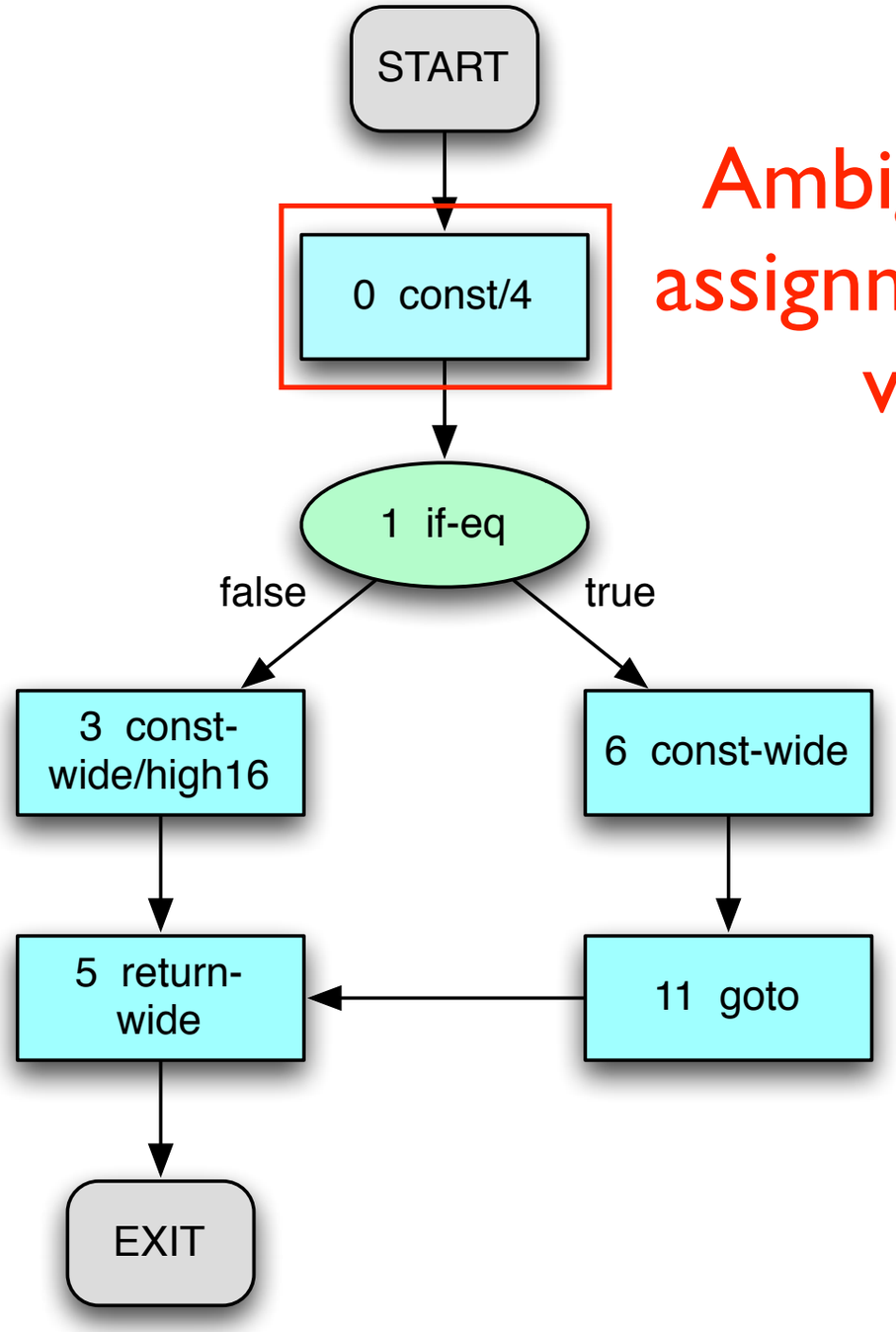
Type Inference

```
double return_a_double(int a) {
    if(a != 1)
        return 2.5;
    else
        return 1.2;
}
```

Source code

```
double return_a_double(int)
  0:  const/4 v0,1
  1:  if-eq v3,v0,6
  3:  const-wide/high16 v0,16388
  5:  return-wide v0
  6:  const-wide v0,4608083138725491507
  11: goto 5
```

DEX bytecode



Ambiguous assignment to v0

CFG

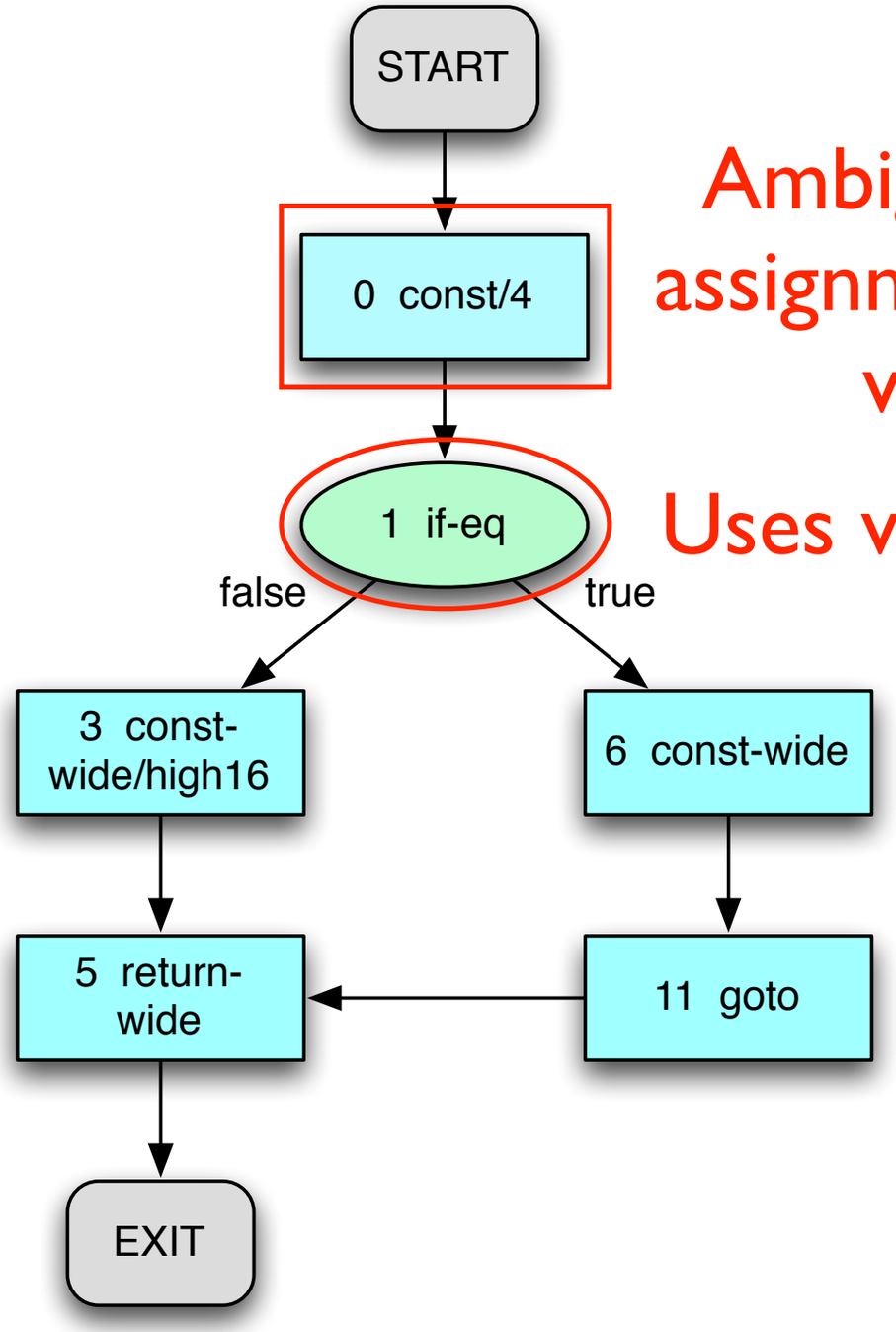
Type Inference

```
double return_a_double(int a) {
    if(a != 1)
        return 2.5;
    else
        return 1.2;
}
```

Source code

```
double return_a_double(int)
  0:  const/4 v0,1
  1:  if-eq v3,v0,6
  3:  const-wide/high16 v0,16388
  5:  return-wide v0
  6:  const-wide v0,4608083138725491507
  11: goto 5
```

DEX bytecode



Ambiguous assignment to v0

Uses v0

CFG

Recovering Types

ded

```
double return_a_double(int var1) {
    double var2;
    if(var1 != 1) {
        var2 = 2.5D;
    } else {
        var2 = 1.2D;
    }

    return var2;
}
```

dex2jar

```
double return_a_double(int var1) {
    long var2;
    if(var1 != 1) {
        var2 = 4612811918334230528L;
    } else {
        var2 = 4608083138725491507L;
    }

    return (double)var2;
}
```

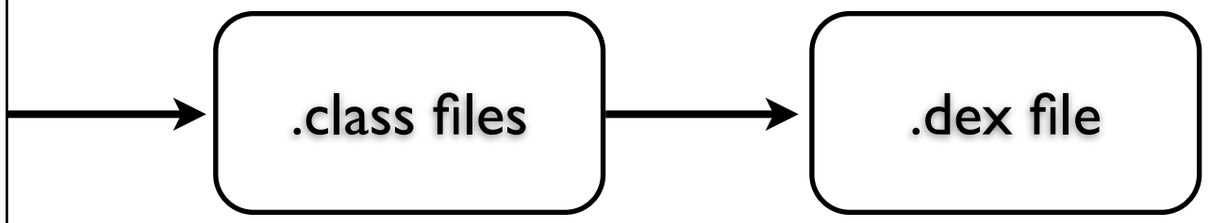
Optimization by Soot

Optimization by Soot

```
public void clearTiles() {
    for (int x = 0; x < mXTileCount; x++) {
        for (int y = 0; y < mYTileCount; y++) {
            setTile(0, x, y);
        }
    }
}
```

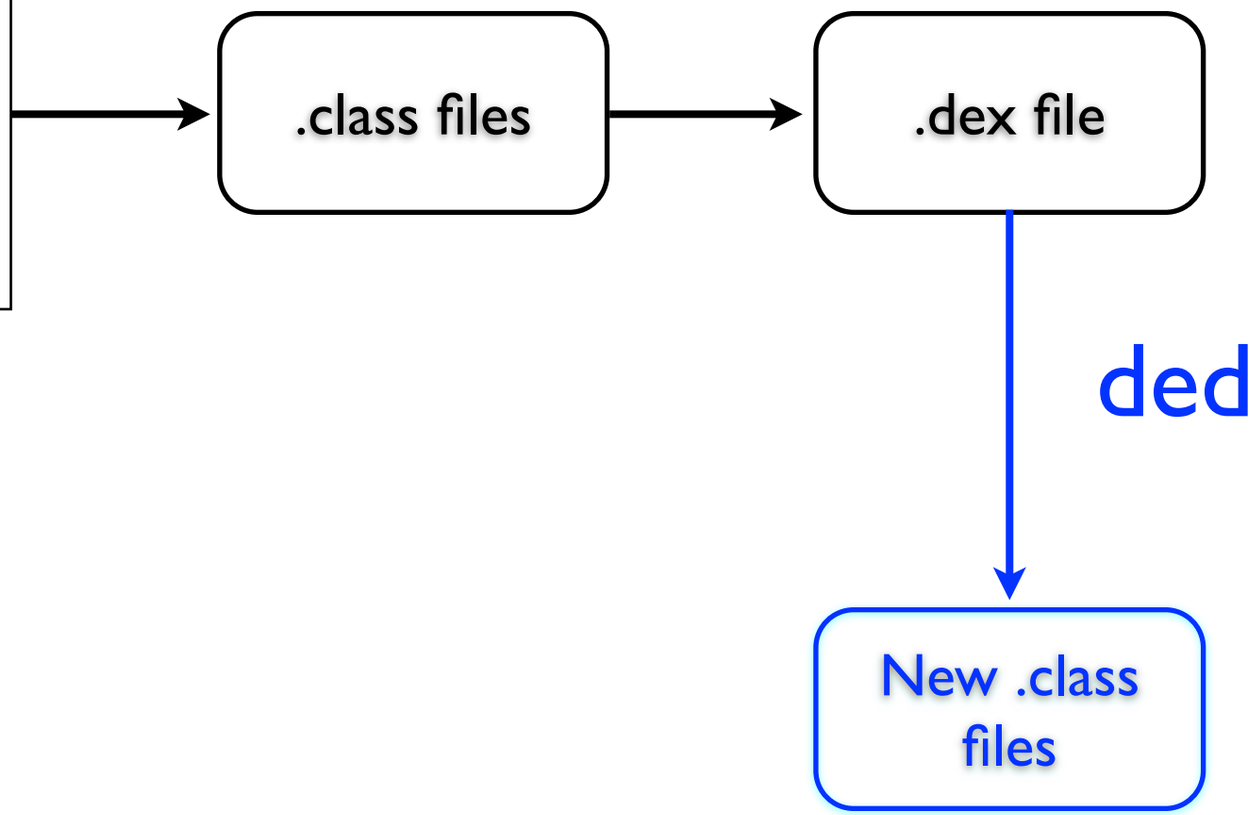
Optimization by Soot

```
public void clearTiles() {
    for (int x = 0; x < mXTileCount; x++) {
        for (int y = 0; y < mYTileCount; y++) {
            setTile(0, x, y);
        }
    }
}
```



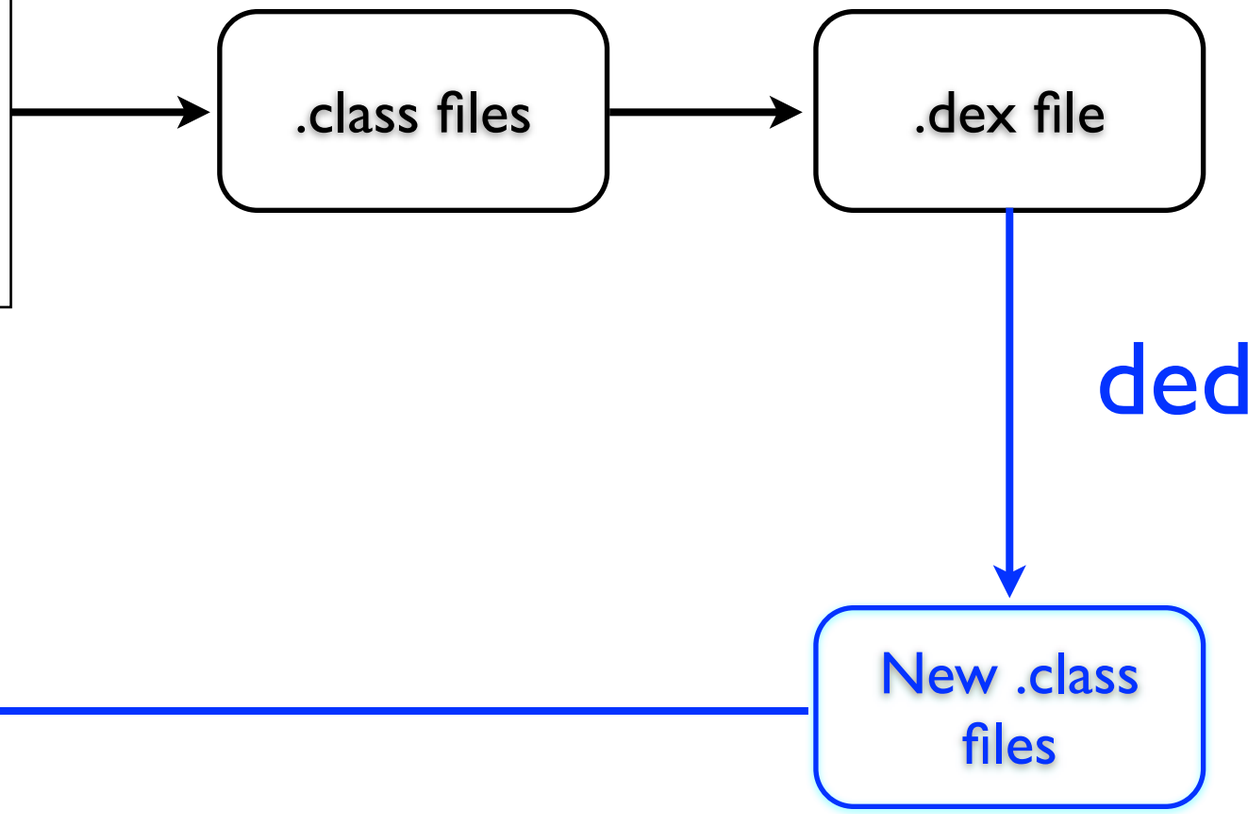
Optimization by Soot

```
public void clearTiles() {
    for (int x = 0; x < mXTileCount; x++) {
        for (int y = 0; y < mYTileCount; y++) {
            setTile(0, x, y);
        }
    }
}
```



Optimization by Soot

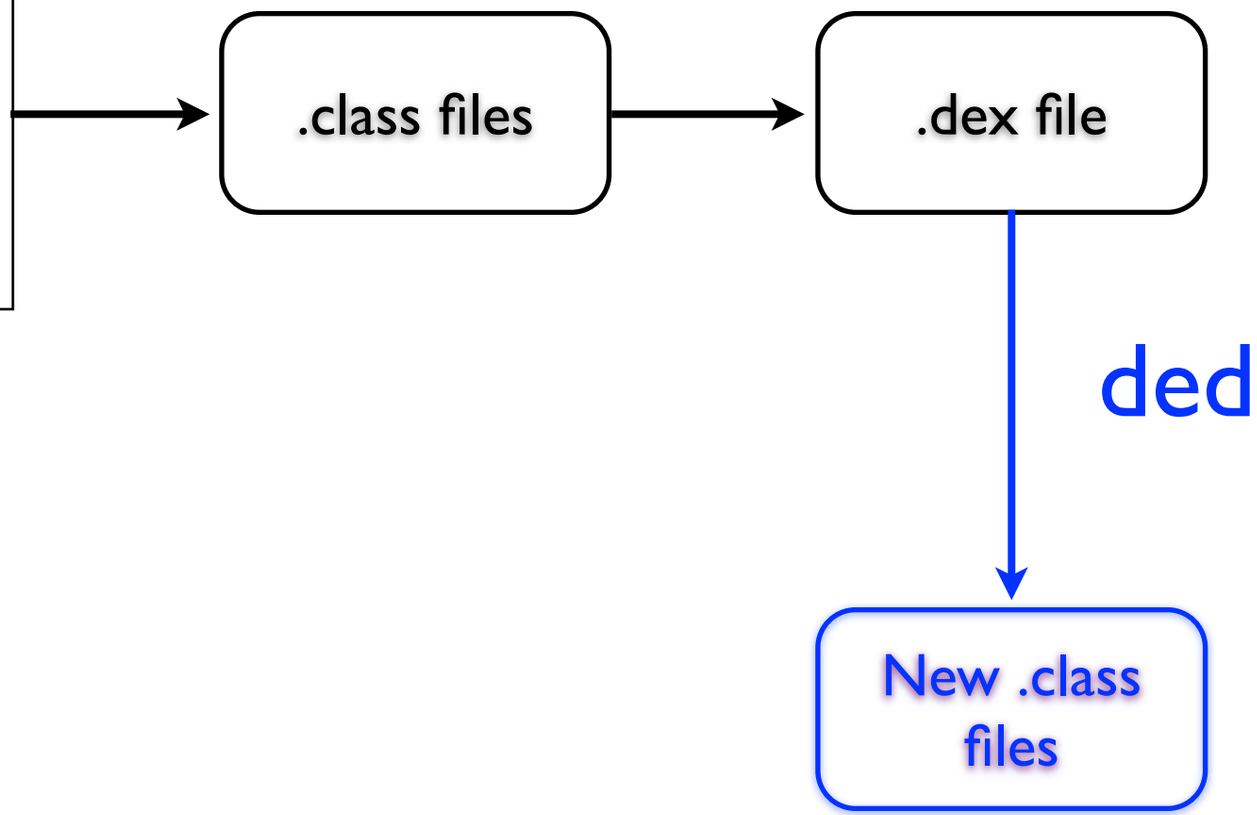
```
public void clearTiles() {
    for (int x = 0; x < mXTileCount; x++) {
        for (int y = 0; y < mYTileCount; y++) {
            setTile(0, x, y);
        }
    }
}
```




```
public void clearTiles() {
   int var1 = 0;
   while(true) {
      int var2 = mXTileCount;
      if(var1 >= var2) {
         return;
      }
      int var3 = 0;
      while(true) {
         var2 = mYTileCount;
         if(var3 >= var2) {
            ++var1;
            break;
         }
         byte var4 = 0;
         this.setTile(var4, var1, var3);
         ++var3;
      }
   }
}
```

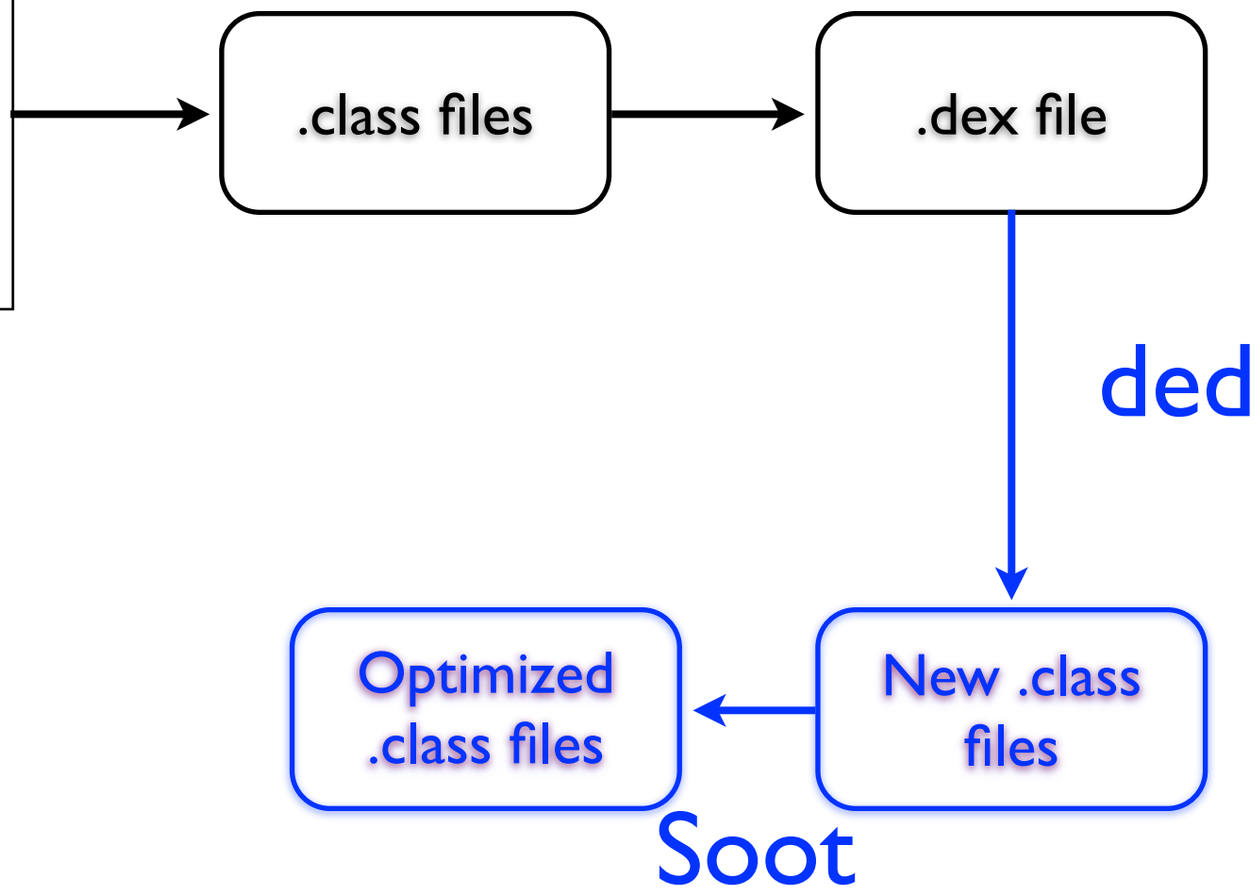
Optimization by Soot

```
public void clearTiles() {
    for (int x = 0; x < mXTileCount; x++) {
        for (int y = 0; y < mYTileCount; y++) {
            setTile(0, x, y);
        }
    }
}
```



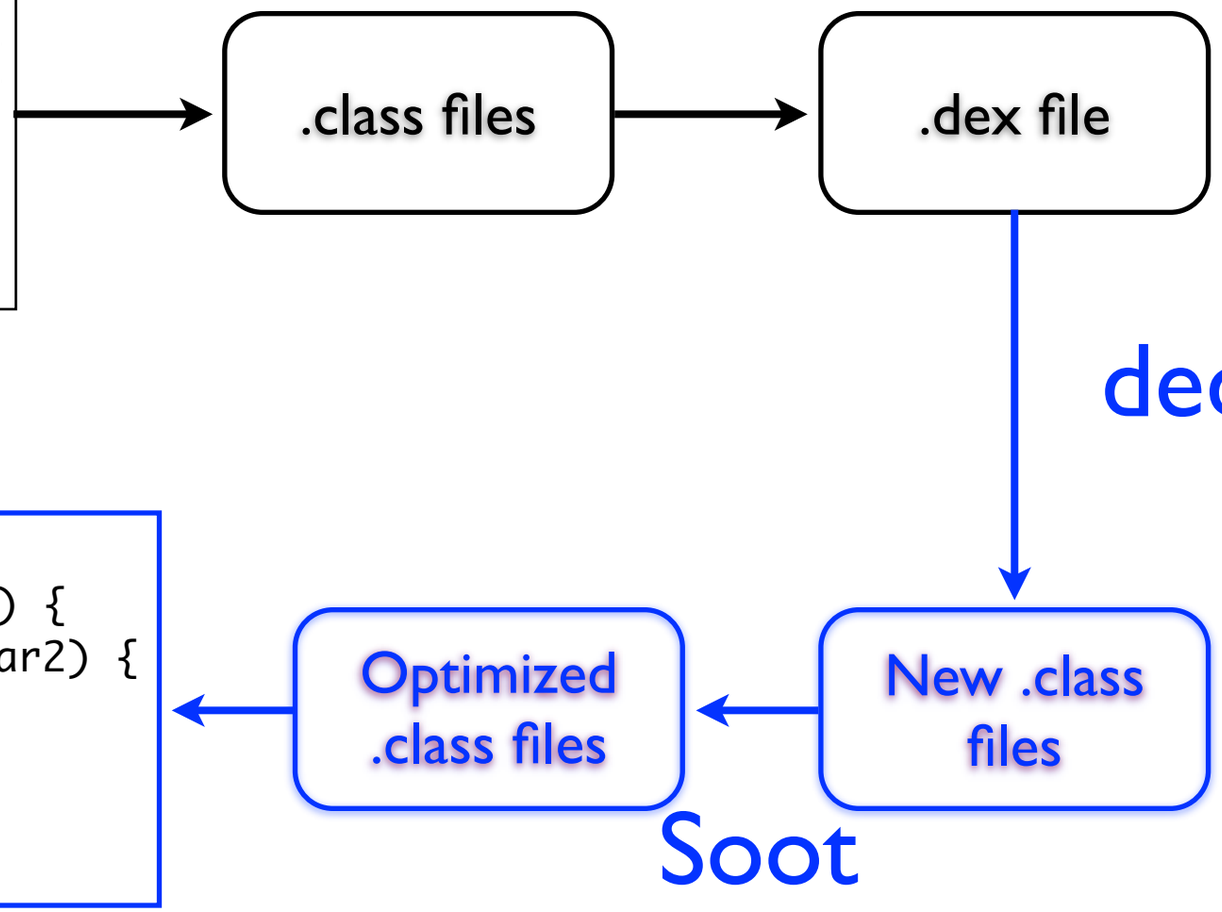
Optimization by Soot

```
public void clearTiles() {
    for (int x = 0; x < mXTileCount; x++) {
        for (int y = 0; y < mYTileCount; y++) {
            setTile(0, x, y);
        }
    }
}
```



Optimization by Soot

```
public void clearTiles() {
    for (int x = 0; x < mXTileCount; x++) {
        for (int y = 0; y < mYTileCount; y++) {
            setTile(0, x, y);
        }
    }
}
```




```
public void clearTiles() {
   for(int var1 = 0; var1 < mXTileCount; ++var1) {
      for(int var2 = 0; var2 < mYTileCount; ++var2) {
         this.setTile(0, var1, var2);
      }
   }
}
```

Studying Apps

- Decompiled top 1,100 free apps from Android market: over *21 million lines* of source code
- We use *static analysis* to identify both *dangerous behavior* and *vulnerabilities* followed by inspection
  - Must identify specific properties for analysis
  - *Note:* Static analysis says what can happen not what does

Analysis Framework

- Using Fortify SCA *custom rules* let you focus on the what, not the how
  - Control flow analysis:
    e.g., look at API options
  - Data flow analysis:
    e.g., information leaks, injection attacks
  - Structural analysis:
    "grep on steroids"
  - Semantic analysis:
    look at possible variable values

Analysis Overview

Analysis for Dangerous Behavior

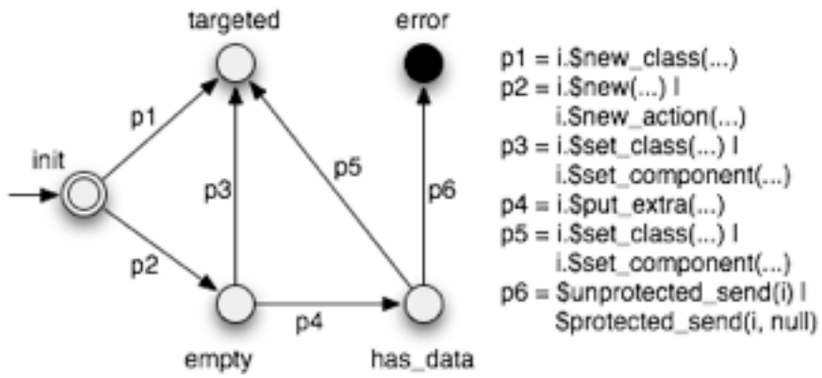
Misuse of Phone Identifiers	Data flow analysis
Exposure of Physical Location	Data flow analysis
Abuse of Telephony Services	Semantic analysis
Eavesdropping on Video	Control flow analysis
Eavesdropping on Audio	Structural analysis (+CG)
Botnet Characteristics (Sockets)	Structural analysis
Havesting Installed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



- Existing Java analysis rules aren't sufficient
- FSMs and other details in Tech Report: http://www.enck.org/pubs/NAS-TR-0144-2011.pdf

Phone Identifiers

- We've seen phone identifiers (Ph.#, IMEI, IMSI, etc) sent to network servers, but how are they used?
  - Program analysis pin-pointed *33 apps* leaking Phone IDs
- *Finding 2* - device fingerprints
- *Finding 3* - tracking actions
- *Finding 4* - along with registration and login

Device Fingerprints (1)

com.eoeandroid.eWallpapers.cartoon - SyncDeviceInfosService.getDevice_info()

```
r1.append((new StringBuilder("device_id=")).append(tm.getDeviceId()).toString()).append((new StringBuilder
("&device_software_version=")).append(tm.getDeviceSoftwareVersion()).toString());
r1.append((new StringBuilder("&build_board=")).append(Build.BOARD).toString()).append((new StringBuilder("&build_brand=")).append
(Build.BRAND).toString()).append((new StringBuilder("&build_device=")).append(Build.DEVICE).toString()).append((new StringBuilder
("&build_display=")).append(Build.DISPLAY).toString()).append((new StringBuilder("&build_fingerprint=")).append
(Build.FINGERPRINT).toString()).append((new StringBuilder("&build_model=")).append(Build.MODEL).toString()).append((new StringBuilder
("&build_product=")).append(Build.PRODUCT).toString()).append((new StringBuilder("&build_tags=")).append(Build.TAGS).toString()).append
((new StringBuilder("&build_time=")).append(Build.TIME).toString()).append((new StringBuilder("&build_user=")).append
(Build.USER).toString()).append((new StringBuilder("&build_type=")).append(Build.TYPE).toString()).append((new StringBuilder
("&build_id=")).append(Build.ID).toString()).append((new StringBuilder("&build_host=")).append(Build.HOST).toString()).append((new
StringBuilder("&build_version_release=")).append(Build$VERSION.RELEASE).toString()).append((new StringBuilder
("&build_version_sdk_int=")).append(Build$VERSION.SDK).toString()).append((new StringBuilder("&build_version_incremental=")).append(Build
$VERSION.INCREMENTAL).toString());
r5 = mContext.getApplicationContext().getResources().getDisplayMetrics();
r1.append((new StringBuilder("&density=")).append(r5.density).toString()).append((new StringBuilder("&height_pixels=")).append
(r5.heightPixels).toString()).append((new StringBuilder("&scaled_density=")).append(r5.scaledDensity).toString()).append((new
StringBuilder("&width_pixels=")).append(r5.widthPixels).toString()).append((new StringBuilder("&xdpi=")).append(r5.xdpi).toString
()).append((new StringBuilder("&ydpi=")).append(r5.ydpi).toString());
r1.append((new StringBuilder("&line1_number=")).append(tm.getLine1Number()).toString()).append((new StringBuilder
("&network_country_iso=")).append(tm.getNetworkCountryIso()).toString()).append((new StringBuilder("&network_operator=")).append
(tm.getNetworkOperator()).toString()).append((new StringBuilder("&network_operator_name=")).append(tm.getNetworkOperatorName()).toString
()).append((new StringBuilder("&network_type=")).append(tm.getNetworkType()).toString()).append((new StringBuilder
("&phone_type=")).append(tm.getPhoneType()).toString()).append((new StringBuilder("&sim_country_iso=")).append(tm.getSimCountryIso
()).toString()).append((new StringBuilder("&sim_operator=")).append(tm.getSimOperator()).toString()).append((new StringBuilder
("&sim_operator_name=")).append(tm.getSimOperatorName()).toString()).append((new StringBuilder("&sim_serial_number=")).append
(tm.getSimSerialNumber()).toString()).append((new StringBuilder("&sim_state=")).append(tm.getSimState()).toString()).append((new
StringBuilder("&subscriber_id=")).append(tm.getSubscriberId()).toString()).append((new StringBuilder("&voice_mail_number=")).append
(tm.getVoiceMailNumber()).toString());
i0 = mContext.getResources().getConfiguration().mcc;
i1 = mContext.getResources().getConfiguration().mnc;
r1.append((new StringBuilder("&imsi_mcc=")).append(i0).toString()).append((new StringBuilder("&imsi_mnc=")).append(i1).toString());
r254 = (ActivityManager) mContext.getSystemService("activity");
$r255 = new ActivityManager$MemoryInfo();
r254.getMemoryInfo($r255);
r1.append((new StringBuilder("&total_mem=")).append($r255.availMem).toString());
```

Device Fingerprints (2)

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(EncodeURL.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=").append(r24.getDeviceId()).append
("&gid=").append(QConfiguration.mGid).append("&msg=").append(QConfiguration.getInstance
().mPCStat.toMsgString()).toString();
     ...
}
```

Registration and Login

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

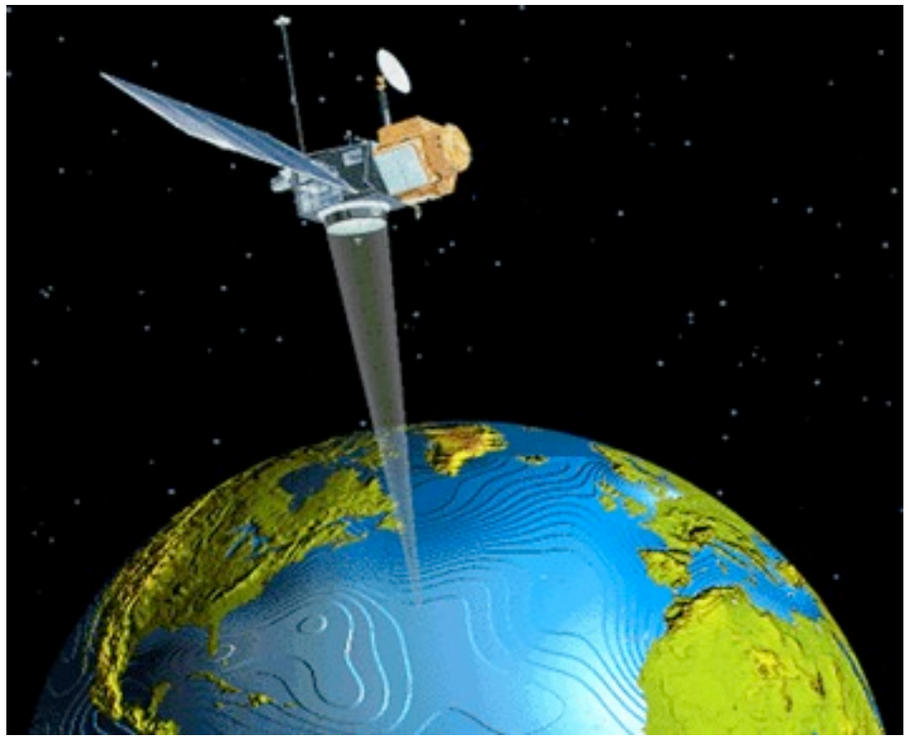
```
public void onClick(View  r1)
{
    ...
    r7 = Host.getDeviceId(this$0.getApplicationContext());
    LogInActivity.access$1(this$0).setUniqueDeviceID(r7);
    this$0.loginTask = new LogInActivity$LoginTask(this$0, null);
    this$0.showProgressDialog(r2, 2131361798, this$0.loginTask);
    r57 =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

IMEI

Is this necessarily bad?

Location

- Found *13 apps* with geographic location data flows to the network
  - Many were legitimate: weather, classifieds, points of interest, and social networking services
- Several instances sent to advertisers (same as TaintDroid). More on this shortly.
- Code recovery error in AdMob library.

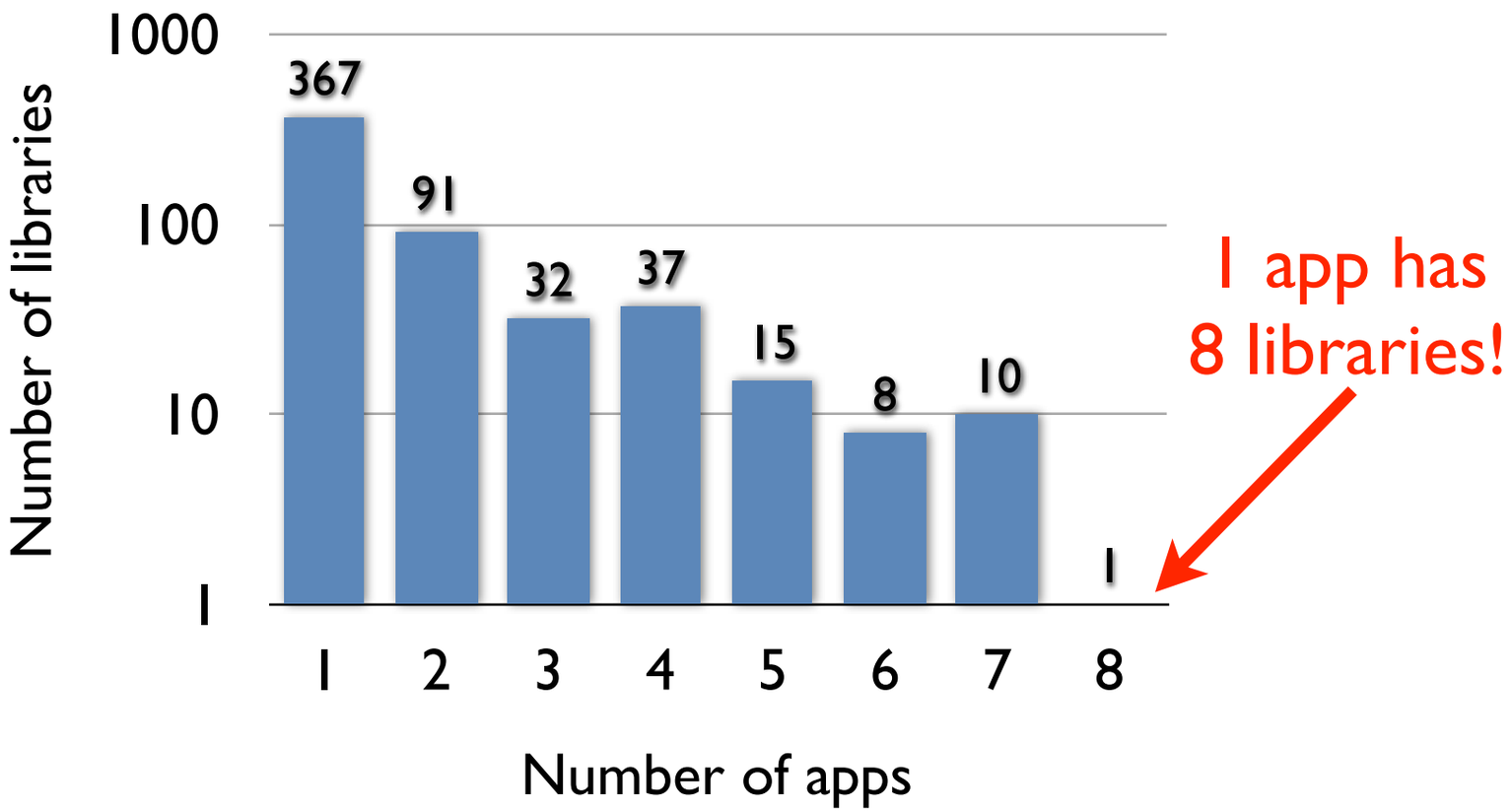


Phone Misuse

- No evidence of abuse in our sample set
  - Hard-coded numbers for SMS/voice (premium-rate)
  - Background audio/video recording
  - Socket API use (not HTTP wrappers)
  - Harvesting list of installed applications

Ad/Analytics Libraries

- 51% of the apps included an ad or analytics library (many also included 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

Probing for Permissions (1)

com/webtrends/mobile/analytics/android/WebtrendsAndroidValueFetcher.java

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

    Context r4;
    String r7;
    r4 = (Context) r0;

    try
    {
        r7 = ((TelephonyManager) r4.getSystemService("phone")).getDeviceId();

        if (r7 == null)
        {
            r7 = "";
        }
    }
    catch (Exception $r8)
    {
        WebtrendsDataCollector.getInstance().getLog().d("Exception fetching TelephonyManager.getDeviceId
value. ", $r8);
        r7 = null;
    }

    return r7;
}
```

Catches SecurityException

Probing for Permissions (2)

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

Developer Toolkits

- We found identically implemented dangerous functionality in the form of *developer toolkits*.
  - Probing for permissions (e.g., Android API, catch SecurityException)
  - Well-known brands sometimes commission developers that include dangerous functionality.
    - "USA Today" and "FOX News" both developed by Mercury Intermedia (com/mercuryintermedia), which grabs IMEI on startup

Custom Exceptions

v00032.com.wordplayer - CustomExceptionHandler.java

```
void init()
{
    URLConnection r3;
    ...
    r3 = (new URL("http://www.word-player.com/HttpHandler/init.sample")).openConnection();
    ...
    try
    {
        $r27 = this.mkStr(((TelephonyManager) _context.getSystemService("phone")).getLine1Number());
    }
    catch (Exception $r81)
    {
        break label_5;
    }
    ...
}
```

Phone Number!?

Intent Vulnerabilities

- Similar analysis rules as independently identified by Chin et al. [Mobisys 2011]
- *Leaking information to IPC* - unprotected intent broadcasts are common, occasionally contain 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

Study Limitations

- The sample set
- Code recovery failures
- Android IPC data flows
- Fortify SCA language
- Obfuscation

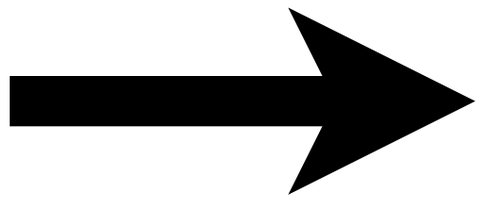
What this all means ...

- Characterization of top 1,100 free apps (21+ MLOC) similar to smaller, vertical studies (e.g., TaintDroid).
  - Development of rules to identify vulnerabilities
  - 27 Findings (more in Tech Report) providing insight into application developer behavior
  - Several APIs need more oversight
    - Phone identifiers are used in *many* different ways and are frequently sent to network servers.
    - Many developers not sensitive to Intent API dangers
  - Ad/Analytic libs in 51% -- as many as 8 in one app
    - 4th party code is becoming a problem

Future Directions

- This is all leading towards more automated certification for both {mal,gray}ware and vulnerabilities
  - App markets need transparency
- Technical Hurdles
  - Analysis framework
  - Code recovery
  - Deployment limitations

Thank You!



William Enck
Assistant Professor
Department of Computer Science
North Carolina State University
Web: http://www.enck.org
Email: enck@cs.ncsu.edu