AppVeto: Mobile Application Self-Defense through Resource Access Veto

Tousif Osman¹, Mohammad Manna¹
Urs Hengartner², Amr Youssef¹

¹Concordia University, ²University of Waterloo
I carefully give permissions, so I am safe!

Flawed Resource Access Model

• Leaks information

• Some resources are not "Dangerous Enough"
Side-channel attacks

A Common side-channel attacks: touch input inference (e.g. Shen et al., 2015)
  • Analyzed Accelerometer, Magnetometer Sensor
  • 83.9% accuracy

Swipe gesture can also be inferred similarly (Aviv et al., 2012)

Less obvious resource, ambient light (R. Spreitzer, 2014)
  • Minor change in the orientation is notable on light sensor
  • 65% accuracy with 5 guesses of PIN

Other Possible Attacks (F. Xu et al., 2019 & A. Sikder et al., 2018)
  • Rogue communication channels using magnetic sensors, Bluetooth, audio
  • Task Inference using magnetic sensors, power analysis
What are the available solutions?

**Smoothing** down the accuracy of motion sensors (Song et al. 2014)

**Randomizing** keys of the keyboard ... with the cost of x3 more time (Song et al. 2014)

Binding the resources with specific Apps only (Demetriou et al. 2015)
Our proposal: AppVeto

An access policy to enable applications to self-defense itself by putting constrains on other applications

Why do we need AppVeto?

• Leave doors open for data leakage
  ✓ Defend against critical data leakage

• Leave doors open for side-channel and covert-channel attacks
  ✓ Defend against known side-channel and covert-channel attacks when it matters
Android application sandbox
Side-channel through resource access
Key Idea: Veto resource access
Hypothesis

Developers know “when”, i.e., passwords, user information, etc.

Request for protections when needed

Protected foreground activity constrains access to vulnerable resources.

**Foreground Activity:** Activity visible on screen and has focus.

**Background App:** Any app with out a foreground activity
Optional Choices

Objectives
• Can be easily distributed among researchers and enthusiasts
• Can be easily deployed with the OS destitution at the same time
• Should not add any more on users

Solutions
• Hook android on the Runtime and change behavior on the fly
• Hooks can directly be called from the OS source code as well
• Delegates responsibilities on the App developers and vendors

Runtime Hooks: Attaching a piece code of with some instruction of a process.
Choices Cont.

**Xposed Framework:** A framework that lets you hook Android Method Calls on the runtime.

**Xposed Module:** An Android App that uses Xposed Framework. Xposed has Module repository (Like an App Store).
Android Activities

main() \sim Activity or Service
  ↦ onCreate()
    ....
  ↦ onResume()
    ....
  ↦ onPause()
    ....
  ↦ onDestroy()
Accessing Android Sensors

Register for callback

System calls back on availability of data
Constraints on Sensor Access

- Control Service
- Know Foreground Activity
- Control Sensor Access
AppVeto: Meta Data Manager

Defines meta keywords

Extract metadata from running applications

```xml
<manifest version="1.0" encoding="utf-8">
<application>
  <meta-data
      android:name="appveto_inference_keystroke"
      android:value="com.example.LoginActivity, com.example.RegisterActivity"/>

  <activity android:name=".LoginActivity"/>
  ...
  ...
</activity>

  <activity android:name=".RegisterActivity"/>
  ...
  ...
</activity>
</manifest>
```
AppVeto: Hook Manager

Entry point of our framework

Intercept method calls and inject methods

Apply our policies on the existing resource access model. e.g. -
  • OS maintains an event queue for sensor callback
  • OS calls “dispatchSensorEvent” on availability of data
  • Hook Manager hooks this method and applies policies before executing the method
  • Hooks other APIs to control Camera and Mic access
AppVeto: Control Service

Control Service

• Hooked methods are executed when that methods are called from other applications
• These methods are executed on different processes
• Learns about foreground and background activity
• Constructs the constraints on the resources
• IPC between the sandbox of different applications

Control Service Client

• Communicates with the service
• Let service know about status of app
Evaluation

Side-channel Evaluation

- Test Applications
- Apps from Google PlayStore

Sensor, Mic, and Camera Pauses and resumes

Exceptions: App misbehaved in some cases.
Performance: CPU usage

<table>
<thead>
<tr>
<th></th>
<th>Without AppVeto</th>
<th>With AppVeto</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average (%)</td>
<td>2.71</td>
<td>3.14</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>0.073</td>
<td>0.86</td>
</tr>
<tr>
<td>Overhead</td>
<td>~0.43%</td>
<td></td>
</tr>
</tbody>
</table>
## Performance: Memory usage

<table>
<thead>
<tr>
<th></th>
<th>Without AppVeto</th>
<th>With AppVeto</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average (MB)</strong></td>
<td>3,397.98</td>
<td>3,445.99</td>
</tr>
<tr>
<td><strong>Std. dev.</strong></td>
<td>40.56</td>
<td>0.40</td>
</tr>
<tr>
<td><strong>Overhead</strong></td>
<td></td>
<td>~48 MB</td>
</tr>
</tbody>
</table>
## Performance: Sensor-access latency

<table>
<thead>
<tr>
<th></th>
<th>Without AppVeto</th>
<th>With AppVeto</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average (ms)</td>
<td>10.0</td>
<td>10.1</td>
</tr>
<tr>
<td>Std. dev.</td>
<td>1.95</td>
<td>1.22</td>
</tr>
<tr>
<td><strong>Overhead</strong></td>
<td>~0.1 ms</td>
<td></td>
</tr>
</tbody>
</table>
Limitations & Future Work

Cannot handle App that uses Native APIs of Android
  • We are working to handle the Native APIs

May lead to Denial of Service
  • To further mitigate this issue we have put a timeout
  • This DoS scenarios needs to be further investigated
  • Timeout period to be further experimented
Takeaways

• Enable finer-grained control on resources access
• Generic solution for side-channel and covert-channel attacks
• Include app vendors and developers in the resource access policy
• Easy to distribute and ready to test solution

Source: https://github.com/tousifosman/app-veto
Email: tousif.osman@mail.concordia.ca