AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings

Giuseppe Petracca
gxp18@cse.psu.edu
The Pennsylvania State University
School of Electrical Engineering and Computer Science
Institute for Networking and Security Research

Ahmad-Atamli Reineh
atamli@cs.ox.ac.uk
University of Oxford, UK
Dept. of Electrical Engineering and Computer Science

Yuqiong Sun
yuqiong_sun@symantec.com
Symantec Research Labs, US

Jens Grossklags
jens.grossklags@in.tum.de
Technical University of Munich, DE

Trent Jaeger
tjaeger@cse.psu.edu
The Pennsylvania State University
School of Electrical Engineering and Computer Science
Institute for Networking and Security Research
Increasing Availability of Privacy-Sensitive Sensors

Controlling when applications may use privacy-sensitive sensors (i.e., cameras, microphones, and touch screens):

- Banking
- Screen Sharing
- Entertainment
- Banking
Abuse of Privacy-Sensitive Sensors

**PlaceRaider: Virtual Theft in Physical Spaces with Smartphones**
Robert Templeman, Zahid Rahman, David Crandall, Apu Kapadia

**Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones.**
Schlegel, Roman Zhang, Keheuan and Zhou, Xiao-yong and Intwala, Mehooll and Kapadia, Apu and Wang, XiaoFeng

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Real World Incidents

Symantec discovered a new HTTP Android Remote administration tool, named Dendroid, available on the underground market for only $300.

Krysanec trojan: Android backdoor lurking inside legitimate apps

Lawsuit claims popular Warriors app accesses phone's microphone to eavesdrop on you

$610K Settlement in School Webcam Spy Case

SAN FRANCISCO — Want to invisibly spy on 10 iPhone owners without their knowledge? Gather their every keystroke, sound, message and location? That will cost you $650,000, plus a $500,000 setup fee with an Israeli outfit called the NSO Group. You can spy on more people if you would like — just check out the company’s price list.

FTC Issues Warning Letters to App Developers Using ‘Silverpush’ Code

Letters Warn Companies of Privacy Risks In Audio Monitoring Technology

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Current Authorization Mechanisms

**Install-Time**

Beginning in Android 6.0 (API level 23), users grant permissions to apps while the app is running, not when they install the app!
Shortcomings

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings

First-Use
Install-Time
Install
Use
Time
Shortcomings

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Shortcomings

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
What Is The Access Control Problem?

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Confused Deputy Problem

- Permission holder (Android system) may be tricked into using its permissions (sensor access) based on a misleading/malicious request (from an app)
  - Typically related to servers being tricked into unauthorized file access
  - E.g., web server serving a password file by mishandling malicious request

- **Goal in this case**: System only grants sensor access when system and user approve
  - How does the Android system validate user approval?
  - How do users know what they are approving?
- **Good news**: Android system can capture all user input events
Proposed Defenses

Input-Driven Access Control (IDAC)
Authorize an operation request that immediately follows a user input event

User inputs associated with operation authorizations

Binding between the user inputs and the authorized operations still unknown to the system!
User-Driven Access Control (UDAC)

Applications must use system-defined gadgets associated with particular operations.

Binding between the user input and the authorized operation explicit to the system

Binding still not explicit to the user!
Limitations of Prior Work

Leverage the **user as weak link** to circumvent protection mechanisms!

“User Interface Attacks”

User may fail to:
- **Identify the application** requesting sensor access
- **Recognize subtle changes** in the Graphic User Interface (GUI)
- **Understand the operation** granted by a particular gadget
User Interface Attacks (Bait-and-Switch)

Interaction #1
User Interface Attacks (Bait-and-Switch)

Interaction #2
User Interface Attacks (Bait-and-Switch)

Interaction #3
User Interface Attacks (Bait-and-Switch)

Interaction #4
User Interface Attacks (Bait-and-Switch)

Interaction #5
User Interface Attacks (Bait-and-Switch)

Interaction #4

The application maintained the windowing display context but switched the widget to record audio.

“Bait-and-Switch Attack”
User Interface Attacks (Application Spoofing)

Window A

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
User Interface Attacks (Application Spoofing)

A click by the user allows the Legitimate App to record audio
User Interface Attacks (Application Spoofing)

A click by the user allows the Spoofing App to record audio.

“Application Spoofing Attack”
User-Driven Access Control (UDAC)

Applications must use system-defined gadgets associated with particular operations.

Compatibility Issue
User-Driven Access Control (UDAC)
Applications must use system-defined gadgets associated with particular operations.

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Research Objectives

- Prevent User Interface Attacks
- Maintain a low authorization effort for the user
- Ensure compatibility with existing applications
- Ensure a performance overhead not perceivable by the user
**Goal:** Prevent applications from changing the mapping between a widget and the associated operation.
Goal: Prevent applications from changing the mapping between a widget and the associated operation
**Goal:** Prevent applications from changing the mapping between a widget and the associated operation

**Insights:**
- Bind each user input event (e) with the widget (w) displayed on the screen by the application (app)
Preventing Operation Switching Attacks

**Goal:** Prevent applications from changing the mapping between a widget and the associated operation

**Insights:**
- Bind each user input event \((e)\) with the widget \((w)\) displayed on the screen by the application \((app)\)
- Intercept the operation request \((op)\) then bind it to the application identity \((app)\) and the set of sensors \((S)\) targeted by the operation

```
capturePhoto()
```
Goal: Prevent applications from changing the mapping between a widget and the associated operation

Insights:
- Bind each user input event (e) with the widget (w) displayed on the screen by the application (app)
- Intercept the operation request (op) then bind it to the application identity (app) and the set of sensors (S) targeted by the operation
- Request the user to authorize the binding request explicitly
AWARE’s Operation Binding

AWARE: Authorization Framework extending OS middleware to make access to privacy-sensitive sensors explicit to both the system and the user

\((\text{app, } S, \text{ op, } e, w, c)\)

- \(\text{app}\) = application associated with widget and operation request
- \(S\) = set of sensors targeted by the request
- \(\text{op}\) = operation being requested
- \(e\) = user input event
- \(w\) = user interface widget
- \(c\) = user interface configuration containing the widget + activity window call graph

System

User

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
AWARE’s Explicit Binding Request

Currently (First-Use)

AWARE

app (Application ID)
AWARE’s Explicit Binding Request

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
AWARE’s Explicit Binding Request

*Requested Operation* (op)

Allow Instagram to use the front Camera to take Pictures when pressing ?

[Buttons: Allow, Deny]
AWARE’s Explicit Binding Request

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
AWARE’s Explicit Binding Request

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Preventing Operation Switching Attacks

**Goal:** Prevent applications from changing the mapping between a widget and the associated operation

**Effect:** Enable the user to verify the association between the operation being authorized \((\text{app, } S, \text{ op})\) and the widget \((w)\) used to obtain the user input event \((e)\) to initiate the operation

**Advantages:**
- Avoid authorizing an unwanted operation by a user input event (IDAC)
- Apps are allowed to choose the widgets to associate with particular operations (UDAC)
Preventing Bait-and-Switch Attacks

**Goal:** Prevent applications from changing the user interface configuration for a widget

G. Petracca *et al.* - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Preventing Bait-and-Switch Attacks

**Goal:** Prevent applications from changing the user interface configuration for a widget

**Insights:**
- Bind the operation request \((\text{app}, S, \text{op})\) with the user interface configuration \((c)\) used to display the widget \((w)\) to elicit a user input event \((e)\)
- Define a display context as set of structural features of the most enclosing activity window containing the widget \((w)\)
**Goal:** Prevent applications from changing the user interface configuration for a widget

**Effects:**
- Identify instance of the same window (i.e., display context) with a **different widget**
- Identify same widget presented in a **different window** (i.e., display context)

**Advantage:** User does not need to check for subtle changes to the widgets or their display context (IDAC and UDAC). Changes detected and flagged by the system automatically.
**Goal:** Prevent applications from replacing the foreground activity window of another application

Activity Window Hijacking
Preventing Application Spoofing Attacks

Goal: Prevent applications from replacing the foreground activity window of another application

Insight: Construct an Activity Window Call Graph (G) where nodes represent activity windows and edges represent enabled transitions (i.e., user inputs or system events)
Preventing Application Spoofing Attacks

**Goal:** Prevent applications from replacing the foreground activity window of another application

**Effects:**
- Activity Window Call Graph \((G)\) built while the application runs
- Record the relationships among windows used by an application

**Advantage:** Identify and block activity window hijacking (IDAC and UDAC)
Maintain a Low Authorization Effort for the User

**Goal:** Limit the number of explicit authorizations by the user

**Insights:**
- Use a *caching mechanism* for operation bindings
- Remove an operation binding from cache if an app changes the way it elicits an operation

**Effect:** “The application will be automatically allowed to perform the requested operation on the set of sensors whenever the user produces the same input event using the same widget within the same user interface configuration”

**Advantages:**
- Require an explicit user's authorization *only the first time* an operation binding is identified (First-Use)
- Ensure that operation bindings do not become stale
- Prevent an operation from being authorized in multiple ways
- Reduce manual authorization by users

G. Petracca *et al.* - **AWARE:** Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Ensure **Compatibility with Existing Apps**

**Goal:** Allow applications to choose how they elicit user approval for use of a sensor

**Insights:**
- No external libraries
- No code annotation
- No app code rewriting
- Dynamic monitoring and creation of operation bindings

**Effect:** Can be integrated with existing off-the-shelf operating systems

**Advantages:**
- Facilitate adoption and deployability
- No effort or burden for app developers

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Experimental Evaluation

Prototyped (Android OS 6.0.1_r5)
Tested (Nexus 5 and Nexus 5X smartphones)

Research Questions:
- To what degree is the AWARE operation binding concept assisting the users in avoiding attacks? (Effectiveness)
- What is the decision overhead imposed to users due to per-configuration access control? (Usability)
- How many existing apps malfunctioned due to the integration of AWARE? (Compatibility)
- What is the performance overhead imposed by AWARE for the operation binding construction and enforcement? (Performance)
Effectiveness

To what degree is the AWARE operation binding concept assisting the users in avoiding attacks?

Laboratory-Based User Study (90 Participants)

Groups: Install-Time, First-Use, Input-Driven, System-Defined Gadgets, and AWARE
Effectiveness

To what degree is the AWARE operation binding concept assisting the users in avoiding attacks?

Laboratory-Based User Study (90 Participants)

Groups: Install-Time, First-Use, Input-Driven, System-Defined Gadgets, and AWARE
Effectiveness

To what degree is the AWARE operation binding concept assisting the users in avoiding attacks?

Experimental Results:

- **TASK 1**: Operation performed by app not visible
  (Exception for Access Control Gadgets)
  Attack Prevention Rate: **First-Use 7% vs AWARE 100% (UDAC)**

- **TASK 2** and **TASK 3**: Users were successfully tricked by switching the user interface configuration!
  Attack Prevention Rate: **UDAC 13% vs AWARE 93%**

- **TASK 4**: Real identity of the app performing the operation was not visible to users
  Attack Prevention Rate: **All Others 7% vs AWARE 100%**
Usability

What is the decision overhead imposed to users due to per-configuration access control?

Field-Based User Study (24 Participants)
- 21 apps (7 categories)*
- 1 week
(Comparison with First-Use)

4 explicit authorizations per-application on average with AWare vs. 2 for first-use

<table>
<thead>
<tr>
<th>App Category</th>
<th>App Name</th>
<th>Explicit User Authorizations</th>
<th>Total Operation Authorizations Avg. (s.d.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>WhatsApp</td>
<td>3</td>
<td>6 (±1)</td>
</tr>
<tr>
<td></td>
<td>Viber</td>
<td>1</td>
<td>1 (±1)</td>
</tr>
<tr>
<td></td>
<td>Messenger</td>
<td>3</td>
<td>7 (±2)</td>
</tr>
<tr>
<td>Recording</td>
<td>Facebook</td>
<td>2</td>
<td>4 (±1)</td>
</tr>
<tr>
<td></td>
<td>SilentEye</td>
<td>2</td>
<td>5 (±1)</td>
</tr>
<tr>
<td></td>
<td>Fideo</td>
<td>2</td>
<td>4 (±1)</td>
</tr>
<tr>
<td>Photo and Video</td>
<td>Ok Screenshot</td>
<td>1</td>
<td>2 (±1)</td>
</tr>
<tr>
<td>Recording</td>
<td>Screenshot Easy</td>
<td>1</td>
<td>2 (±1)</td>
</tr>
<tr>
<td></td>
<td>Screenshot Capture</td>
<td>1</td>
<td>2 (±1)</td>
</tr>
<tr>
<td>Screenshot Capture</td>
<td>REC Screenshot Recorder</td>
<td>2</td>
<td>3 (±1)</td>
</tr>
<tr>
<td>Screen Recording</td>
<td>AZ Screen Recorder Rec.</td>
<td>2</td>
<td>3 (±1)</td>
</tr>
<tr>
<td>Full Screen Mode</td>
<td>Instagram</td>
<td>2</td>
<td>6 (±1)</td>
</tr>
<tr>
<td></td>
<td>Snapchat</td>
<td>2</td>
<td>6 (±1)</td>
</tr>
<tr>
<td></td>
<td>Skype</td>
<td>2</td>
<td>9 (±3)</td>
</tr>
<tr>
<td>Remote Control</td>
<td>Prey Anti Theft</td>
<td>2</td>
<td>8 (±2)</td>
</tr>
<tr>
<td></td>
<td>Lost Android</td>
<td>2</td>
<td>6 (±1)</td>
</tr>
<tr>
<td></td>
<td>Avast Anti-Theft</td>
<td>2</td>
<td>4 (±1)</td>
</tr>
<tr>
<td>Hands-Free Control</td>
<td>Google Voice Search</td>
<td>1</td>
<td>1 (±1)</td>
</tr>
<tr>
<td></td>
<td>HappyShutter</td>
<td>1</td>
<td>1 (±0)</td>
</tr>
<tr>
<td></td>
<td>SnapClap</td>
<td>1</td>
<td>1 (±0)</td>
</tr>
</tbody>
</table>

*www.statistica.com

G. Petracca et al. - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
How many existing apps malfunctioned due to the integration of AWARE?

**Android Compatibility Test Suite (CTS):**
- 1,000 apps (Google Play)
- 13 hours and 28 minutes

**Experimental Results:**
- **126,681 passed** tests over **126,686**
- [Viber] Camera and microphone probing at reboot (No impact on video or voice calls)
Performance

*What is the performance overhead imposed by AWARE for the operation binding construction and enforcement?*

**Android UI/Application Exerciser Monkey:**
- 1,000 apps (Google Play)
- Nexus 5 and Nexus 5X

**Microbenchmark:**
- Access requests for operation targeting privacy-sensitive sensors
- 10,000 operations
- About 3% overhead on microbenchmarks

**Experimental Results:**
- **0.33%** system-wide performance overhead
- About **3 MB of memory** for the operation binding cache and window call graphs

G. Petracca *et al.* - AWARE: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings
Conclusion

- Authorization of sensor operations *explicit* to both system and user (*Operation Binding + Explicit Authorization*)
  - Up to 100% user interface attack prevention (only up to 13% with alternative approaches)

- Low user effort (*Caching of Bindings* when the user interface configuration is same for the same operation)
  - 4 explicit authorizations per-application on average

- Compatible with existing applications (*No app modification or redesign*)
  - Only 5 minor compatibility issues out of 1,000 tested apps

- Negligible Performance Overhead (limited number of authorization hooks and quick retrieval of bindings)
  - 0.33% performance overhead and 3 MB of memory
Thank You
For Your Attention!

Giuseppe Petracca
Ph.D. Candidate
gxp18@cse.psu.edu
https://sites.psu.edu/petracca/

Source Code: https://github.com/gxp18/AWare

Research Funded by:
**Approach Overview**

**AWAre Authorization Workflow**

1. User Interface (Widgets)
2. User Interface (AppID, Widgets)
3. User Input (AppID, Widget, Configuration)
4. User Input Events (AppID, Widget, Configuration)
5. Input Events (Widget, InputEvent)
6. Operation Request (Operation, Sensors)
11. Request (AppID, Operation, Sensors)
12. User Notification (AppID, Operation, Sensors)
13. Data (Sensors)

G. Petracca et al. - AWAre: Preventing Abuse of Privacy-Sensitive Sensors via Operation Bindings