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

```
1  application DoorCon
2  request { Taint_CAMERA -> Door.Open,
3            Taint_DOORSTATE -> Door.Open,
4            Taint_DOORSTATE -> Internet }
5
6  void QM_recog(faceBmp, status):
7    Features f = extractFeatures(faceBmp);
8    if(status != unlocked AND isAuth(f))
9      TrustedAPI.door[0].open();
10
11 void QM_report(status):
12   TrustedAPI.network.send(status);
13
14 void QM_mal(faceBmp):
15   /* this is denied */
16   TrustedAPI.network.send(faceBmp);
17
18 receive hCam from CamPub;
19 Handle hStatus =
       DoorStatePub.getDoorState();
20 QM.call(QM_recog, hCam, hStatus);
21 QM.call(QM_mal, hCam);
22 QM.call(QM_report, hStatus);
```

Publisher examples

```
1  application CamPub
2  taint_label Taint_CAMERA;
3  allow { Taint_CAMERA -> UI }
4
5  Bitmap QM_bmp():
6    Bitmap face = camDevice.snapshot();
7    return face;
8
9  if (motion at FrontDoor)
10   hCam = QM.call(QM_bmp, Taint_CAMERA);
11   send hCam to DoorCon;
12 ----------------------------------------
13 application DoorStatePub
14 taint_label Taint_DOORSTATE;
15
16 Status QM_status():
17   return (door[0].state(), 0); //state,idx
18
19 /* IPC */ Handle getDoorState():
20   return QM.call(QM_status,
         Taint_DOORSTATE);
```

Taint arithmetic

Operation	Taint Action
Sandbox S loads a QM	$T[S] := \varnothing$
QM inside S reads opaque handle $d = OH^{-1}(h)$	$T[S] \mathrel{+}= T[h]$
QM inside S returns $h = OH(d)$	$T[h] := T[S]$
QM manually adds taints $\{t\}$ to its sandbox	$T[S] \mathrel{+}= \{t\}$
QM_0 inside S_0 calls QM_1 inside S_1	$T[S_1] = T[S_0]$

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.

Architecture

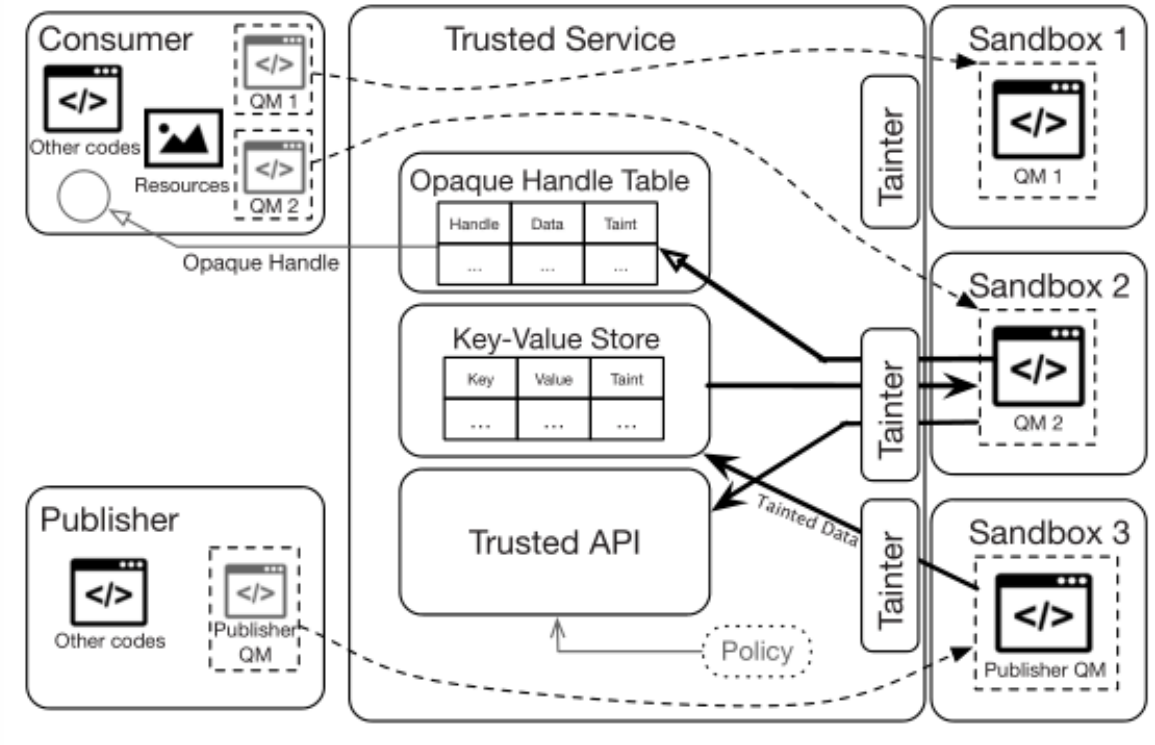

Figure 2: FlowFence Architecture. Developers split apps into Quarantined Modules, that run in sandbox processes. Data leaving a sandbox is converted to an opaque handle tainted with the sandbox taint set.

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

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

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