Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis

Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, Giovanni Vigna

NDSS’17
02-28-2017
Mobile Privacy Leak Detection

● Mobile apps are known to leak private information over the network (e.g., IMEI, Location, Contacts)

● Researchers developed approaches to detect them
  ○ Static taint analysis
  ○ Dynamic taint analysis
Mobile Privacy Leak Detection

- Mobile apps are known to leak private information over the network (e.g., IMEI, Location, Contacts)
- Researchers developed approaches to detect them
  - Static taint analysis
  - Dynamic taint analysis
- Recently, network-based detection
  - Leaked values need to flow through the network
http://i.w.inmobi.com/showad.asm?u-id-map=iB7WTkCLJvNsaEQakKKXFhk8ZEIZlnL0jqbbYexcBAXYHH4wSKyCDWVfp+q+FeLFTQV6jS2Xg97liEzDkw+XNTghe9ekNyMnjypmiu7xBS1TcwZmFxYOjJkgP OzkI9j2lryBaLlAJBSDkEqZeMVvcjcNkx+Ps6SaTRzBbYf8UY=&u-key-ver=2198564
Motivation

// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId", Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextInt();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
Motivation

// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId", Context.class).invoke(context);
MessageDigest shal = getInstance("SHA-1");  // hash
shal.update(aid.getBytes());
byte[] digest = shal.digest();

Random random = new Random();  // generate random key
int key = random.nextInt();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
Motivation

// get Android ID using the Java Reflection API
String aid = class.getMethod("getAndroidId", Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextInt();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write("key=" + key.getBytes());
// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId", Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextInt();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);
String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
Motivation

// get Android ID using the Java Reflection API
String aid = Class.getDeclaredMethod("getAndroidId", Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextInt();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic
Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic

IMEI: 12345678  →  APP
Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic

IMEI: 12345678 → APP → http://host.com/?id=39979edb58
Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption

- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic

IMEI: 12345678 → APP → http://host.com/?id=39979edb58
IMEI: 98765432 → APP
Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic

IMEI: 12345678 → APP → http://host.com/?id=39979edb58

IMEI: 98765432 → APP → http://host.com/?id=bae6a29c9b
Not so easy...

- Network traffic is **non-deterministic**
- The output **changes** even if you don't change the source
- Cannot pin a change in the output to a specific change in the input
We found that non-determinism can be often explained and removed, making differential analysis possible.
Sources of Non-Determinism
Sources of Non-Determinism

Random values
Sources of Non-Determinism

Random values
Timing values
Sources of Non-Determinism

Random values  Timing values  Network values
Sources of Non-Determinism

Random values

Timing values

Network values

System values
Sources of Non-Determinism

- Random values
- Timing values
- Network values
- System values
- Encryption
Sources of Non-Determinism

- Random values
- Timing values
- Network values
- System values
- Encryption
- Executions
Contextual Information

- Eliminate and explain non-determinism by **recording** and **replacing** non-deterministic values (either with previously seen or constant values)
  - Record and replay timestamps
  - Record random identifiers (UUID)
  - Record ptx and ctx during encryption
  - Set fixed seed for random num generation functions
  - Set values of performance measures to constants
Contextualized Trace

Network Trace

https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69
https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c
https://auth.domain.com/user/sign

Contextual info

UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]
Timestamps: [146897456, 146897562]
Decryption map: {"7aca67bfc75d7816a1d907fb834c8f69"=>"146897456_c734f4ec"}

Contextualized Trace

https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign
Contextualized Trace

Network Trace

https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69
https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c
https://auth.domain.com/user/sign

Contextual info

UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]
Timestamps: [146897456, 146897562]
Decryption map: {"7aca67bfc75d7816a1d907fb834c8f69":"146897456_c734f4ec"}

Contextualized Trace

https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign
Contextualized Trace

Network Trace
https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69
https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c
https://auth.domain.com/user/sign

Contextual info
UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]
Timestamps: [146897456, 146897562]
Decryption map: "7aca67bfc75d7816a1d907fb834c8f69" => "146897456\_c734f4ec"

Contextualized Trace
https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign
Contextualized Trace

Network Trace

https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69
https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c
https://auth.domain.com/user/sign

Contextual info

UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]
Timestamps: [146897456, 146897562]
Decryption map: "{7aca67bfc75d7816a1d907fb834c8f69}"="146897456_c734f4ec"

Contextualized Trace

https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign
Agrigento: High-level Overview

Instrumented Environment

#1 Run

App

Sources of Leak
Agrigento: High-level Overview
Agrigento: High-level Overview

Instrumented Environment

#1 Run

App
Sources of Leak

Contextualized Trace

Network Trace
Contextual Info

... ...

Instrumented Environment

#n Run

App
Sources of Leak

Contextualized Trace

Network Trace
Contextual Info
Agrigento: High-level Overview

Phase 1: Network Behavior Summary Extraction
Agrigento: High-level Overview

Phase 1: Network Behavior Summary Extraction

Phase 2: Differential Analysis
Agrigento: High-level Overview

Phase 1: Network Behavior Summary Extraction

Phase 2: Differential Analysis
Agrigento: High-level Overview

Phase 1: Network Behavior Summary Extraction

Phase 2: Differential Analysis

A. Continella et al. Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis
Agrigento: High-level Overview

**Conservative, fail-safe approach:**
We flag any differences we cannot explain

Phase 1: Network Behavior Summary Extraction

Phase 2: Differential Analysis

---

A. Continella et al. Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis
Number of Runs

- **Automatically** determine number of executions
- After each run, differential analysis *without* any source modification
- An app reaches **convergence** when there are no diffs in the network for $K$ consecutive runs
System Architecture

A. Continella et al. Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis
Experimental Setup & Datasets

- **Setup**
  - Six Nexus 5 running Android 4.4.4
  - 10 mins execution per app, Monkey for UI stimulation (fixed seed)

- **Datasets**
  - 100 most popular free apps across all the categories from the Google Play Store in June 2016
  - 100 randomly selected less popular apps
  - 750 apps from ReCon dataset
  - 54 apps from BayesDroid dataset
Non-Determinism in Network Traffic

- Top 100 Google Play apps from the ReCon dataset
- % of apps with non-deterministic network traffic

---

![Graph showing non-determinism in network traffic](image)

- Leveraging contextual information
- Trivial differential analysis
## Comparison with Existing Tools

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Tool (Approach)</th>
<th>#Apps detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReCon</td>
<td>FlowDroid (Static taint analysis)</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Andrubis/TaintDroid (Dynamic taint analysis)</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>AppAudit (Static &amp; dynamic taint flow)</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>ReCon (Network flow analysis)</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>AGRIGENTO</td>
<td>278</td>
</tr>
<tr>
<td>ReCon</td>
<td>ReCon (Network flow analysis)</td>
<td>229</td>
</tr>
<tr>
<td>(same flows)</td>
<td>AGRIGENTO</td>
<td>278</td>
</tr>
<tr>
<td>BayesDroid</td>
<td>BayesDroid (Bayesian reasoning)</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>AGRIGENTO</td>
<td>21</td>
</tr>
</tbody>
</table>

Agrigento detected many more apps & we manually verified most of them were true positives!
Privacy Leaks in Popular Apps

- Top 100 apps from the Google Play Store (July 2016)
- We classified the type of leak in three groups:
  - plaintext, encrypted, obfuscated
- Agrigento identified privacy leaks in 46 of the 100 apps
  - 42 true positives, 4 false positives

<table>
<thead>
<tr>
<th>Results</th>
<th>Any</th>
<th>Android ID</th>
<th>IMEI</th>
<th>MAC Address</th>
<th>IMSI</th>
<th>ICCID</th>
<th>Location</th>
<th>Phone Number</th>
<th>Contacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPs</td>
<td>31</td>
<td>30</td>
<td>13</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Plaintext</td>
<td>22</td>
<td>18</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Encrypted</td>
<td>11</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Obfuscated</td>
<td>42</td>
<td>38</td>
<td>22</td>
<td>11</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FPs</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>13</td>
<td>11</td>
<td>16</td>
<td>13</td>
</tr>
</tbody>
</table>
Case Study: ThreatMetrix

https://h.online-metrix.net/fp/clear.png?ja=33303426773f3a3930643667663b33383831303d343526613f2d363830247a3f363026663d333539347a3132383266603d687c7672253163253066253066616f6e74656e766c6f7926636f6524666723f6a747670253161273266253266616d6d2e6561643761373261363a61616335636761266d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39353166633036666361373261363a616163356367612666d66733f353b32306d383230613230643b6534643934336303b64336a39
Case Study: ThreatMetrix

1. IMEI, Location, MAC address ~> HashMap
2. **XOR** HashMap with a *randomly* generated key
3. Hex-encode HashMap
4. Send obfuscated HashMap & random key

https://h.online-metrix.net/fp/clear.png?ja=33303426773f3a3930643667663b33383831303d343526613f2d363830247a3f363026663d333539347a3132383266c603d87c7672253163253066253066616f6e74656e762f6a732c746370626f7926663f652466723f6a74667025316127326625326661646d62e6561633631333131333131266564643f65616336313331313331312661743772777174666566676e6665722b6d6f606b6c652733632b392e3226342d3b...
Limitations & Future Work

- Limited code coverage
- Covert channels
- No native code instrumentation
  - We use a conservative approach: FP in worst case
- Only HTTP(S) GET and POST
- Investigate malicious intents behind obfuscation
Conclusions

● Non-Determinism in network traffic can be often explained and removed

● Agrigento can detect privacy leaks using a black-box, obfuscation-resilient approach

● Apps and ad libraries hide their information leaks using different types of encoding and encryption

https://github.com/ucsb-seclab/agrigento
Thank you!
Questions?

andrea.continella@polimi.it

@_conand

https://github.com/ucsb-seclab/agrigento