Automatic Mediation Of Privacy-sensitive Resource Access In Smartphone Applications

Ben Livshits and Jaeyeon Jung

Microsoft Research
PERMISSIONS IN MOBILE APPS
Permissions Flavors

- installation-time permissions
- runtime permissions

Allow access to GPS location?

[Images of checkboxes, OK, Cancel buttons]
Examples of Permissions in Different Mobile Operating Systems

Installation-time permissions:
- **Your location**
- **Your personal information**
- **Network communication**
- **Your accounts**
- **Storage**
- **System tools**

Runtime permissions:
- **Allow IMDb to access and use your location?**
  - Sharing this information allows us to find theaters and showtimes near you. We won't share this information.

Screen with permission options:
- Allow
- Cancel
Obtain user permission before accessing location data. Precise geo-location information is increasingly considered sensitive information. You should only collect and transmit such information when you have your users’ clear, opt-in permission.

While most platforms do require express permission for an app to access location information, if you are using that data in unexpected ways or are transmitting that information to third-parties, make sure you get your own permission from the user before doing so.20

In your app’s privacy policy, specify how you collect, use and share location data. You should also provide disclosure for: (1) the level of location data collection such as precise or fine, zip level, zip+4, or coarse; (2) whether the data is being used with a unique mobile identifier; and (3) the period of time that the user’s location data is linked with the user’s identifier.
Guarding Location Access

• Focus on 3 representative applications in the Windows Phone store

<table>
<thead>
<tr>
<th>App</th>
<th>Resource</th>
<th>Access</th>
<th>APIs used</th>
<th>DLLs using location</th>
</tr>
</thead>
<tbody>
<tr>
<td>AroundMe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Burger King</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LumiaClock</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public static bool AroundMe.App.CheckOptin() {
    if (((Option)Enum.Parse(typeof(Option), Config.GetSetting(SettingConstants.UseMyLocation), true)) == Option.Yes)
        return GetCurrentCoordinates();

    if (MessageBox.Show("This app needs ...", "Use location data?", MessageBoxButton.OKCancel) == MessageBoxButton.OK)
        Config.UpdateSetting(new KeyValuePair<string, string>(SettingConstants.UseMyLocation, Option.Yes.ToString));

    return GetCurrentCoordinates();
}

...
public BurgerKing.View.MapPage() {
    this.InitializeComponent();
    base.DataContext = new MapViewModel();
    this.BuildApplicationBar();
    if (AppSettings.Current.UseLocationService) {
        this.watcher = new GeoCoordinateWatcher();
    }
    ..
}

protected virtual void GART.Controls.ARDisplay.
    OnLocationEnabledChanged(
        DependencyPropertyChangedEventArgs e) {
    if (this.servicesRunning) {
        if (this.LocationEnabled) {
            this.StartLocation();
        ...
}
public SomaAd() {
    ...
    this._locationUseOK = true;
    ...
    if (this._locationUseOK) {
        this.watcher = new GeoCoordinateWatcher
            (GeoPositionAccuracy.Default);
        this.watcher.MovementThreshold = 20.0;
        this.watcher.StatusChanged +=
            new EventHandler
                <GeoPositionStatusChangedEventArgs>
                (this.watcher_StatusChanged);
        this.watcher.Start();
    }
}
Where Does that Leave Us?

- Properly protecting location access is challenging
- Location access is common
  - Some location-related code is in the app
  - A lot of location access in third-party libraries
- Location choices are sometimes ignored
- Third-party libraries such as ad libraries sometimes expose flags for enabling location access but those are frequently ignored by developers
Contributions

- Study how existing applications implement resource access prompts on a set of Windows Phone applications
Static analysis

- Formulate a problem of valid prompt placement in graph-theoretic terms
- Propose a static analysis algorithm for correct resource access prompt placement
Evaluation

• We evaluate our approach to both locating missing prompts and placing them when they are missing on 100 apps.

• Overall, our two-prong strategy of dominator-based and backward placement succeeds in about 95% of all unique cases.

• Our analyses run in seconds, making it possible to run them as part of the app submission process.
ANALYSIS APPROACH
In This Paper...

• We focus on a completely automatic way to **insert missing prompts**

• Our approach is **static**: we want to be able to check for missing prompts and insert compensating code even if we cannot hit it at through runtime testing

• **Graph-theoretic approach**
  • Represent the application statically as a graph
  • An inter-procedural version of control flow graph (CFG)
  • Reason about prompt placement in graph-theoretic terms

• **Not information flow**
  • A lot of work on finding undesirable information flows
  • We reason about **control flow** not **data flow**
Challenges

1. Avoiding double-prompts
2. Sticky prompts
3. Avoiding weaker prompts
4. Minimizing prompting
5. Avoiding prompts in background tasks
6. Avoiding prompts in libraries

```java
if(P) l1 = getLocation();
l2 = getLocation();

if(P){
    prompt();
    l1 = getLocation();
l2 = getLocation();
} else{
    prompt();
    l2 = getLocation();
}
```

16
Challenges

1. Avoiding double-prompts
2. Sticky prompts
3. Avoiding weaker prompts
4. Minimizing prompting
5. Avoiding prompts in background tasks
6. Avoiding prompts in libraries

```csharp
if (MessageBox.Show(
    "This app needs to know your location in order to find locations around you, can it use your location data? note: you can change the settings later through the settings menu",
    "Use location data? ", 1) == 1)
{
    Config.UpdateSetting(
        new KeyValuePair<string, string>(
            SettingConstants.UseMyLocation,
            Option.Yes.ToString()));
    return GetCurrentCoordinates();
}
```
Challenges

1. Avoiding double-prompts
2. Sticky prompts
3. Avoiding weaker prompts
4. Minimizing prompting
5. Avoiding prompts in background tasks
6. Avoiding prompts in libraries
Valid Placement

Definition  We say that placement $P \subseteq N$ is a valid placement for a prompt placement problem $\mathcal{P} = \langle N, A, B, E, C, L \rangle$ if the following conditions hold for every runtime execution of the app:

- **Safe:** Every access to resource $r \in R$ is preceded by a prompt check for $r$.
- **Visible:** No prompt is located within a background task or a library.
- **Frugal:** Prompt for $r \in R$ is never invoked unless it is either followed by a call to $get(r)$ or an exception occurs.
- **Not-repetitive:** Prompt for permission $r_2 \in R$ is never invoked if permissions for $r_1 \in R$ have already been granted and $r_2 \subseteq r_1$ (that is, $r_1$ is at least as or more permissive than $r_2$).
Intuition for Placement

1. Start with a resource access
2. “Move” the prompts up until we are outside of background tasks

• Downside:
  • possible to move these prompts too far (to the beginning of the app in the most extreme case)
  • This would violate the frugal requirement.
  • This gives rise to a notion of a prompt being needed at a particular point, for which we use the term anticipating

getLocation()
Dominator-Based Approach

```javascript
getLocation() // Not frugal!
```
Backward Placement

```plaintext
getLocation()
```
Analysis Steps

1. For every resource access type and every node \( n \), pre-compute r-anticipated value \( A_r(n) \)

2. Merge values by meeting them in the semi-lattice of resource types

\[
A(n) = \Lambda \ A_r(n)
\]

3. For every

![Diagram](image)
EVALUATION
## Input Statistics

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
<th>Percentage or Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>apps analyzed</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>app size</td>
<td>7.3MB</td>
<td></td>
</tr>
<tr>
<td>processed methods</td>
<td>352,816</td>
<td>3.5K on average</td>
</tr>
<tr>
<td>background/library methods</td>
<td>26,033</td>
<td>7%</td>
</tr>
<tr>
<td>library methods</td>
<td>25,898</td>
<td>7%</td>
</tr>
<tr>
<td>nodes</td>
<td>1,333,056</td>
<td></td>
</tr>
<tr>
<td>anticipating</td>
<td>171,253</td>
<td>12%</td>
</tr>
<tr>
<td>accesses</td>
<td>227</td>
<td>2 per app</td>
</tr>
<tr>
<td>accesses in background/library methods</td>
<td>78</td>
<td>1/3rd</td>
</tr>
</tbody>
</table>
Benchmarks

• Took 100 WP 7 apps

• To make this meaningful, chose apps with LOCATION and NETWORKING caps

• An average app is 7.3 MB of code

• Uses third-party ad libraries

<table>
<thead>
<tr>
<th>Component</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOMA WP7</td>
<td>42</td>
</tr>
<tr>
<td>NetDragon.PandaReader</td>
<td>13</td>
</tr>
<tr>
<td>EchoEchoBackgroundAgent</td>
<td>10</td>
</tr>
<tr>
<td>Utilities</td>
<td>10</td>
</tr>
<tr>
<td>BMSApp</td>
<td>10</td>
</tr>
<tr>
<td>MobFox.Ads.LocationAware</td>
<td>8</td>
</tr>
<tr>
<td>XIMAD_Ad_Client</td>
<td>7</td>
</tr>
<tr>
<td>EchoEcho</td>
<td>5</td>
</tr>
<tr>
<td>DirectRemote</td>
<td>5</td>
</tr>
<tr>
<td>DCMetroApp</td>
<td>5</td>
</tr>
</tbody>
</table>
Prompt Placement Success

Total
- Succeeded 91%
- Failed 9%

Unique
- Succeeded 95%
- Failed 5%
Dominator-Based vs. Backward

- When dominator-based placement succeeds, it is usually immediate

- Backward placement is helpful for cases where dominator-based placement fails

- However, some of these cases are still too hard, leading to 7 unique failures
Timing

- **app loading**: 1,779
- **call graph construction**: 18,152
- **placement graph construction**: 15,103
- **anticipating computation**: 158
- **finding missing prompts**: 123
- **prompt insertion, per app**: 942
- **dominator-based, per access**: 0
- **backward, per access**: 1,366

Bar chart showing the timing for the mentioned tasks.
Manual Examination

- Picked 10 apps with 27 resource accesses
- Manually exercised as much functionality as possible
- Verification includes running these apps in an emulator to collect network packets and API calls
- **False negatives**: resource access we think is protected whereas in fact at runtime it has no preceding prompts
- Out of 27 accesses our analysis reports 10 as unprotected
- No false negatives observed: analysis correctly identifies them as unprotected and finds proper prompt placements
False Positives

- **False positives**: analysis classifies a resource access as unprotected whereas it is properly protected at runtime.

- 11 out of 21 accesses found as unprotected turn out to be false positives.

- Reasons include:
  - Not recognizing sticky prompts
  - Custom consent dialogs
  - Async calls and XAML

- Our analysis errs on the safe side, introducing false positives and not false negatives.

- False positives may lead to double-prompting:
  - Inserted prompts are sticky, so at most one extra runtime prompt per app
  - Easy to spot and suppress by app store maintainers

- Interesting future research
Conclusions

• Explored the problem of missing prompts that should guard sensitive resource accesses

• Graph-theoretic algorithm for placing prompts

• Approach that balances execution speed and few prompts inserted via dominator-based placement with a comprehensive nature of a more exhaustive backward analysis

• Overall, our two-prong strategy of dominator-based and backward placement succeeds in
  • about 95% of all unique cases
  • highly scalable: analysis usually takes under a second on average

• Suggests that fully-automatic prompt placement is viable