Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges And Solutions for Analyzing Android

Alexandre Bartel

University of Luxembourg

September 8, 2014

Supervisor: Yves Le Traon
Advisors: Jacques Klein & Martin Monperrus
Static Analysis of a Permission-Based Security System

The application declares permissions $p_1$ and $p_2$. The permissions are used in different contexts:

- $p_1$: Context 1
- $p_2$: Context 2

The diagram shows the relationship between the application and the different contexts.
Static Analysis of a Permission-Based Security System

The application declares permissions $p_1$ and $p_2$
Static Analysis of a Permission-Based Security System

The application declares permissions $p_1$ and $p_2$
The application declares permissions $p_1$ and $p_2$. 

The diagram illustrates the relationship between the application and the framework, showing how permissions are distributed and managed.
The application declares permissions $p_1$ and $p_2$.
Static Analysis of a Permission-Based Security System

The application declares permissions $p_1$ and $p_2$.
The application declares permissions \( p_1 \) and \( p_2 \).
The application declares permissions $p_1$ and $p_2$.
The application declares permissions $p_1$ and $p_2$

Framework

\[
\begin{pmatrix}
1 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
\end{pmatrix}
\]

Application

\[
\begin{pmatrix}
e_1 & e_2 & e_3 & e_4 \\
1 & 1 & 1 & 0 \end{pmatrix}
\]
Methodology to Compute Permission Set (Step 1/3)

Step 1: Extract Framework Permission Matrix

\[
M = \begin{pmatrix}
    e_1 & 1 & 0 & 0 \\
    e_2 & 1 & 0 & 0 \\
    e_3 & 0 & 0 & 0 \\
    e_4 & 0 & 1 & 0 \\
\end{pmatrix}
\]

This step is only done *once* (for a given framework).
Methodology to Compute Permission Set (Step 2/3)

Step 2: Extract Application Access Vector

\[ AV_{app} = \begin{pmatrix} e_1 & e_2 & e_3 & e_4 \\ 1 & 1 & 1 & 0 \end{pmatrix} \]

This step is done for every application.
Methodology to Compute Permission Set (Step 3/3)

Step 3: Infer Permission Set of the Application

\[ IP_{app} = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \]

\[ IP_{app} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \]

This step is done for every application.
Android Framework Call Graph Construction
Android Framework Call Graph Construction

Framework

\[
\begin{align*}
e_1 & \rightarrow f_1 & e_2 & \rightarrow f_2 & e_3 & \rightarrow f_3 & e_4 & \rightarrow f_8 \\
& \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \\
f_4 & \rightarrow f_5 & f_3 & \rightarrow f_8 & & f_9 & \rightarrow ck_2 & \\
& & & & \quad \downarrow & \quad \downarrow & \quad \downarrow & \\
f_6 & \rightarrow ck_1 & & & & & & \\
& & & & & & & \\
p_1 & & & & & & & \\
p_2 & & & & & & & \\
p_3 & & & & & & &
\end{align*}
\]
Android Framework Call Graph Construction

Framework

- e1
  - f1
    - f4
      - f6
        - ck1
  - f2
    - f5
  - f3
  - f8
    - f9
      - ck2

- p1
- p2
- p3
Android Framework Call Graph Construction

Framework

\[ f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_8 \rightarrow ck_2 \rightarrow p_2 \]

\[ f_4 \rightarrow f_5 \rightarrow f_9 \rightarrow ck_1 \rightarrow p_1 \]

\[ e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \]

Alexandre Bartel

Static Analysis of Permission-Based Systems
Android Framework Call Graph Construction

Framework

$\text{ck}_1 \rightarrow f_6 \rightarrow f_4 \rightarrow f_1 \rightarrow e_1$

$\text{ck}_2 \rightarrow f_9 \rightarrow f_8 \rightarrow e_4$

$p_1 \rightarrow f_2 \rightarrow e_2$

$p_2 \rightarrow f_3 \rightarrow e_3$

$p_3$
Call Graph Construction Techniques for Java

- Not precise: CHA (based on class hierarchy)
  - CHA essential (1/4)
  - CHA intelligent (2/4)

- Field sensitive: Spark
  - Spark naive (3/4)
  - Spark intelligent (4/4)
Uses CHA algorithm for call graph

Locates check methods in the call graph

Extracts names of checked permissions

Permission Set

<table>
<thead>
<tr>
<th># entry points</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>31,791</td>
<td>0%</td>
</tr>
<tr>
<td>1</td>
<td>&lt;0.01%</td>
</tr>
<tr>
<td>18,237</td>
<td>36%</td>
</tr>
<tr>
<td>50,029</td>
<td>100%</td>
</tr>
</tbody>
</table>

Why explosion of permission set size?

Call graph goes through binder code
CHA Essential (1/4)

- Uses CHA algorithm for call graph
CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
- Extracts names of checked permissions
CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
- Extracts names of checked permissions

<table>
<thead>
<tr>
<th>Permission Set</th>
<th># entry points</th>
</tr>
</thead>
<tbody>
<tr>
<td>with 0 permissions</td>
<td>31,791 (64%)</td>
</tr>
<tr>
<td>with 1 permissions</td>
<td>1 (&lt; 0.01%)</td>
</tr>
<tr>
<td>with 105 permissions</td>
<td>18,237 (36%)</td>
</tr>
<tr>
<td></td>
<td>50,029 (100%)</td>
</tr>
</tbody>
</table>
CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
- Extracts names of checked permissions

<table>
<thead>
<tr>
<th>Permission Set</th>
<th># entry points</th>
</tr>
</thead>
<tbody>
<tr>
<td>with 0 permissions</td>
<td>31,791 (64%)</td>
</tr>
<tr>
<td>with 1 permissions</td>
<td>1 (&lt; 0.01%)</td>
</tr>
<tr>
<td>with 105 permissions</td>
<td>18,237 (36%)</td>
</tr>
<tr>
<td></td>
<td>50,029 (100%)</td>
</tr>
</tbody>
</table>

- Why explosion of permission set size?
CHA Essential (1/4)

- Uses CHA algorithm for call graph
- Locates check methods in the call graph
- Extracts names of checked permissions

<table>
<thead>
<tr>
<th>Permission Set</th>
<th># entry points</th>
</tr>
</thead>
<tbody>
<tr>
<td>with 0 permissions</td>
<td>31,791 (64%)</td>
</tr>
<tr>
<td>with 1 permissions</td>
<td>1 (&lt; 0.01%)</td>
</tr>
<tr>
<td>with 105 permissions</td>
<td>18,237 (36%)</td>
</tr>
<tr>
<td></td>
<td>50,029 (100%)</td>
</tr>
</tbody>
</table>

- Why explosion of permission set size?
  - Call graph goes through binder code
CHA Essential (1/4): The Real World System with Multiple Software Layers

(source: Gargenta, 2012)
CHA Essential (1/4): The Reason of the Explosion

API
methods
Binder
transact
method
Services
onTransact
methods
Services
target
methods
CHA Intelligent (2/4)

- Uses CHA algorithm for call graph
- Finds check methods in the call graph
- Extracts names of checked permissions
- Handles system service communication through the "Binder"
Uses CHA algorithm for call graph
Uses CHA algorithm for call graph

Finds check methods in the call graph
CHA Intelligent (2/4)

- Uses CHA algorithm for call graph
- Finds check methods in the call graph
- Extracts names of checked permissions
CHA Intelligent (2/4)

- Uses CHA algorithm for call graph
- Finds check methods in the call graph
- Extracts names of checked permissions
- Handles system service communication through the "Binder"
CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

```java
r = getSystemService()
p = r.getPassword()
```

Account System Service

```java
getPassword() {
    checkPermission();
    return password;
}
```

Binder (Linux module)
CHA Intelligent (2/4): Handling Binder

Application Code

![Diagram of application code and account system service](image)

Binder (Linux module)

```java
r = getSystemService()
p = r.getPassword()
```

```java
getPassword() {
    checkPermission();
    return password;
}
```
CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

1. $r = \text{getSystemService}()$
2. $p = r.get\text{Password}()$

Account System Service

getPassword() {
  checkPermission();
  return password;
}

Binder (Linux module)
CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

r = getSystemService()
p = r.getPassword()

Account System Service

getPassword() {
checkPermission();
return password;
}

Binder (Linux module)
CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

```
r = getSystemService()
p = r.getPassword()
```

Account System Service

```
getPassword() {
    checkPermission();
    return password;
}
```

Binder (Linux module)
CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

getPassword()

checkPermission();

return password;

Account System Service

g = r.getPassword();

Binder (Linux module)
Android Framework

CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

getPassword()

checkPermission();

return password;

Account System Service

r = getSystemService()
p = r.getPassword()

getPassword() {
    checkPermission();
    return password;
}

Binder (Linux module)
CHA Intelligent (2/4): Handling Binder

Application Code

Service Call

r = getSystemService()
p = r.getPassword()

getPassword() { checkPermission(); return password; }

Account System Service

Binder (Linux module)
### CHA Intelligent Results (2/4)

<table>
<thead>
<tr>
<th>Permission Set</th>
<th># entry points (CHA Intelligent)</th>
<th># entry points (CHA Essential)</th>
</tr>
</thead>
<tbody>
<tr>
<td>with 0 permissions</td>
<td>32,924 (65.8%)</td>
<td>32,924 (64%)</td>
</tr>
<tr>
<td>with 1 permissions</td>
<td>39 (0.08%)</td>
<td>1 (&lt; 0.01%)</td>
</tr>
<tr>
<td>with 2 permissions</td>
<td>55 (0.12%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>with &gt; 65 permissions</td>
<td>17,011 (34.0%)</td>
<td>18,237 (36%)</td>
</tr>
<tr>
<td></td>
<td>50,029 (100%)</td>
<td>50,029 (100%)</td>
</tr>
</tbody>
</table>
Spark Naive (3/4)

- Off-the-shelf
Spark Naive (3/4)

- Off-the-shelf
- Only about 1800 methods are analyzed: why?
Spark Naive (3/4)

- Off-the-shelf
- Only about 1800 methods are analyzed: why?
  - Static methods

This approach completely fails to generate entry point wrappers to initialize objects.
Spark Naive (3/4)

- Off-the-shelf
- Only about 1800 methods are analyzed: why?
  - Static methods
- This approach completely fails
Spark Naive (3/4)

- Off-the-shelf
- Only about 1800 methods are analyzed: why?
  - Static methods
- This approach completely fails

→ generate entry point “wrappers” to initialize objects
Spark Intelligent (4/4)
Spark Intelligent (4/4)

- Generates entry point wrappers
Spark Intelligent (4/4)

- Generates entry point wrappers
- Handles system services initialization and managers initialization
Spark Intelligent (4/4)

API methods
Binder transact method
Services onTransact methods
Services target methods
Spark Intelligent (4/4)
Spark Intelligent (4/4)

API 

Binder method 

Services methods 

Services target methods

Android Framework
Spark Intelligent (4/4)

Android Framework

API methods
Binder transact method
Services onTransact methods
Services target methods

inject service

Alexandre Bartel
Spark Intelligent (4/4)

API methods
Bind onTransact method
Services methods
Services target methods

inject service

Alexandre Bartel
Static Analysis of Permission-Based Systems
## Spark Intelligent Results (4/4)

<table>
<thead>
<tr>
<th>Permission Set</th>
<th># entry points (Spark Intelligent)</th>
<th># entry points (CHA Intelligent)</th>
<th># entry points (CHA Essential)</th>
</tr>
</thead>
<tbody>
<tr>
<td>with 0 permissions</td>
<td>42,895 (98.77%)</td>
<td>32,924 (65.8%)</td>
<td>32,924 (64%)</td>
</tr>
<tr>
<td>with 1 permissions</td>
<td>471 (1.08%)</td>
<td>39 (0.08%)</td>
<td>1 (&lt; 0.01%)</td>
</tr>
<tr>
<td>with 2 permissions</td>
<td>48 (0.11%)</td>
<td>55 (0.12%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>with 3 permissions</td>
<td>10 (0.01%)</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>with &gt; 3 permissions</td>
<td>3 (0.02%)</td>
<td>17,011 (34.0%)</td>
<td>18,237 (36%)</td>
</tr>
<tr>
<td></td>
<td>43,427 (100%)</td>
<td>50,029 (100%)</td>
<td>50,029 (100%)</td>
</tr>
</tbody>
</table>
# Spark Intelligent Results (4/4)

<table>
<thead>
<tr>
<th>Permission Set</th>
<th># entry points (Spark Intelligent)</th>
<th># entry points (CHA Intelligent)</th>
<th># entry points (CHA Essential)</th>
</tr>
</thead>
<tbody>
<tr>
<td>with 0 permissions</td>
<td>42,895 (98.77%)</td>
<td>32,924 (65.8%)</td>
<td>32,924 (64%)</td>
</tr>
<tr>
<td>with 1 permissions</td>
<td>471 (1.08%)</td>
<td>39 (0.08%)</td>
<td>1 (&lt; 0.01%)</td>
</tr>
<tr>
<td>with 2 permissions</td>
<td>48 (0.11%)</td>
<td>55 (0.12%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>with 3 permissions</td>
<td>10 (0.01%)</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>with &gt; 3 permissions</td>
<td>3 (0.02%)</td>
<td>17,011 (34.0%)</td>
<td>18,237 (36%)</td>
</tr>
<tr>
<td></td>
<td>43,427 (100%)</td>
<td>50,029 (100%)</td>
<td>50,029 (100%)</td>
</tr>
</tbody>
</table>

classes are removed to speed up the experiment
Evaluation (1/3): Android 4

Comparison Spark Intelligent vs. PScout [1]

Evaluation (1/3): Android 4

Comparison Spark Intelligent vs. PScout [1]

<table>
<thead>
<tr>
<th>Permission set</th>
<th>Number of Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>#API Methods in Spark and PScout</td>
<td>468 (100%)</td>
</tr>
<tr>
<td>Identical</td>
<td>289 (61.75%)</td>
</tr>
<tr>
<td>we find less permission checks</td>
<td>176 (37.60%)</td>
</tr>
<tr>
<td>we find more permission checks</td>
<td>3 (0.64%)</td>
</tr>
</tbody>
</table>

### Evaluation (1/3): Android 4

#### Comparison Spark Intelligent vs. PScout [1]

<table>
<thead>
<tr>
<th>Permission set</th>
<th>Number of Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>#API Methods in Spark and PScout</td>
<td>468 (100%)</td>
</tr>
<tr>
<td>Identical</td>
<td>289 (61.75%)</td>
</tr>
<tr>
<td>we find less permission checks</td>
<td>176 (37.60%)</td>
</tr>
<tr>
<td>we find more permission checks</td>
<td>3 (0.64%)</td>
</tr>
</tbody>
</table>

- We are more precise (ex: 1 permission against 5 for entry point `exitKeyguardSecurely(...)`)  

---

Evaluation (1/3): Android 4

Comparison Spark Intelligent vs. PScout [1]

<table>
<thead>
<tr>
<th>Permission set</th>
<th>Number of Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>#API Methods in Spark and PScout</td>
<td>468 (100%)</td>
</tr>
<tr>
<td>Identical</td>
<td>289 (61.75%)</td>
</tr>
<tr>
<td>we find less permission checks</td>
<td>176 (37.60%)</td>
</tr>
<tr>
<td>we find more permission checks</td>
<td>3 (0.64%)</td>
</tr>
</tbody>
</table>

- We are more precise (ex: 1 permission against 5 for entry point exitKeyguardSecurely(...))
- We are less precise: we not analyze some modules (ex: non-Java code)

Evaluation (2/3): Android 2.2

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

→ Stowaway = Testing Approach

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]

→ Stowaway = Testing Approach

Results

▶ 552 / 673 entry points are “correct”

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]
→ Stowaway = Testing Approach

Results

- 552 / 673 entry points are “correct”
- 119 / 673 have more permissions

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]
→ Stowaway = Testing Approach

Results

- 552 / 673 entry points are “correct”
- 119 / 673 have more permissions
- At least 3 entry points in Stowaway were missing permissions

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]
→ Stowaway = Testing Approach

Results

- 552 / 673 entry points are “correct”
- 119 / 673 have more permissions
- At least 3 entry points in Stowaway were missing permissions

Testing (1) yields an under-approximation.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]
→ Stowaway = Testing Approach

Results

▶ 552 / 673 entry points are “correct”
▶ 119 / 673 have more permissions
▶ At least 3 entry points in Stowaway were missing permissions

Testing (1) yields an under-approximation.
Static (2) Analysis yields an over-approximation.

Evaluation (2/3): Android 2.2

Comparison Spark Intelligent vs. Stowaway [1]
→ Stowaway = Testing Approach

Results

- 552 / 673 entry points are “correct”
- 119 / 673 have more permissions
- At least 3 entry points in Stowaway were missing permissions

Testing (1) yields an under-approximation.
Static (2) Analysis yields an over-approximation.
Combining the (1) and (2) to have “correct” results?

Evaluation (3/3): Permission Gaps in Real World Applications
Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications:
Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications: 96 (13%) have a permission gap
Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications: 96 (13%) have a permission gap
- 679 Android Market applications:
Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications: 96 (13%) have a permission gap
- 679 Android Market applications: 35 (5%) have a permission gap
Evaluation (3/3): Permission Gaps in Real World Applications

- 742 Freewarelovers applications: 96 (13%) have a permission gap
- 679 Android Market applications: 35 (5%) have a permission gap

![Diagram showing the number of applications with 1, 2, or 3+ permissions with percentage of each category for Freewarelovers and Google Play.]
Contributions Summary

- Empirically demonstrated that off-the-shelf static analysis cannot address the extraction of permissions in Android
- Static analysis of Android requires inner knowledge of the stack
- Static analysis components must be put together:
  1. Entry point initialization
  2. String analysis
  3. Service initialization
  4. Service redirection
Contributions Summary