Finding Clues For Your Secrets:
Semantics-Driven, Learning-Based
Privacy Discovery in Mobile Apps

Yuhong Nan, Zhemin Yang, Yuan Zhang,
Donglai Zhu and Min Yang

Fudan University

Xiaofeng Wang

Indiana University Bloomington
Today's Mobile Apps

• **Multiple web services integration**
  • Ad services, social SDKs, development tools, etc.

• **Privacy implications**
  • Ability to collect user data. E.g., Pluto [NDSS'16]
  • Ability to associate user activities, infer user secrets. E.g., Linkdroid [Security'15]
Motivation

An OSN sharing syndicator (SDK) for post-sharing
Motivation

Collecting user’s detailed profile and shared content...
## Motivation

<table>
<thead>
<tr>
<th>App Info</th>
<th>Social Network Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>top-task app list, app start timestamp,</td>
<td>weibo id, nick name, true name, verified reason,</td>
</tr>
<tr>
<td>app end timestamp, new install app,</td>
<td>gender, sns url, resume, friendlist, shared posts,</td>
</tr>
<tr>
<td>new uninstall app info, etc.</td>
<td>latitude, longitude, liked posts, etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weibo</th>
<th>Facebook</th>
</tr>
</thead>
<tbody>
<tr>
<td>weibo id, nick name, true name, verified</td>
<td>facebook id, nickname, gender, birthday, sns-url,</td>
</tr>
<tr>
<td>reason, gender, sns url, friendlist (including</td>
<td>friend list (including accessible friend info),</td>
</tr>
<tr>
<td>accessible friend info), verify status,</td>
<td>verify status, education (school name, type, year),</td>
</tr>
<tr>
<td>education (school name, type, year), work</td>
<td>work (company, employer, start &amp; end date), etc.</td>
</tr>
<tr>
<td>(company, employer, start &amp; end date), etc.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Others</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tumblr, dropbox, pinterest, line, tencent</td>
<td></td>
</tr>
<tr>
<td>qq, tencent qzone, wechat (friend list),</td>
<td></td>
</tr>
<tr>
<td>twitter, net-ease microblog, evernote, google+</td>
<td></td>
</tr>
</tbody>
</table>

Preliminary question: sensitive data identification
Automated leakage analysis

• System-controlled privacy
  • Fixed list of APIs
    - Location ✓ LocationManager.getLastKnownLocation()
    - Contact ✓ ContentResolver.query(CONTACT_URI)
    - SMS ✓ SmsMessage.getMessageBody()
    - Phone Number ✓ TelephonyManager.getLine1Number()
    - ...

• User-input privacy (UIP data)
  • UI-based identification.
    • SUPOR, UIPicker [USENIX Security’15]

• UIPicker: User-Input Privacy Identification in Mobile Applications
• SUPOR: Precise and Scalable Sensitive User Input Detection for Android Apps
Automated leakage analysis

• Server-side sensitive data

  • UI or System API-based Labelling?
    • Go through system API without specific characters

  • Network Communication?
    • Difficult to capture network traffic at a large scale with runtime analysis
      • E.g., a valid login for each app
Observation

• Finding **clues** from app code
  • Preserved semantics
Our Work

• **ClueFinder**
  • New technique for *sensitive data source discovery* from app code
    ✓ System APIs
    ✓ User interfaces
    ✓ Server-side sensitive data

• Large-scale exposure risk analysis for third-party libraries in Android apps
  • 445,688 apps from multiple app stores
  • New findings
Technical Challenges

• Ambiguity of text strings in app code
  • com/tencent/padqq/activity/AddFriendListActivity
  • UserProfileUri
  • is_mobile_phone_valid

• Privacy-related strings != Sensitive Data
  • Log.e ( "Username is null, check valid user input.."")
  • XXX.setContentTitle(" Your current Location: ");
## Getting user profile on Facebook

```java
JsonObject getUserFbProfile(HashMap userBasicInfo) {
    JsonObject userJson = UserBasicInfo.toJsonObject();
    if (userJson.containsKey("home_addr")) {
        jsonObject.put("home_addr", this.homeAddr);
    }
    this.uri = jsonObject.get("userProfile_uri");
    if (this.uri == null) {
        throw NullPointerException("Profile URI is null", exception);
    }
    return jsonObject;
}
```

## Sharing content to Facebook

```java
Builder shareToFacebook(String shareContent) {
    Builder builder = new Builder();
    builder.setContentTitle("I'm designing my own tees on my phone!");
    builder.setContentUrl(Uri.parse("https://snaptee.co/getapp"));
    builder.setShareContent(shareContent);
    Log.d("FacebookFunctions", "Try to invite FB");
    return builder;
}
```
## Getting user profile on Facebook

```java
JsonObject getUserFbProfile(HashMap<UserBasicInfo> userBasicInfo) {
    JsonObject userJson = UserBasicInfo.toJson();
    if (userJson.containsKey("home_addr")) {
        JsonObject.put("home_addr", this.homeAddr);
        this.uri = jsonObject.get("userProfile_uri");
        if (this.uri == null) {
            throw new NullPointerException("Profile URI is null");
        }
    }
    return jsonObject;
}
```

## Sharing content to Facebook

```java
Builder shareToFacebook(String shareContent) {
    Builder builder = new Builder();
    builder.setContentTitle("I'm designing my own tees on my phone!");
    builder.setContentUrl(Uri.parse("https://snaptee.co/getapp"));
    builder.setShareContent(shareContent);
    Log.d("FacebookFunctions", "Try to invite FB");
    return builder;
}
```
## Getting user profile on Facebook

```java
JsonObject getUserFbProfile(HashMap userBasicInfo) {
    JsonObject userJson = UserBasicInfo.toJson();
    if (userJson.contains("home_addr")) {
        jsonObject.put("home_addr", this.homeAddr);
    }
    this.uri = jsonObject.get("userProfile_uri");
    if (this.uri == null) {
        throw new NullPointerException("Profile URI is null");
    }
    return jsonObject;
}
```

## Sharing content to Facebook

```java
Builder shareToFacebook(String shareContent) {
    Builder builder = new Builder();
    builder.setContentTitle("I'm designing my own tees on my phone!");
    builder.setContentUrl(Uri.parse("https://snaptee.co/getapp"));
    builder.setShareContent(shareContent);
    Log.d("FacebookFunctions", "Try to invite FB");
    return builder;
}
```
Getting user profile on Facebook

```java
JsonObject getUserFbProfile(HashMap userBasicInfo) {
    JsonObject userJson = UserBasicInfo.toJson();
    if (userJson.containsKey("home_addr")) {
        jsonObject.put("home_addr", this.homeAddr);
    }
    this.uri = jsonObject.get("userProfile_uri");
    if (this.uri == null) {
        throw new NullPointerException("Profile URI is null");
    }
    return jsonObject;
}
```

Sharing content to Facebook

```java
Builder shareToFacebook(String shareContent) {
    Builder builder = new Builder();
    builder.setContentTitle("I'm designing my own tees on my phone!");
    builder.setContentUrl(Uri.parse("https://snaptee.co/getapp"));
    builder.setShareContent(shareContent);
    Log.d("FacebookFunctions", "Try to invite FB");
    return builder;
}
```
## Getting user profile on Facebook

```java
JsonObject getUserFbProfile(HashMap userBasicInfo) {
    JsonObject userJson = UserBasicInfo.toJson();
    if (userJson.containsKey("home_addr")) {
        jsonObject.put("home_addr", this.homeAddr);
    }
    this.uri = jsonObject.get("userProfile_uri");
    if (this.uri == null) {
        throw NullPointerException("Profile URI is null", exception);
    }
    return jsonObject;
}
```

## Sharing content to Facebook

```java
Builder shareToFacebook(String shareContent) {
    Builder builder = new Builder();
    builder.setContentTitle("I'm designing my own tees on my phone!");
    builder.setContentUrl(Uri.parse("https://snaptee.co/getapp"));
    builder.setShareContent(shareContent);
    Log.d("FacebookFunctions", "Try to invite FB");
    return builder;
}
```

Image Source: FlowDroid
Semantic Clue Locating

• Semantics Locator
  • Knowledge base: 35 privacy items
    • Google Privacy Policy, Financial Times report, prior research, etc.

• Resources in focus
  • Method names
  • Variable names
  • Constant strings

<table>
<thead>
<tr>
<th>Category</th>
<th>Sample Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Attributes</td>
<td>first name, last name, gender, birth date, nick name, education, app list, device os, credit card, etc.</td>
</tr>
<tr>
<td>User Identifiers</td>
<td>user id, account number, access token, sina id, facebook id, twitter id, etc.</td>
</tr>
<tr>
<td>Location</td>
<td>latitude, longitude, lat, lng, user address, zip code, city, street, etc.</td>
</tr>
<tr>
<td>Account</td>
<td>account name, user name, phone number, mobile no, password, passwd, pwd etc.</td>
</tr>
</tbody>
</table>
Semantic Clue Locating

• Semantics Checker
  • Goal: In-depth semantic analysis for privacy-related tokens

• Typed-dependency parsing

  • Direct-object relation (dobj)
  • Nominal subject (nsubj)
  • Negation modifier (neg)
  • ...

Semantic Clue Locating

• Semantics Checker
  • Goal: In-depth semantic analysis for privacy-related tokens

• Typed-dependency parsing

Method Name: get_location_update_time_interval ()
Semantic Clue Locating

- **Semantics Checker**
  - Goal: In-depth semantic analysis for *privacy-related tokens*

- Typed-dependency parsing

“I’m designing my own tees on my phone!”
Sensitive Data Discovery

Privacy-related semantics ≠ Sensitive data

```java
If (userJson.contains("home_addr")) {
    ....
    ✔ userJson.get("home_addr")
}
Else {
    ✔ userJson.put("home_addr", this.homeAddr)
    Log.d("location_info", "location updated.")
}
```
**Sensitive Data Discovery**

- Structure Analyzer
  - SVM classifier for identifying **data objects**

```
Log.d("Location_info", "location updated.");
```

```
userJson.get("home_addr")
```

```
userJson.containsKey("home_addr")
```

```
userJson.put("home_addr", this.homeAddr)
```

[Diagram showing non-data objects and data objects]
Sensitive Data Discovery

- Structure Analyzer
  - SVM classifier for identifying **data objects**
  - Selected features

Data read or write operations:
- Method name
- Parameter/Return type
- Base value (Class) type
- Constant-variable pattern

```
If (userJson.contains("home_addr")) {
    // ...
    userJson.get("home_addr")
}
else {
    userJson.put("home_addr", homeAddr)
    Log.d("location_info", "location updated.")
}
```

- Method name
- Parameter/Return type
- Base value (Class) type
- Constant-variable pattern
Leakage Tracker

• Integrate with existing framework
  • Sources: parameters or return values in identified statements
  • Data-flow based taint analysis
    • E.g., FlowDroid [PLDI’14], Epicc [Security’13]
Evaluation

- Overall effectiveness
  - Manual validation
    - 100 randomly selected popular apps from Google Play
  - Final precision: 91.5%

- FP/FN analysis
  - Insufficient semantic analysis
  - Cases not covered by our labeled training set

```java
void saveEvent("init", "put access token to extras", $r1);

Integer gender = getUserGender(user);
```
Limitation

• Obfuscation
  
  • Limited to deeply obfuscated code with all its semantic information removed
  
  • Preserved semantic information under moderate obfuscation
    • System-level methods (APIs)
    • Reflections
    • Interfaces of third-party SDKs
  
  • 11.3% (426/3,775) of the statements were obfuscated in our testing dataset.
Measurement Highlight

- Seek for data exposure risk to third-party libraries

- 118,296 apps (26.5%) leak private user data
  - Exclude system controlled sources (e.g., IMEI, ICCID)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Collect Time</th>
<th>Total Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play-2015</td>
<td>Nov.15 - Dec.15</td>
<td>13,500</td>
</tr>
<tr>
<td>Play-2016</td>
<td>Jul.16 - Aug.16</td>
<td>71,686</td>
</tr>
<tr>
<td>Tencent-2015</td>
<td>Feb.15 - Apr.15</td>
<td>169,051</td>
</tr>
<tr>
<td>Tencent-2016</td>
<td>Jun.16 - Jul.16</td>
<td>191,431</td>
</tr>
<tr>
<td>Total</td>
<td>Nov.15 - Aug.16</td>
<td>445,668</td>
</tr>
</tbody>
</table>
Leakage Patterns

• Third-party libraries
  • Deliberately harvest information from apps
    • E.g., Social network Sharing SDK (over 30% + share market in China)

• Apps developers
  • Give private information in apps to third-party libraries through API interfaces
    • Intended information disclosure and over-sharing
Intended Disclosure

• Popular Dating App

• Each time user seeks for a nearby potential dating target, the app sends user’s precise location, bio information, dating targets, name on Instagram, etc. to Appboy.

"user":{"Can Create Group":true,"Seeking Distance":50,"Account Creation Date":"2016-11-17T16:56:32.163Z","Profile Enabled Groups":false,"gender":"f","Seeking Gender":1,"Group Status":0,"Has Work Info":true,"Has Education Info":true,"Instagram":"Susan_***","Has e0c000e0a2b9", "start_time": 1.479401816693E9, "events": [{"n":"lr","d":{"ll_accuracy":19.80900001525879,“altitude”:0,"longitude":-86.47*,"latitude":39.16*}}}
Conclusion

- **ClueFinder**
  - A novel technique for identifying sensitive sources
  - Extend scope for labelling more sensitive data from app code

- **Large scale measurement**
  - Privacy exposure risk to third-party libraries
  - Highlight the importance of data protection in today’s software composition
Thanks!

Q&A

• nanyuhong@fudan.edu.cn
Over-sharing

• SnapTee
  • Customize Tee design and shopping.
  • Installs 1,000,000 - 5,000,000

• MixPanel
  • “understand who your users are, see what they do before or after they sign up”

```json
{"$set":
  
  
  
  
  
  
  
  
  
  
  
  "$set":
  "$username":"p***t",
  "$email":"li**v@gmail.com",
  "$first_name":"John",
  "$last_name":"Smith",
  "Twitter":"795**16"},
  "$token":"f81d***cdf96",
  "$time":"1479324910201","...
}
Evaluation

• Implementation
  • Java (1,604 LoCs) and Python (609 LoCs)
  • Extends FlowDroid framework
  • Stanford Parser for NLP analysis (in Java)
  • SVM from scikit-learn (in Python)

• Experimental Settings
  • 32-core server
  • Linux 2.6.32 kernel
  • 64GB memory
Evaluation

• **Classifier for Structure Analyzer**
  • **Training**
    • Randomly selected Statements from 100 popular apps
      • Processed by *Semantic Clue Locating* first
    • 4,326 manually labelled statements
      • Half positive and Half Negative

• **Effectiveness**
  • 92.7% precision and 97.2% recall
    • Based on ten-fold cross validation (over labelled dataset)
Landscape

- 118,296 apps (26.5%) leaking private user data to 3,502 third-party libraries.
  - Exclude system controlled sources (e.g., IMEI, ICCID)
- Play-15 (most popular apps on GP) dataset, was found to have 39.9% of its apps leaking out user data
  - Half of the flagged method invocations (53.1%) are related to HTTP connections

<table>
<thead>
<tr>
<th>DataSet</th>
<th>Affected Apps</th>
<th>Affected Libs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% Apps</td>
<td>Avg.Items/App</td>
</tr>
<tr>
<td>Play-2015</td>
<td>39.9%</td>
<td>7.6</td>
</tr>
<tr>
<td>Play-2016</td>
<td>22.8%</td>
<td>5.26</td>
</tr>
<tr>
<td>Tencent-2015</td>
<td>26.3%</td>
<td>7.55</td>
</tr>
<tr>
<td>Tencent-2016</td>
<td>27.3%</td>
<td>9.53</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>26.5%</strong></td>
<td><strong>8.07</strong></td>
</tr>
</tbody>
</table>
Landscape

• Averagely, each app exposes 8.07 data items (e.g., an identifier name, location, etc.) to 1.97 libraries.

• Individual apps on the un-official market (Tencent) tend to integrate more third-party libraries (1.32 vs. 2.1).

<table>
<thead>
<tr>
<th>DataSet</th>
<th>% Apps</th>
<th>Avg. Items/App</th>
<th>Avg. Libs/App</th>
<th># Libs</th>
<th>Avg. Items/Lib</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play-2015</td>
<td>39.9%</td>
<td>7.6</td>
<td>2.83</td>
<td>709</td>
<td>2.45</td>
</tr>
<tr>
<td>Play-2016</td>
<td>22.8%</td>
<td>5.26</td>
<td>1.32</td>
<td>1,011</td>
<td>2.36</td>
</tr>
<tr>
<td>Tencent-2015</td>
<td>26.3%</td>
<td>7.55</td>
<td>1.64</td>
<td>2,315</td>
<td>2.43</td>
</tr>
<tr>
<td>Tencent-2016</td>
<td>27.3%</td>
<td>9.53</td>
<td>2.1</td>
<td>3,097</td>
<td>2.33</td>
</tr>
<tr>
<td>Total</td>
<td>26.5%</td>
<td>8.07</td>
<td>1.97</td>
<td>3,502</td>
<td>2.39</td>
</tr>
</tbody>
</table>