FlowFence: IoT security

Mikaël Fourrier
Internet of Things

- Interconnection of numerous devices which interacts and exchange data
- Examples: smart home, smart grid
- Vague term, like the Cloud
Study: Samsung SmartThings

- Subscribe: abstraction of the hardware
- Polling
- Access control with a device-level granularity
Study: Google Fit

- Wearables-oriented
- Only callbacks
- Access control with scopes
  - Ex: FITNESS_BODY_READ
Study: Android Sensor API

- Events: Motion, Environment, Position
- Callback-based except for Position
- No access control except for Position and heart rate
Study: IoT architecture

- Hub
- Cloud
Problems with IoT

• Lots of devices → hard to secure
• Very sensitive data: health, home locking, cameras
• Third-party applications have few restrictions: a face-recognition door unlocker can send images to the network
FlowFence: basic ideas

- Normal execution environment vs sandbox (Quarantined Modules)
- Use of opaque handles
- Enforce declared data use patterns
- Sandbox treated as a black box
API example

```java
application DoorCon
request { Taint_CAMERA -> Door.Open,
            Taint.DOORSTATE -> Door.Open,
            Taint.DOORSTATE -> Internet }

void QM_recog(faceBmp, status):
    Features f = extractFeatures(faceBmp);
    if(status != unlocked AND isAuthenticated(f))
        TrustedAPI.door[0].open();

void QM_report(status):
    TrustedAPI.network.send(status);

void QM_malloc(faceBmp):
    /* this is denied */
    TrustedAPI.network.send(faceBmp);

receive hCam from CamPub;
Handle hStatus =
    DoorStatePub.getDoorState();
QM.call(QM_recog, hCam, hStatus);
QM.call(QM_malloc, hCam);
QM.call(QM_report, hStatus);
```
Publisher examples

```java
application CamPub
taint_label Taint_CAMERA;
allow { Taint_CAMERA -> UI }

Bitmap QM_bmp():
    Bitmap face = camDevice.snapshot();
    return face;

if (motion at FrontDoor)
    hCam = QM.call(QM_bmp, Taint_CAMERA);
    send hCam to DoorCon;

application DoorStatePub
taint_label Taint_DOORSTATE;

Status QM_status():
    return (door[0].state(), 0); //state, idx

/* IPC */ Handle getDoorState():
    return QM.call(QM_status,
                   Taint_DOORSTATE);
```
## Taint arithmetic

<table>
<thead>
<tr>
<th>Operation</th>
<th>Taint Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandbox $S$ loads a QM</td>
<td>$T[S] := \emptyset$</td>
</tr>
<tr>
<td>QM inside $S$ reads opaque handle $d = OH^{-1}(h)$</td>
<td>$T[S] + = T[h]$</td>
</tr>
<tr>
<td>QM inside $S$ returns $h = OH(d)$</td>
<td>$T[h] := T[S]$</td>
</tr>
<tr>
<td>QM manually adds taints ${t}$ to its sandbox</td>
<td>$T[S] + = {t}$</td>
</tr>
<tr>
<td>$QM_0$ inside $S_0$ calls $QM_1$ inside $S_1$</td>
<td>$T[S_1] = T[S_0]$</td>
</tr>
</tbody>
</table>

Table 1: Taint Arithmetic in FlowFence. $T[S]$ denotes taint labels of a sandbox running a QM. $T[h]$ denotes taint label of a handle $h.$
Figure 2: FlowFence Architecture. Developers split apps into Quarantined Modules, that run in sandbox processes. Data leaving a sandbox is converted to an opaque handle tainted with the sandbox taint set.
Sandboxes

• Android process with the “isolatedProcess” flag
  – Disable all rights except IPC for FlowFence

• Cleaned after QM execution
Key-value store

- key → (sensible value, taint)
- Polling easy to implement
- Event channels for callbacks
- Device agnostic
Overhead

- **3M/sandbox**
  - reasonable

- **100ms if spare sandboxes**
  - same as network call

- **30M/s bandwidth**
  - the Nest camera uses 1M/s, so should be sufficient
## Ported applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Security Risk without FlowFence</th>
<th>LoC original</th>
<th>LoC FlowFence</th>
<th>Flow Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmartLights [47]</td>
<td>Reads a location beacon and if the beacon is inside a geofence around the home, automatically turn on the lights</td>
<td>App can leak user location information</td>
<td>118</td>
<td>193</td>
<td>loc $\rightarrow$ switch</td>
</tr>
<tr>
<td>FaceDoor [34]</td>
<td>Uses a camera to recognize a face; If the face is authorized, unlock a doorlock</td>
<td>App can leak images of people</td>
<td>322</td>
<td>456</td>
<td>cam $\rightarrow$ lock, doorstate $\rightarrow$ lock, doorstate $\rightarrow$ net</td>
</tr>
<tr>
<td>HeartRateMonitor [67]</td>
<td>Uses a camera to measure heart rate and display on UI</td>
<td>App can leak images of people, and heart rate information</td>
<td>257</td>
<td>346</td>
<td>cam $\rightarrow$ ui</td>
</tr>
</tbody>
</table>

Table 2: Features of the three IoT apps ported to FlowFence. Implementing FlowFence adds 99 lines of code on average to each app (less than 140 lines per app).
Weaknesses

- QM could forge keys to leak data
  - Keys must already exist in the QM
- QM can control it's execution time
  - Asynchronous execution in future version
- Can't prevent user to approve all
- Over-tainting
  - Taint bound