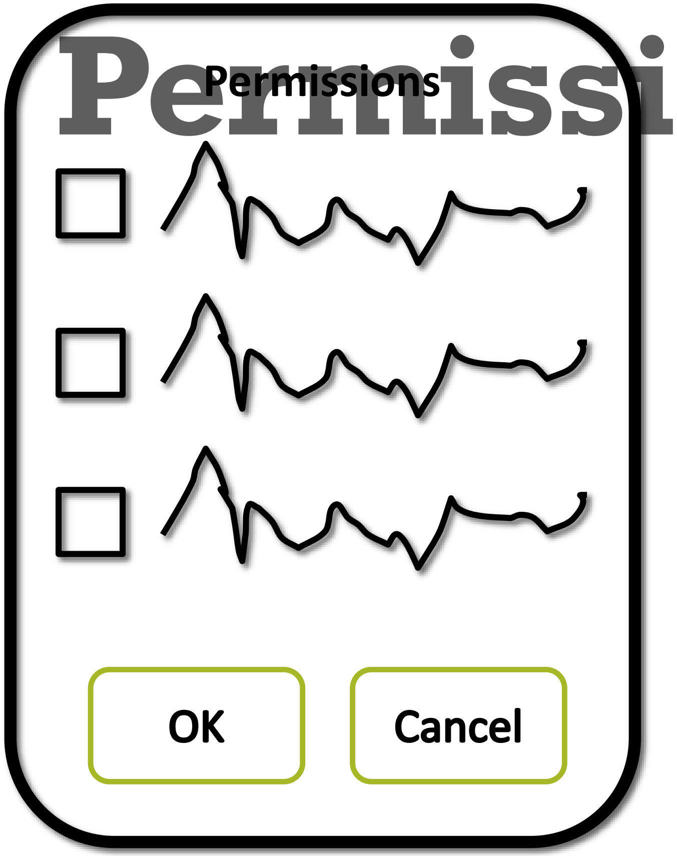
Automatic Mediation Of Privacy-sensitive Resource Access In Smartphone Applications

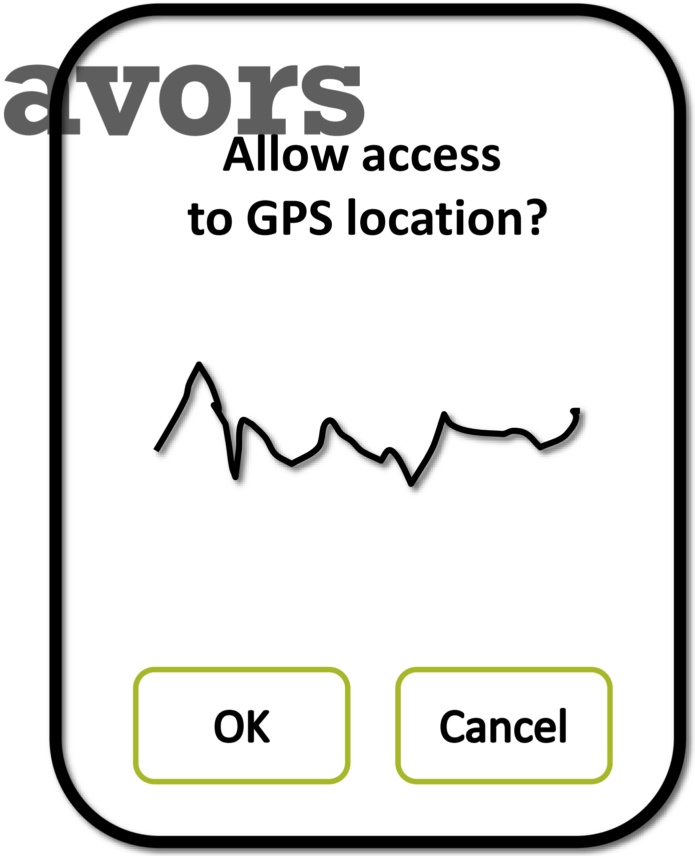
Ben Livshits and Jaeyeon Jung

Microsoft Research

PERMISSIONS IN MOBILE APPS

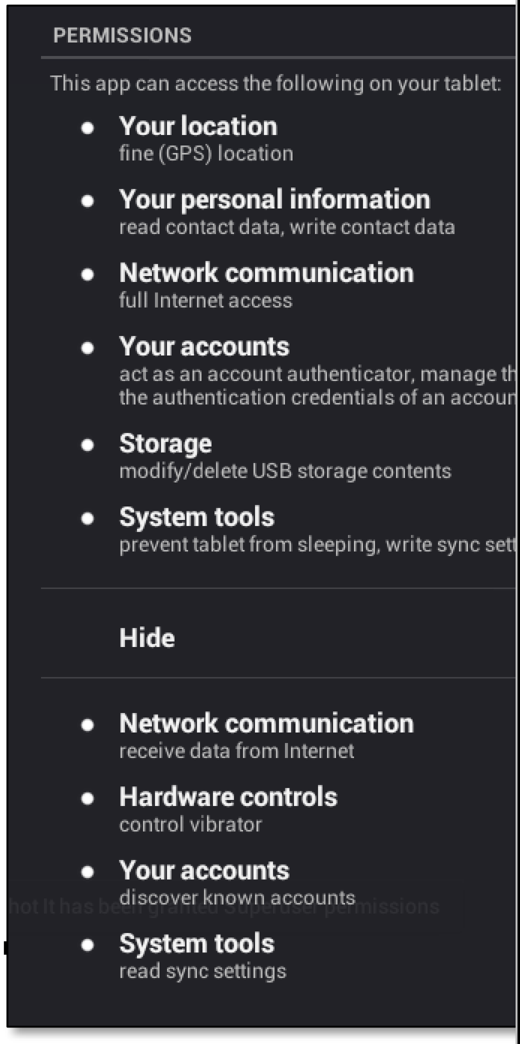
Permissions Flavors





installation-time
permissions

runtime
permissions



PERMISSIONS

This app can access the following on your tablet:

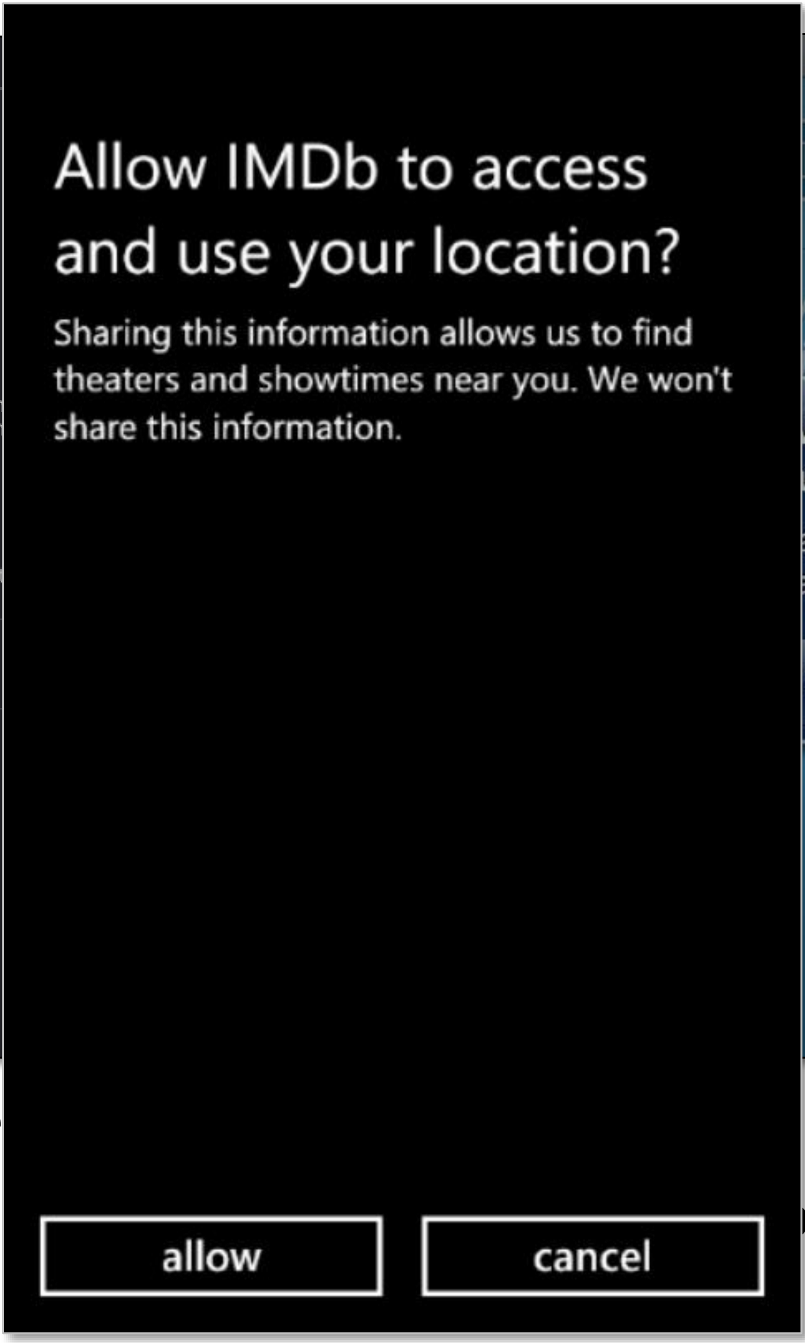
- **Your location**
  fine (GPS) location
- **Your personal information**
  read contact data, write contact data
- **Network communication**
  full Internet access
- **Your accounts**
  act as an account authenticator, manage th
  the authentication credentials of an accoun
- **Storage**
  modify/delete USB storage contents
- **System tools**
  prevent tablet from sleeping, write sync sett

Hide

- **Network communication**
  receive data from Internet
- **Hardware controls**
  control vibrator
- **Your accounts**
  discover known accounts
- **System tools**
  read sync settings

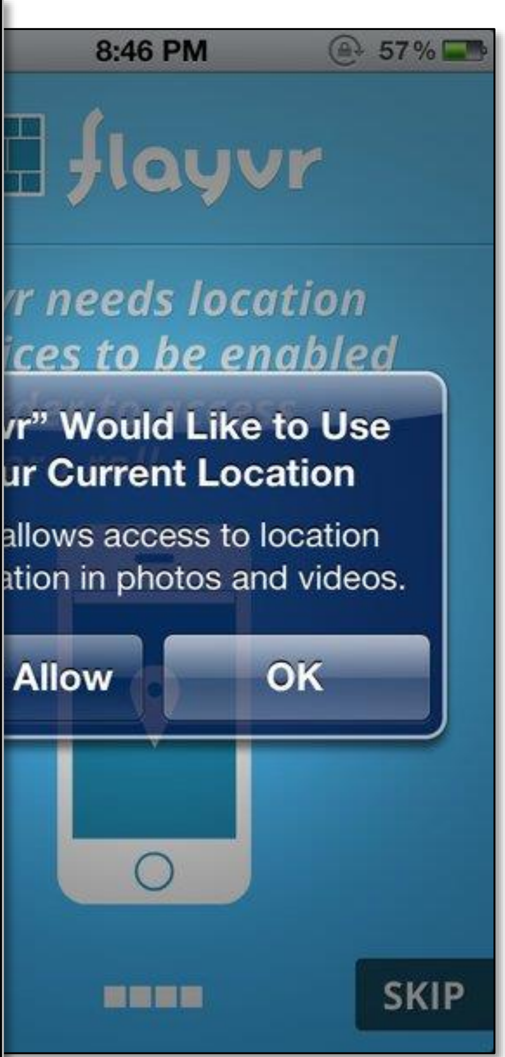
hot It has been granted Superuser permissions

installation-time 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.



r needs location
ices to be enabled

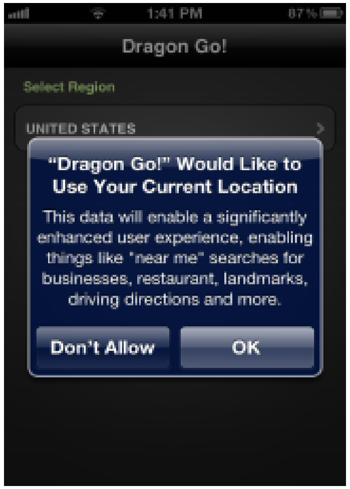
r" Would Like to Use
ur Current Location

allows access to location
ation in photos and videos.

runtime
ermissions

General Guidelines

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.



"Dragon Go!" Would Like to Use Your Current Location

This data will enable a significantly enhanced user experience, enabling things like "near me" searches for businesses, restaurant, landmarks, driving directions and more.

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.

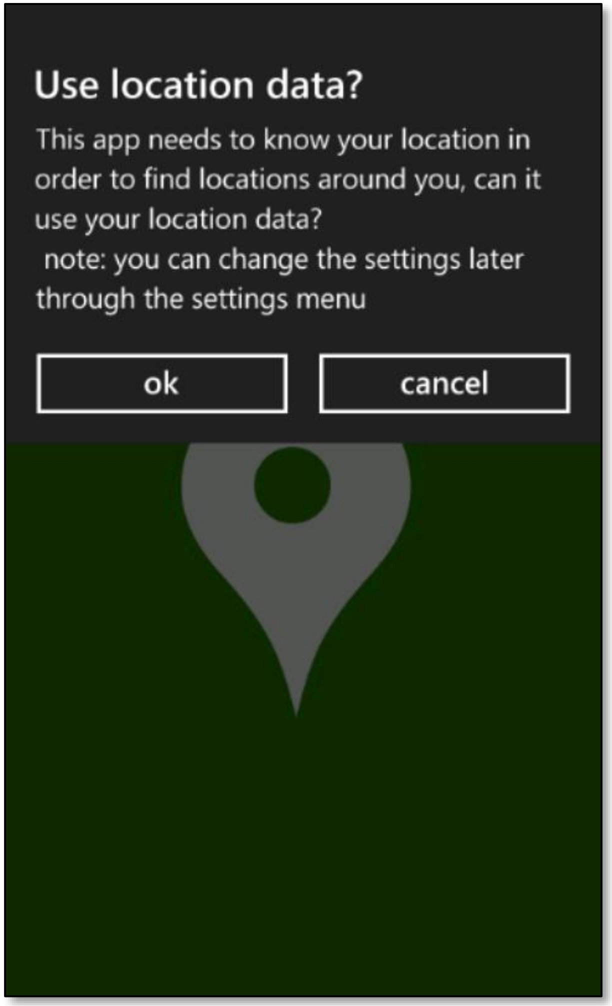
Best Practices for Mobile Application Developers
Center for Democracy & Technology

Guarding Location Access

- Focus on 3 representative applications in the Windows Phone store

App
AroundMe
Burger King
LumiaClock

AroundMe



Use location data?

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

```
public static bool AroundMe.App.CheckOptin() {
 if (((Option)Enum.Parse(typeof(Option),Config.
  (SettingConstants.UseMyLocation),true)) == Op
  return GetCurrentCoordinates();
 }
 if (MessageBox.Show("This app needs ...",
          "Use location data?", MessageBoxButton
        == MessageBoxResult.OK)
 {
  Config.UpdateSetting(new KeyValuePair<string,
  (SettingConstants.UseMyLocation,Option.Yes.To

  return GetCurrentCoordinates();
 }
 ...
}
```

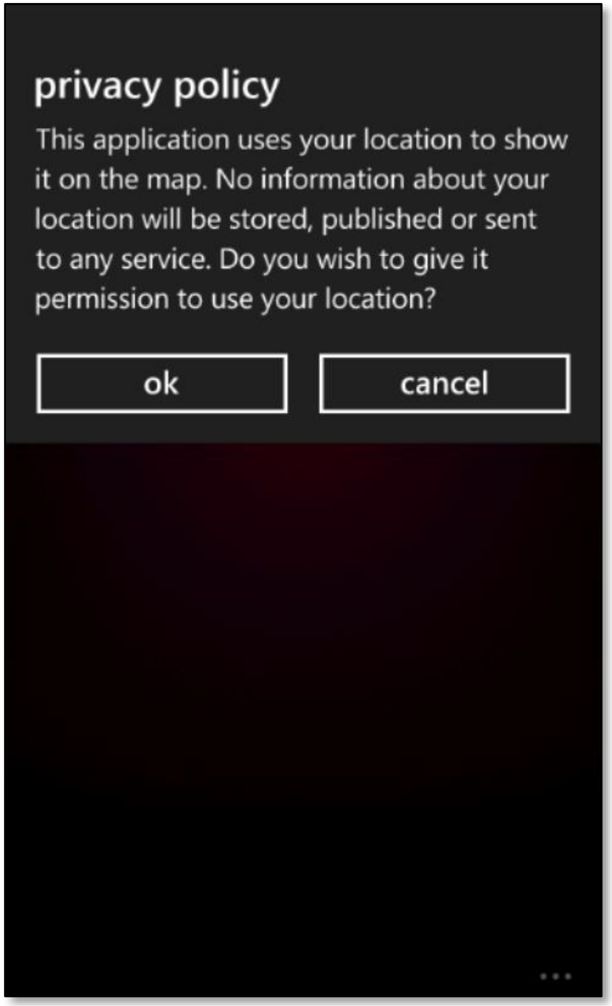
check

prompt

save

access

Burger King



privacy policy

This application uses your location to show it on the map. No information about your location will be stored, published or sent to any service. Do you wish to give it permission to use your location?

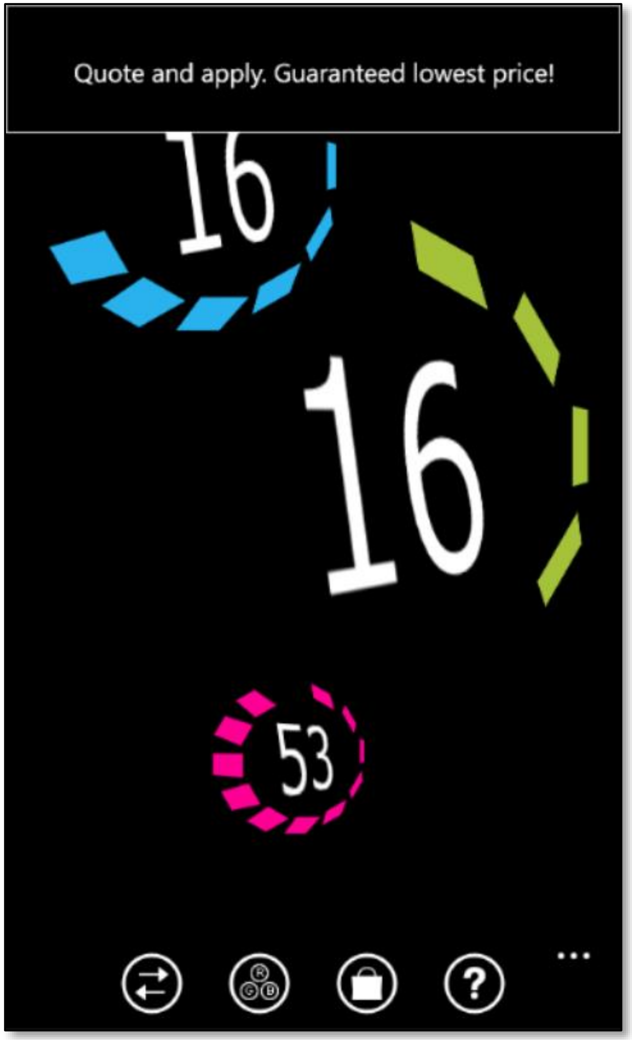
```
public BurgerKing.View.MapPage() {
 this.InitializeComponent();
 base.DataContext = new MapViewModel();
 this.BuildApplicationBar();
 if (AppSettings.Current.UseLocationService){
  this.watcher = new GeoCoordinateWatcher();
 }
..
}

protected virtual void GART.Controls.ARDisplay.
  OnLocationEnabledChanged(
          DependencyPropertyChangedEventArgs e)
{
 if (this.servicesRunning) {
  if (this.LocationEnabled) {
   this.StartLocation();
 …
```

application code

library code

LumiaClock



Quote and apply. Guaranteed lowest price!

```
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();
 }
}
```

library: just do it!

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

1

Contributions

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

2

Static analysis

- Formulate a **problem of valid prompt placement** in graph-theoretic terms

- Propose a **static analysis algorithm** for correct resource access **prompt placement**

3

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

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

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

```
  l2 = getLocation();
}
```

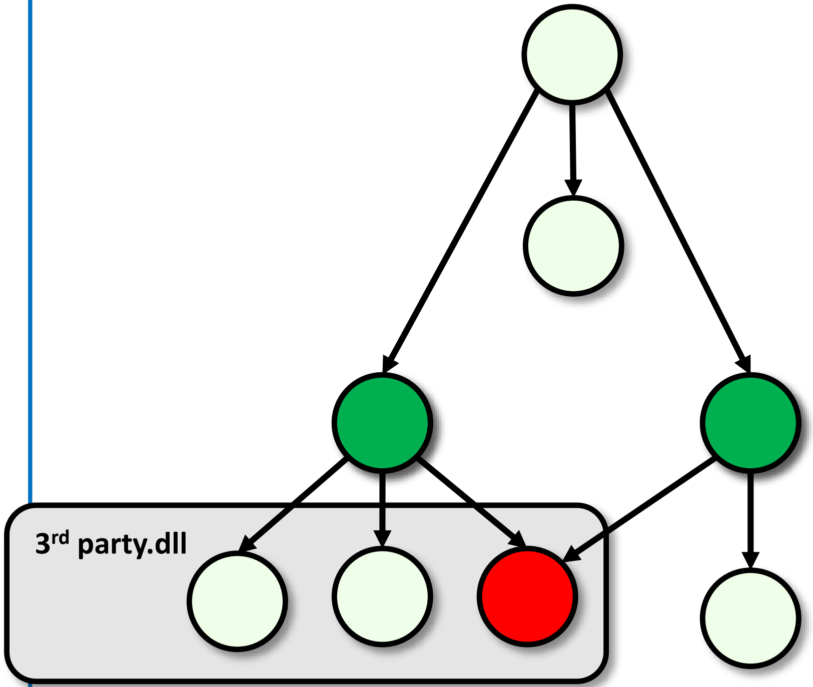
Challenges

1. Avoiding double-prompts
2. Sticky prompts
3. Avoiding
4. Minimizi
5. Avoiding backgrou
6. Avoiding libraries

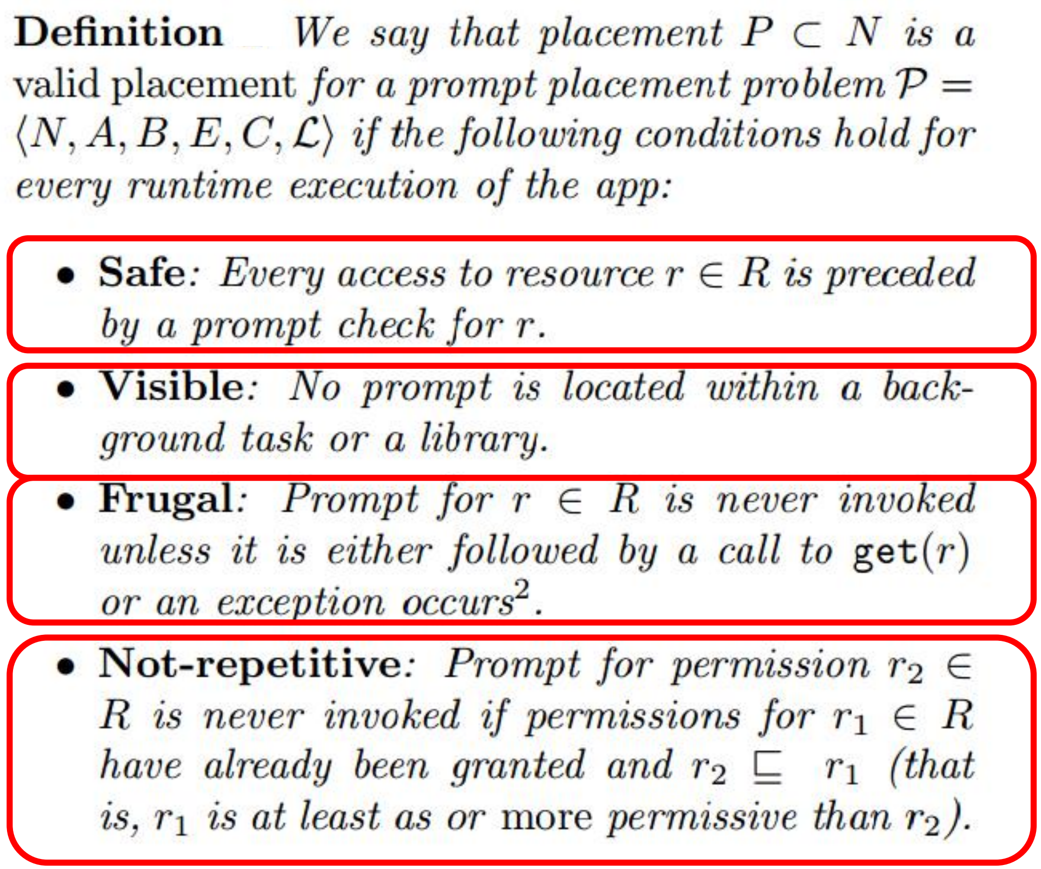
```
if (MessageBox.Show(
   "This app needs to know your location
   in order to find locations
   around you, can it use your location data?
   note: you can change the settings later
   through the settings menu",
   "Use location data? ", 1) == 1)
{
   Config.UpdateSetting(
   new KeyValuePair<string, 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



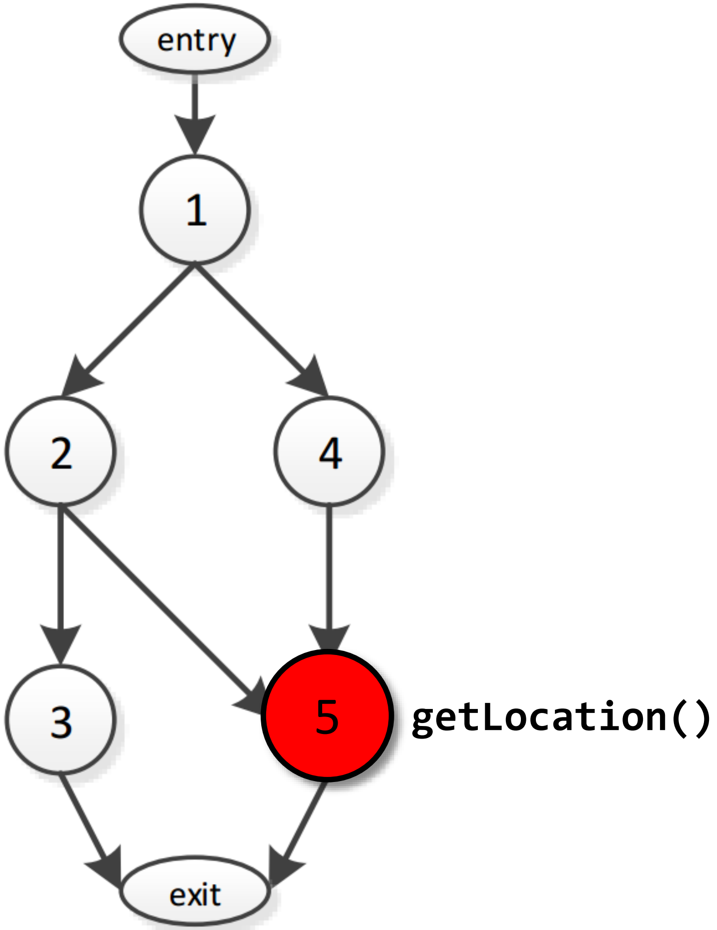
Definition *We say that placement $P \subset N$ is a* valid placement *for a prompt placement problem* $\mathcal{P} = \langle N, A, B, E, C, \mathcal{L} \rangle$ *if the following conditions hold for every runtime execution of the app:*

- **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*[2].
- **Not-repetitive**: *Prompt for permission $r_2 \in R$ is never invoked if permissions for $r_1 \in R$ have already been granted and $r_2 \sqsubseteq r_1$ (that is, r_1 is at least as or* more *permissive than r_2).*

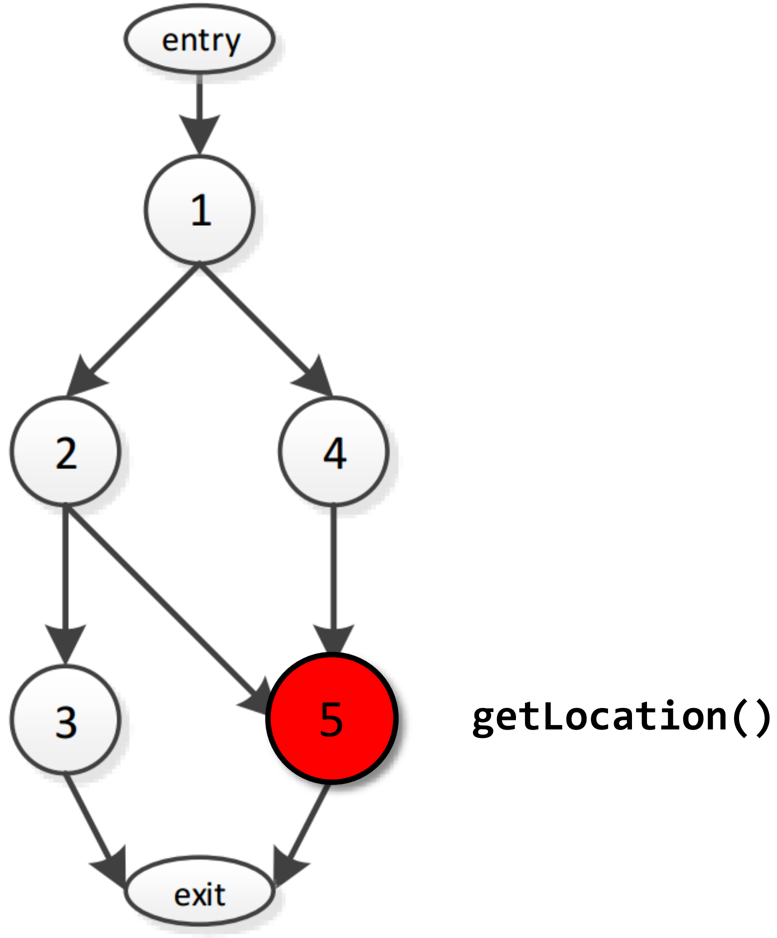
Intuition for Placement

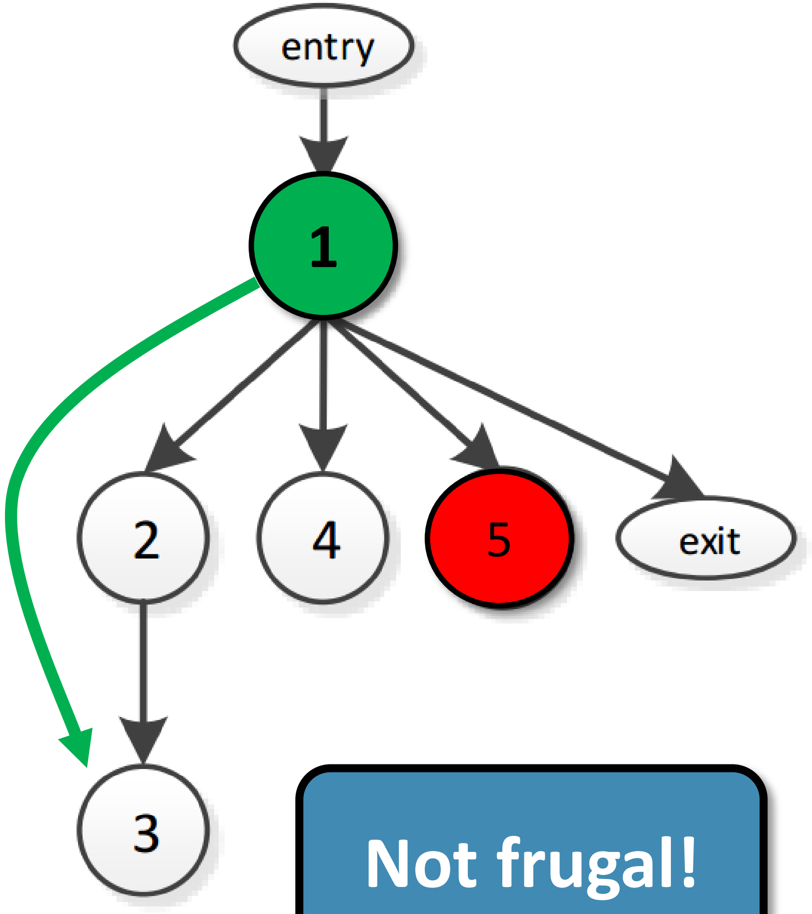
1. Start with a resource access
2. "Move" the prompts up until we are outside of background tasks

- 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*



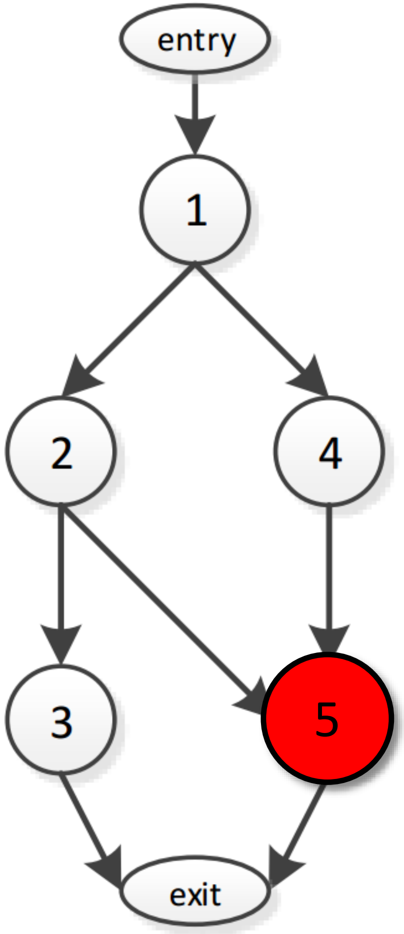
Dominator-Based Approach



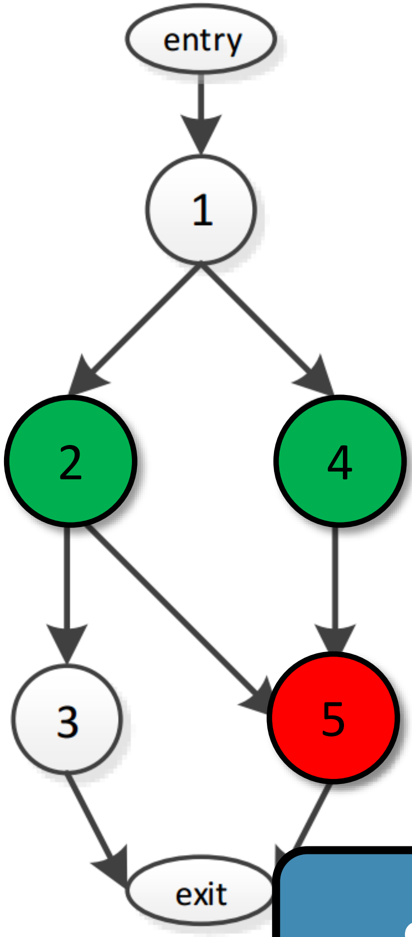


Not frugal!

Backward Placement



`getLocation()`



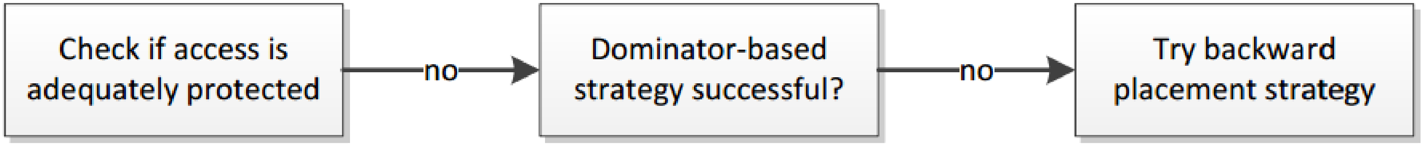
Slower

Analysis Steps

1. For every resource access type and every node $\mathbf{n}$, pre-compute r-anticipated value $A_r(n)$
2. Merge values by meeting them in the semi-lattice of resource types

$$A(n) = \bigwedge A_r(n)$$

3. For every



Check if access is adequately protected

Dominator-based strategy successful?

Try backward placement strategy

EVALUATION

Input Statistics

apps analyzed	100	
app size	7.3MB	
processed methods	352,816	3.5K on average

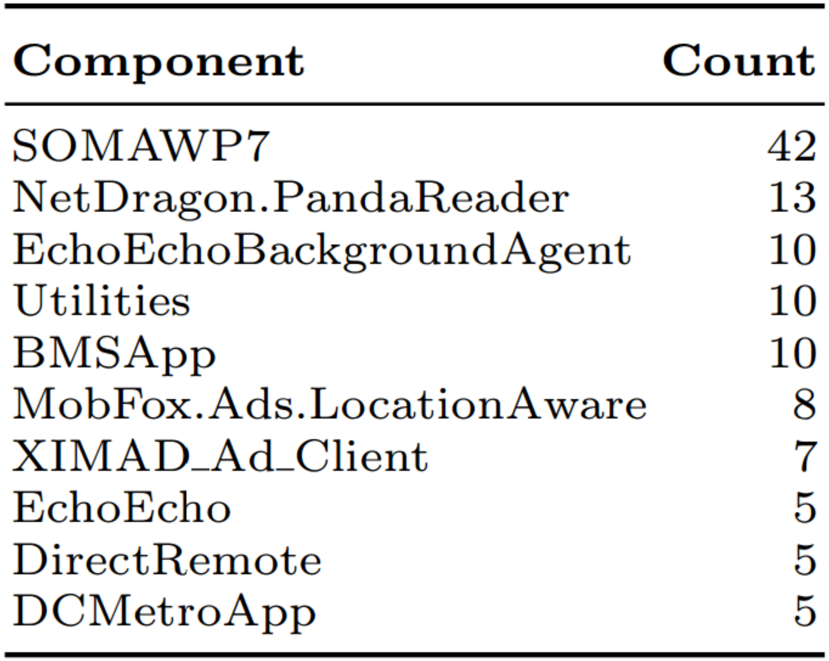
background/library methods	26,033	7%
library methods	25,898	7%

nodes	1,333,056	
anticipating	171,253	12%

accesses	227	2 per app
accesses in background/library methods	78	1/3rd

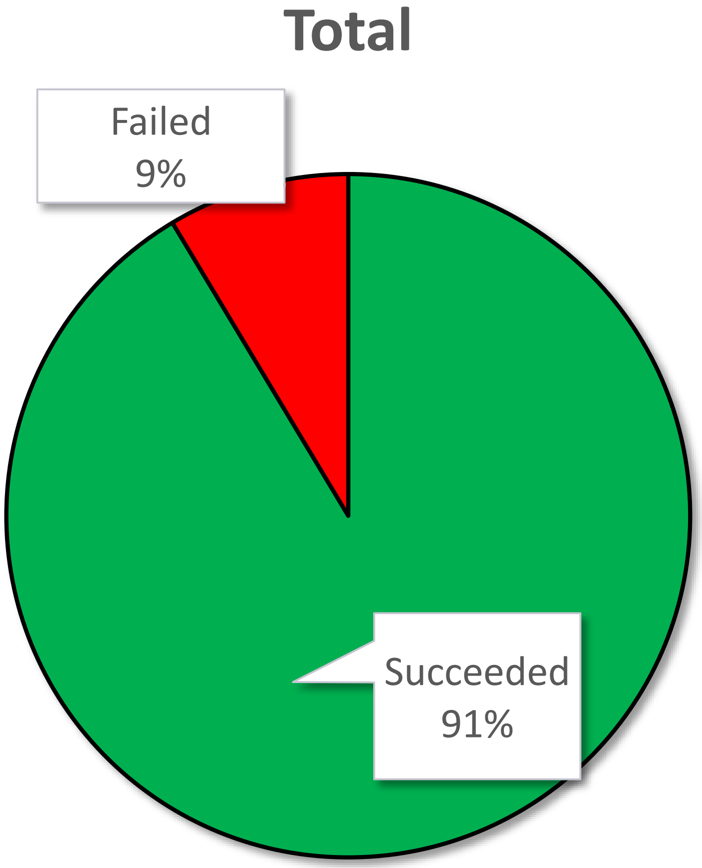
Benchmarks

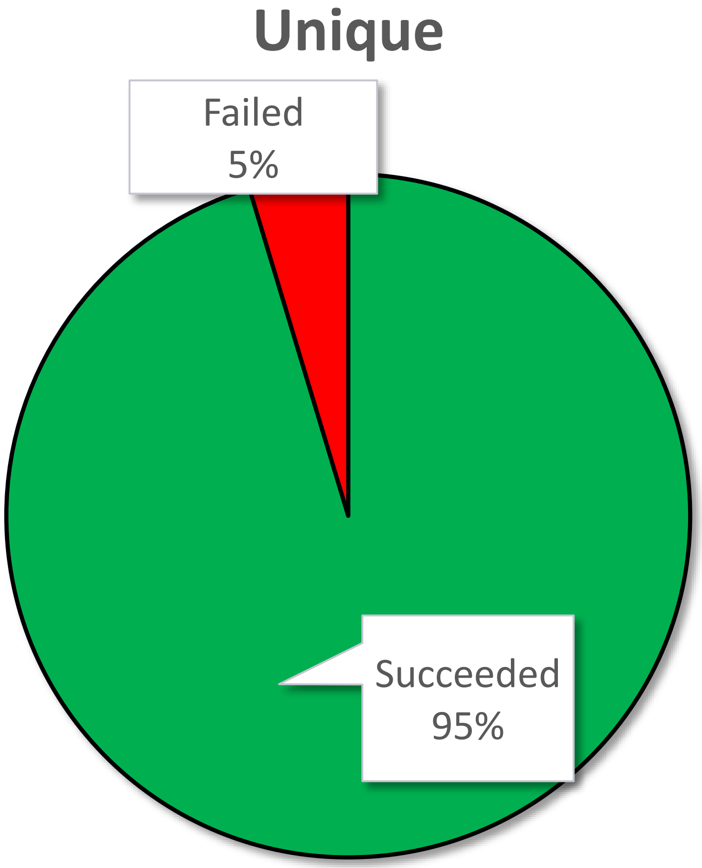
- Took 100 WP 7 apps
- To make this meaningful, chose apps with **LOCATION** and **NETWORKING** caps
- An average app is 7.3 MB of code
- Uses third-party ad libraries



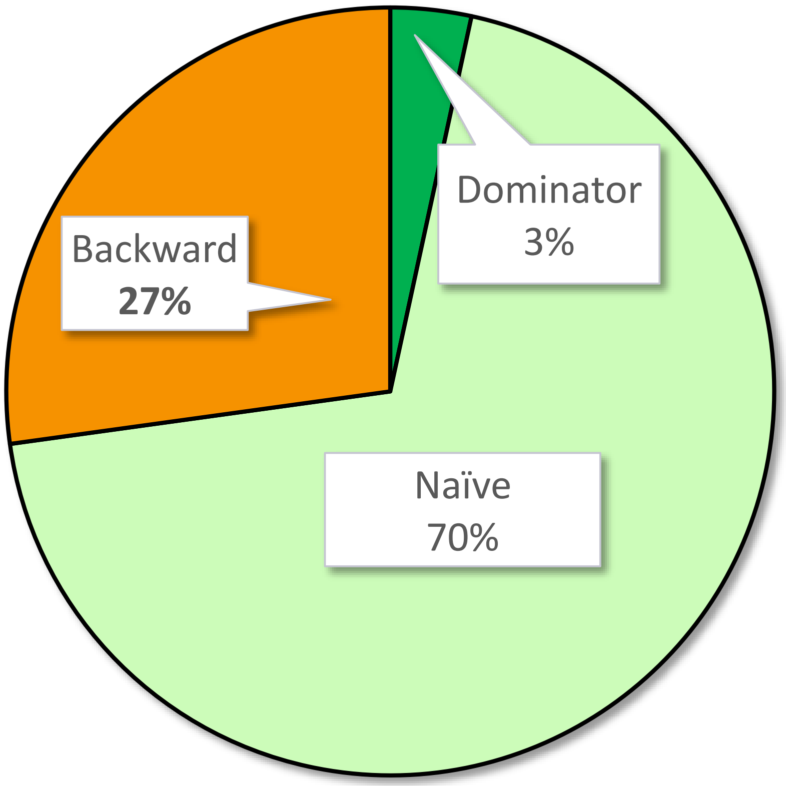
Component	Count
SOMAWP7	42
NetDragon.PandaReader	13
EchoEchoBackgroundAgent	10
Utilities	10
BMSApp	10
MobFox.Ads.LocationAware	8
XIMAD_Ad_Client	7
EchoEcho	5
DirectRemote	5
DCMetroApp	5

Prompt Placement Success



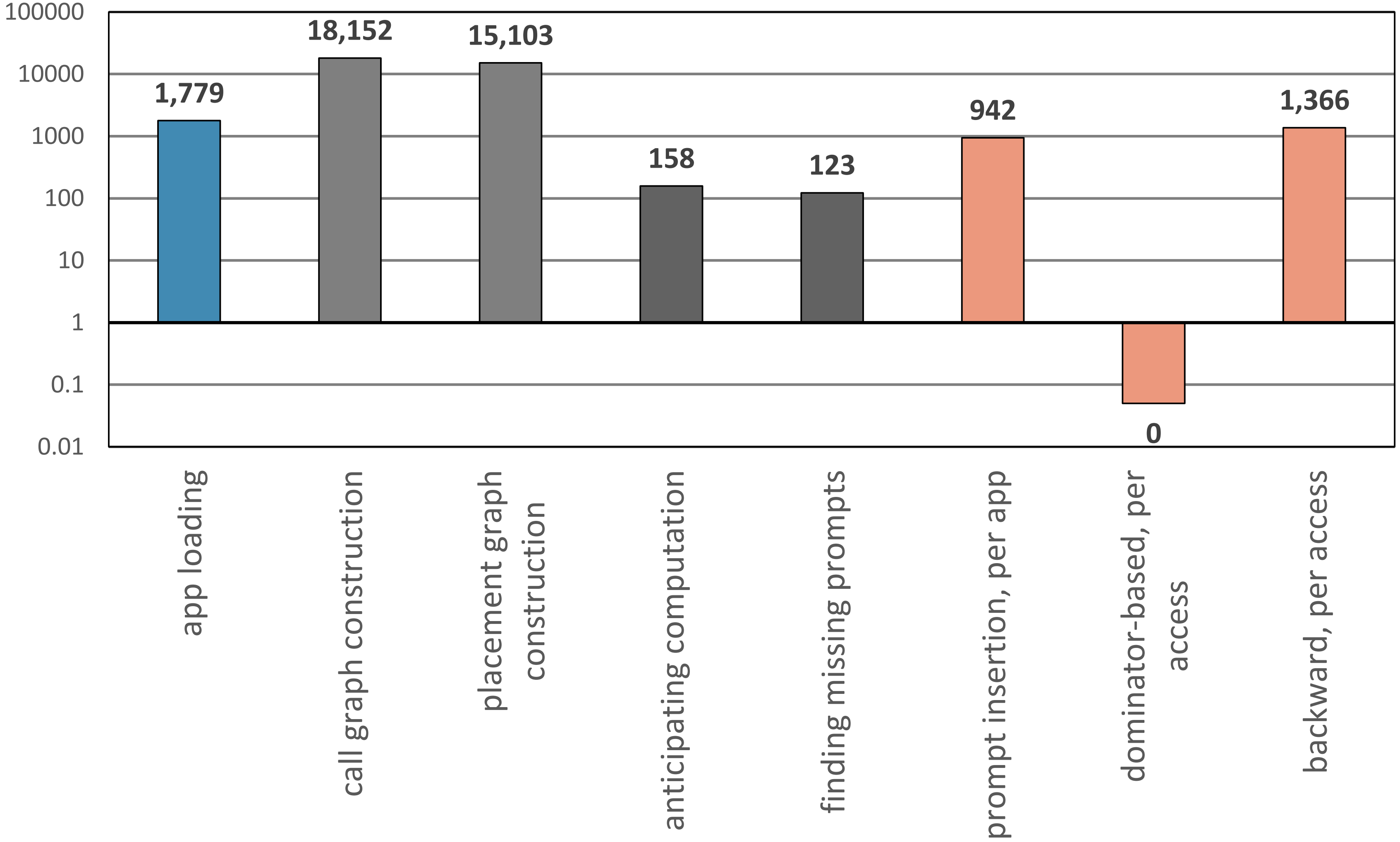


Dominator-Based vs. Backward



- 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


100000
10000
1000
100
10
1
0.1
0.01
1,779
18,152
15,103
158
123
942
0
1,366
app loading
call graph construction
placement graph construction
anticipating computation
finding missing prompts
prompt insertion, per app
dominator-based, per access
backward, per access

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