Privacy Violations Detection in Android like Systems

Nora Cuppens
Directrice de Recherche

Colloque IMT’2017, November 10th
SIRI - TELL ME YOUR INNERMOST SECRETS - JUST THIS ONCE.

Sorry, G-man, but if I did that, I wouldn't be a smart phone, would I?

Privacy Settings Tip:

See where it says “AUTOMATICALLY SHARE MY PERSONAL INFORMATION WITH IDENTITY THIEVES, SEX OFFENDERS AND ALL MY PSYCHO EXES”? Yeah, you’re gonna wanna unclick that box.
Why targeting Android?

- 2009-2010
  - Malware and Market Share Correlation

- 2010
  - Malware and Market Share Correlation

- 2011
  - Malware and Market Share Correlation

- 2017
  - 71%-87% market share
  - 2.7+ billion apps, 70+ billion downloads (Src: Google)
  - 1M+ Android devices activated everyday (Src: Google)

- Ideal platform for security research

* Juan Tapiador
Informal problem statement:
– How invasive Android Apps are?

- Uber: knows everywhere you go
  • Tracking customers

- Whisper, Yik-Ya: supposedly anonymous
  • “De-anonymizing users and take control of the account …”

- Angry Birds: only a game?
  • User profiling

- Snapchat: self-destructing photo app that doesn't
  • Hacked and lost a database of several million usernames connected to phone numbers.

- Brightest Flashlight: flashlight apps
  • Exploiting their phone's internet connection in order to deliver targeted advertising
Informal problem statement:
– Data leakage / Privacy loss

- Android type system offers a nominal security solutions

- Progress has been made in this area
  - Access control
  - Data flow control

- Our Research works
  - Solving under-tainting problem
  - Detecting flows in JNI
  - Dealing with side channel attacks
  - Detecting / Reacting activity hijacking
Couple of reminders:
- Android System Architecture and Security

- System Applications
- User Applications
- IPC reference monitor
- Sandboxing
- Permission levels
- Secure boot
- Secure file system
- Native executables protection
- Discretionary AC
- Application Sandbox
Couples of reminders:
– Security in the Application Level

Authorization Checks
Application framework level
Hooks

appli 1

Permission_1

appli 2

Sensitive
Component_a / Permission_1

Component_c / Default

Sensitive
Component_b / Permission_2
Some Weaknesses of Android Security Model

- Revocation limits
- Few sources for applications, warnings about security implications displayed during run-time
- Flawed permissions model
- Malware obfuscated inside legitimate-looking applications
- Google play store: insufficient control
- Applications isolation: malicious k-ary applications
- Tricky problem of Patching / Updating
Android Security
– Enhanced solutions

Access Control
Data Flow Control
Access Control in Android Systems – The progress

- **Applications certification**
  - Kirin (Enck et al.)
  - Avoid manual certification by code inspection (SymbianSigned, Apple)
  - Provide lightweight certification based on predefined rules at install-time

- **Application access control policy at install**

- **Application inter-communications security policy at execution**
  - [Saint (Ongtang et al.)]
  - Managing authorization assignments and their use at run-time
  - In accordance with the application provider policy

- **Dynamic control of permissions granted to applications**
  - [Apex (Nauman et al.)]
  - The user chooses the permissions to be granted to the applications and imposes constraints on the use of resources
Access control is not sufficient

- ... of course
- Does not address the data flow problem
A command sequence implies an information flow from $x$ to $y$ if the value of $y$ after performing this sequence makes it possible to infer information on the value of $x$ before the execution of this sequence.

```plaintext
boolean b := <secret>
if (b) {
x := true; f();
```

Information flow from $b$ to $x$
Data flow control: static analysis – Reminder

- Analyzing the code without executing it
- Performed at the install-time or compile-time
- Performed on the source or on the bytecode
| Static analysis of Dalvik bytecode of applications | Tracking flows between URIs to generate constraints on permissions | Requiring the source code, Packaged applications are not considered | ScanDroid [Fuchs et al.]
---|---|---|---
| Analyzing applications before making them available | Analysis of decompiled DEX files to discover vulnerabilities based on intents exchanges | Secure communication: No formal guaranties | ComDroid [Chin et al.]

Only Explicit flows are considered
Data flow control: dynamic analysis
– Reminder

- Instrumentation of the code before its execution
- Analysis performed at run-time
- Binary code Tracking
Data flow control:  
– “Tainting” based dynamic analysis

- TaintDroid [Enck et al.]
- Tainting
  - Technique for tracing dependencies of information from a given point

\[
x = \text{taint} ()
\]
...
\[
y = z + x
\]
...
\[
\text{Sent}_\text{Net}(y)
\]
### "Tainting" propagation logic

#### Examples

<table>
<thead>
<tr>
<th>Op Format</th>
<th>Op Semantics</th>
<th>Taint Propagation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const-op $v_A$ $C$</td>
<td>$v_A \leftarrow C$</td>
<td>$\tau(v_A) \leftarrow \emptyset$</td>
<td>Clear $v_A$ taint</td>
</tr>
<tr>
<td>move-op $v_A$, $v_B$</td>
<td>$v_A \leftarrow v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
<tr>
<td>move-op $K$, $v_A$, $v_B$</td>
<td></td>
<td></td>
<td>Set $v_A$ taint to return taint</td>
</tr>
<tr>
<td>unary-op $v_A$, $v_B$</td>
<td>$v_A \leftarrow \otimes v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
<tr>
<td>binary-op $v_A$, $v_B$, $v_C$</td>
<td>$v_A \leftarrow v_B \otimes v_C$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B) \cup \tau(v_C)$</td>
<td>Set $v_A$ taint to $v_B$ taint $\cup$ $v_C$ taint</td>
</tr>
<tr>
<td>binary-op $v_A$, $v_B$</td>
<td>$v_A \leftarrow v_B \otimes v_B$</td>
<td>$\tau(v_A) \leftarrow \tau(v_A) \cup \tau(v_B)$</td>
<td>Update $v_A$ taint with $v_B$ taint</td>
</tr>
<tr>
<td>binary-op $v_A$, $v_B$, $C$</td>
<td>$v_A \leftarrow v_B \otimes C$</td>
<td>$\tau(v_A) \leftarrow \tau(v_B)$</td>
<td>Set $v_A$ taint to $v_B$ taint</td>
</tr>
<tr>
<td>aput-op $v_A$, $v_B$, $v_C$</td>
<td>$v_B[v_C] \leftarrow v_A$</td>
<td>$\tau(v_B) \leftarrow \tau(v_B) \cup \tau(v_A)$</td>
<td>Update array $v_B$ taint with $v_A$ taint</td>
</tr>
<tr>
<td>aget-op $v_A$, $v_B$, $v_C$</td>
<td>$v_B[v_C] \leftarrow v_A$</td>
<td>$\tau(v_B) \leftarrow \tau(v_B) \cup \tau(v_A)$</td>
<td>Update array $v_B$ taint with $v_A$ taint</td>
</tr>
<tr>
<td>sput-op $v_A$, $f_B$</td>
<td>$v_A \leftarrow f_B$</td>
<td>$\tau(v_A) \leftarrow \tau(f_B)$</td>
<td>Set field $f_B$ taint to $v_A$ taint</td>
</tr>
<tr>
<td>sget-op $v_A$, $f_B$</td>
<td>$v_A \leftarrow f_B$</td>
<td>$\tau(v_A) \leftarrow \tau(f_B)$</td>
<td>Set field $f_B$ taint to $v_A$ taint</td>
</tr>
</tbody>
</table>

\[
\tau(\quad) \quad \tau(\quad)
\]

\[
\tau(v_A) \quad \tau(v_B)
\]
Data flow control: dynamic analysis, – Limits

- False negatives

- Management of data flows
  - Explicit flows
    \[ y = x \]

- Do not consider control flows
  - Implicit flows
    \[
    \text{if (} x \text{)} \\
    y = \text{true} \\
    \text{else} \\
    y = \text{false}
    \]
Control dependencies attack

1. X = false
2. Y = false
3. char c[256];
4. If( gets(c) != user_contact )
5. X = true;
6. else
7. Y = true;
8. NetworkTransfer(x);
9. NetworkTransfer(y);

Data leakage
Our Research topics
– Solving “under-tainting” problem

- Leakage using flow control
  - Mariem Graa, Nora Cuppens-Boulahia, Frédéric Cuppens, Ana R. Cavalli: Detecting Control Flow in Smartphones: Combining Static and Dynamic Analyses. CSS 2012

- Code obfuscation
Solving “under-tainting” problem

- Launching the app
- CFG & Basic blocs creation
- Detecting flows in “IF” dependencies
- IF tainting
- Detecting modified variables
- Tainting of modified variables
New tainting propagation rules

\[(x \rightarrow y) \Rightarrow (Taint(y) \leftarrow Taint(x))\]

\[(x \leftarrow y) \Rightarrow (y \rightarrow x)\]

\[(Taint(x) \leftarrow Taint(y)) \land (Taint(x) \leftarrow Taint(z))\]

\[\Rightarrow (Taint(x) \leftarrow Taint(y) \oplus Taint(z))\]

\[
\frac{Is\ modified(x) \land Dependency(x,\ condition) \land BranchTaken(br,\ conditional\ statement)}{Taint(x) \leftarrow Context\_Taint \oplus Taint(explicit\ flow\ statement)}
\]

\[
\frac{Is\ assigned(x, y) \land Dependency(x,\ condition) \land \neg BranchTaken(br,\ conditional\ statement)}{Taint(x) \leftarrow Taint(x) \oplus Context\_Taint}
\]
Obfuscated code

IMEI (International Mobile Equipment Identity)

Dynamic analysis

1. X ← User_Location
2. NetworkTransfer(X);

3. For each symbol in AsciiTable do
4.   If(symbol = x then)
5.     Y ← Y + symbol
6.   end if
7. end for
8. NetworkTransfer(Y);
Obfuscated code: Solved!

1. \( X \leftarrow \text{User\_Location} \)
2. For each \( x \) in \( X \) do
3.   For each symbol in \( \text{AsciiTable} \) do
4.     If (symbol = \( x \) then)
5.         \( Y \leftarrow Y + \text{symbol} \)
6.     end if
7. end for
8. end for
9. \( \text{NetworkTransfer}(Y); \)
Our Research topics:

- Detecting flows in native codes

Instrumenting JNI code to avoid sensitive data leakage


```java
package com.tuto.attackndk;
public class MainActivity extends Activity {
    static {
        System.loadLibrary("attackndk");
    }
    public static native void
        invokeNativeFunction(String IMEI);
    @Override
        protected void onCreate(Bundle savedInstanceState) {

            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            String device_id = GetDeviceId();
            invokeNativeFunction(device_id);
        }
```

```c
#include <string.h>
#include <jni.h>

VoidJava_com_tuto_attackndk_MainActivity_invokeNativeFunction(JNIEnv* env, jobject thiz, jstring IMEI){
    String Private_Data;
    String Z;
    strcpy(Private_Data, IMEI);
    for(int i = 0; i < sizeof(Private_Data); i++)
    {
        char s;
        sprintf(s, "%d", i);
        for(int j = 1; j < sizeof(TabAsc); j++)
        {
            if(strcmp(s,TabAsc[j]) == 0)
            {
                strcat(Z,TabAsc[j]);
            }
        }
    } NetworkTransfer(Z);
```

Attack exploiting native code

Native malicious function
Our Research topics

- Side channel attacks

**Dealing with different side channel attacks**

- Mariem Graa, Nora Cuppens-Boulahia, Frédéric Cuppens, Jean-Louis Lanet, Routa Moussaileb: Detection of Side Channel Attacks Based on Data Tainting in Android Systems. SEC 2017

```
X ← Private_Data
for each x ∈ X do
    n ← CharToInt(x)
    StartTime ← ReadSystemTime()
    Wait(n)
    StopTime ← ReadSystemTime()
    y ← (StopTime - StartTime)
    Y ← Y + IntToChar(y)
end for
Send_Network_Data(Y)
```

**Timing attack example**

- Enrich the tainting policy rules
- The system clock is sensitive
Our Research topics
– Hijacking attacks

- Detecting activity hijacking

- Android users Activities require to communicate sensitive data
  - passwords, security codes, and credit card numbers) with applications

- Hacker can launch hijacking attacks to compromise user’s data confidentiality / privacy

- [Chen et al.] stealthily inject into the foreground a hijacking activity at the right timing and steal sensitive information in Android smartphones
Our Research topics
– Hijacking attacks

- The core security mechanisms of Android cannot detect activity hijacking

- Some not satisfactory existing work
  - [Malisa et al.] and [Sun et al.] analyse application resource files (XML layout) to detect similarity of UI

- Our proposal
  - Modify the Android operating system
  - Extract and compare UI elements features of the legitimate and the hijacking interfaces
  - Use the indistinguishability level between the attack and legitimate Activities
  - Reacting: blocking or alerting
Our Research topics
– Hijacking attacks

- False positives
  - 4.2% in the case of partial indistinguishability
  - $10^{-3}$% in the case of full indistinguishability

- **Performance:** 0.39% performance overhead on a CPU-bound micro-benchmark
Conclusion

- Malicious and behaviour in smartphone platforms has evolved significantly in the last decade
  - Android particularly

- It currently target Internet of Things devices
  - Many open research problems in this context
    - Privacy of course,
    - But also Trust and Security that need to be revised
Confidentiality violations
detection in Android systems

Nora Cuppens
Directrice de Recherche

Colloque IMT’2017, November 10th