Applying the Opacified Computation Model to Enforce Information Flow Policies in IoT Applications

Amir Rahmati, Earlence Fernandes, Atul Prakash
Applying the Opacified Computation Model to Enforce Information Flow Policies in IoT Applications

Amir Rahmati, Earlence Fernandes, Atul Prakash
INTERNET of THINGS
INTERNET of THINGS

Wearables

Smart Home

Connected Health
INTERNET of THINGS

Wearables

Smart Home

Connected Health

Frameworks

Android

Weave

Google

Samsung

SmartThings

Apple

HomeKit

IoTivity
Smart Light

Location

Internet

App

Switch
Smart Light

Permissions

Access Control
Location
Internet
Switch
Smart Light

Permissions

Access Control
Location
Internet
Switch

Flow Control
Location
Switch
Location
Internet
Enable apps to compute on sensitive data while mitigating data abuse
Label-based Flow Control

- Component level information tracking
- Enforce flows through label policies
Label-based Flow Control
• Component level information tracking
• Enforce flows through label policies

Language-based Flow Control
• Restructure apps to obey flow rules
• Developer declares flows
Label-based Flow Control
• Component level information tracking
• Enforce flows through label policies

Language-based Flow Control
• Restructure apps to obey flow rules
• Developer declares flows

FlowFence
• Supports source- and user-approved data flows
• Allows use of existing languages, tools, and OSes
Primitives
Primitives

Location → Presence Detector → Presence Status
Primitives

Location → Presence Detector → Presence Status

Location → Presence Detector → Presence Status

Taint → Presence Detector → Quarantined Module → Presence Status

Sandbox
Primitives

![Diagram showing the flow of data through a Presence Detector and a Quarantined Module within a Sandbox.](image)
Primitives

Location → Presence Detector → Presence Status

Location → Presence Detector (Quarantined Module) → Presence Status

Sandbox → Opaque Handle
Primitives

Location → Presence Detector → Presence Status

Location → Presence Detector → Quarantined Module → Presence Status

Sandbox → Opaque Handle
Primitives

Sandbox

Provide two methods for data sharing:
- Key-value store
- Event channels
Primitives

Sandbox

Opaque Handles

do **NOT** reveal
• Raw data
• Data type
• Taint label
• Data size
• Exceptions
to code not running in a QM

Sandboxes

Provide two methods for data sharing:
• Key-value store
• Event channels
Policy Decisions

Quarantined Module

Sandbox
Policy Decisions

```
T1
Trusted Source

T2
Opaque Handle

Quarantined Module

Sandbox
```
Policy Decisions

- Trusted Source
- Opaque Handle
- Quarantined Module
- Sandbox

T₁, T₂

...
Policy Decisions

- Trusted Source
- Quarantined Module
- Trusted Sink
- Opaque Handle
- Opaque Handle

\[ T_1, T_2 \] → Sink
Policy Decisions

Trusted Source → Quarantined Module → Trusted Sink

Opaque Handle → Sandbox → Opaque Handle

Source-Approved Policy

T₁, T₂ → Sink
Policy Decisions

- Trusted Source
- Opaque Handle
- Quarantined Module
- Sandbox
- Trusted Sink
- Opaque Handle

Source-Approved Policy
User-Approved Policy
SmartLight
SmartLight Policy

```xml
<?xml version="1.0" encoding="utf-8"?>
<FlowFenceManifest>
  <policy>
    <allow flowfence:src="locationTaint"
           flowfence:sink="SmartThings.SmartSwitch"/>
  </policy>
</FlowFenceManifest>
```
<xml version="1.0" encoding="utf-8">  
<FlowFenceManifest ...>  
  <policy>  
    <allow flowfence:src="locationTaint" flowfence:sink="SmartThings.SmartSwitch" />  
  </policy>  
</flowfenceManifest>
**SmartLight Policy**

```
<?xml version="1.0" encoding="utf-8"?>
<FlowFenceManifest ...
<policy>
<allow flowfence:src="locationTaint"
flowfence:sink="SmartThings.SmartSwitch" />
</policy>
</flowfenceManifest>
```

**Location Service Policy**

```
<?xml version="1.0" encoding="utf-8"?>
<FlowFenceManifest ...
<event-channel flowfence:name =
"presenceUpdateChannel" flowfence:exported =
"both" />
</flowfenceManifest>
```
SmartLight Policy

Location Service Policy

```xml
<?xml version="1.0" encoding="utf-8"?>
<FlowFenceManifest ...
  <policy>
    <allow flowfence:src="locationTaint"
      flowfence:sink="SmartThings.SmartSwitch" />
  </policy>
</flowfenceManifest>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<FlowFenceManifest ...
  <event-channel flowfence:name =
    "presenceUpdateChannel" flowfence:exported =
    "both" />
</flowfenceManifest>
```
SmartLight

Location

Internet

App

Switch

SmartLight Policy

Location Service Policy

```xml
<?xml version="1.0" encoding="utf-8"?>
<FlowFenceManifest ...>
  <policy>
    <allow flowfence:src="locationTaint"
      flowfence:sink="SmartThings.SmartSwitch" />
  </policy>
</FlowFenceManifest>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<FlowFenceManifest ...>
  <event-channel flowfence:name =
    "presenceUpdateChannel" flowfence:exported =
    "both" />
</FlowFenceManifest>
```
Normal Structure
Normal Structure

Location

Presence Detector
Normal Structure

```java
firebaseRef.child(LOC_KEY).addValueEventListener
  (new ValueEventListener()) {
    public void onDataChange (DataSnapshot
      dataSnapshot) {
      String presence = (String)
        dataSnapshot.getValue();
      toggleSwitch(presence);
    }
  }
...
Normal Structure

```java
1. firebaseRef.child(LOC_KEY).addListenerForSingleValueEvent(new ValueEventListener) {
2.     public void onDataChange(DataSnapshot dataSnapshot) {
3.         String presence = (String) dataSnapshot.getValue();
4.         toggleSwitch(presence);
5.     }
6. }
7. ...
8. }
9.
10. (a) Default:
```
Normal Structure

```java
private void toggleSwitch(String presence) {
    if (!history.equals(presence)) {
        if (presence.equals("home")) {
            Log.i(TAG, "let there be light!");
            List<SmartSwitch> switches =
                SmartThingsService.getInstance().j
                getSwitches();
            if (switches != null) {
                for (SmartSwitch ssw : switches) {
                    SmartThingsService.getInstance().j
                    switchOnOff("on",
                    ssw.getSwitchId());
                }
            }
        } else if (presence.equals("away")) {
            Log.i(TAG, "lights off!");
            List<SmartSwitch> switches =
                SmartThingsService.getInstance().j
                getSwitches();
            if (switches != null) {
                for (SmartSwitch ssw : switches) {
                    SmartThingsService.getInstance().j
                    switchOnOff("off",
                    ssw.getSwitchId());
                }
            } else {
                history = presence;
            }
        }
    }
}
```
Presence Detector

(b) FlowFence
Presence Detector

```
public class PresenceQM implements Parcelable {
    // ...

    public static void putLoc(String presenceVal) {
        // Write presence value to KV store
        SharedPreferences myPrefs = FlowFenceContext.getInstance().getSharedPreferences("presenceKVS", Context.MODE_WORLD_READABLE);
        SharedPreferences.Editor edit = myPrefs.edit();
        edit.putString("location", presenceVal);
        edit.apply();

        // Fire an event to any listening QM
        EventChannelApi eventApi = (EventChannelApi) FlowFenceContext.getInstance().getTrustedAPI("event");
        eventApi.fireEvent(buildTS, ComponentName.unflattenFromString("presenceChannel"));
        Log.i("PresenceQM", "updated KV with value: " + presenceVal + ", and fired channel event");
    }
    // ...
}
```
public class ResponderQM implements Parcelable
{
    public static void pollPresenceAndCompute()
    {
        // Read updated presence value from KV store
        SharedPreferences presencePrefs = FlowFenceContext.getInstance().j
                                            .createPackageContext("presenceQM",
                                                0).getSharedPreferences("PresenceKVS",
                                                    Context.MODE_WORLD_READABLE);
        String presence = presencePrefs.getString("location", "null");

        // Read previous presence value from KV store
        SharedPreferences myprefs = FlowFenceContext.getInstance().j
                                        .getSharedPreferences("hist_store",
                                            Context.MODE_WORLD_READABLE);
        String history = myprefs.getString("history", "");

        // Toggle switch function
        if (history.equals(presence)) {
            String op = "null";
            if (presence.equals("home")) {
                Log.i(TAG, "let there be light!");
                op = "on";
            } else if (presence.equals("away")) {
                Log.i(TAG, "lights off!");
                op = "off";
            }
        } else if (op != null) {
            ISmartSwitchAPI switchAPI = (ISmartSwitchAPI) FlowFenceContext.j
                                            .getInstance().getTrustedAPI("smartswitch");
            List<SmartDevice> switches = switchAPI.getSwitches();
            if (switches != null) {
                for (SmartDevice ssw : switches) {
                    switchAPI.switchOp(op, ssw.getId());
                }
            }
        }
        history = presence;

        // Store new presence value in KV store
        SharedPreferences.Editor edit = myprefs.edit();
        edit.putString("history", history);
        edit.commit();
    }
}
Future Work
Future Work

• Information flow tracking across multiple environments
Future Work

• Information flow tracking across multiple environments

• Mitigating side channel
Future Work

• Information flow tracking across multiple environments
• Mitigating side channel
• Policy management
Future Work

• Information flow tracking across multiple environments
• Mitigating side channel
• Policy management

FlowFence code will be released on December 1st
https://iotsecurity.eecs.umich.edu
Opacified Computation

• Enables practical data flow control for IoT applications.

• Uses Quarantined Modules and Opaque Handles to Explicitly embed control and data flows within app structure.

• Supports publisher and consumer flow policies.

• FlowFence code will be released on December 1st.
  
  https://iotsecurity.eecs.umich.edu
Opacified Computation

- Enables practical data flow control for IoT applications.

- Uses Quarantined Modules and Opaque Handles to Explicitly embed control and data flows within app structure.

- Supports publisher and consumer flow policies.

- FlowFence code will be released on December 1st. https://iotsecurity.eecs.umich.edu