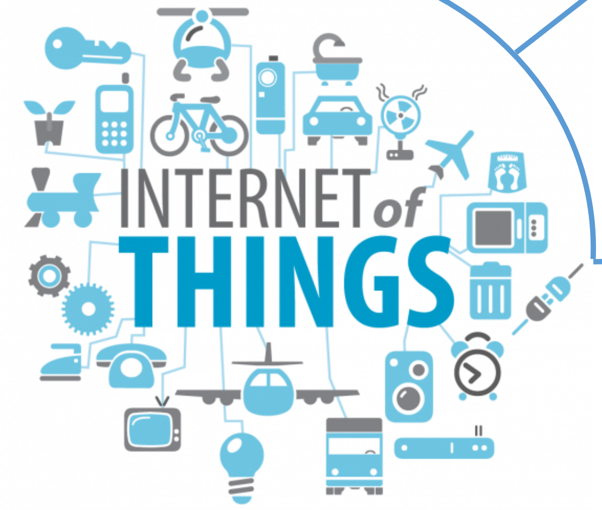
FlowFence: Practical Data Protection for Emerging IoT Application Frameworks

Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato,
Mauro Conti, Atul Prakash

USENIX Security Symposium
11 August 2016

Wearables/Quantified Self

Connected Healthcare



Smart Homes

WEAVE On WEAVE

Google Brillo inside

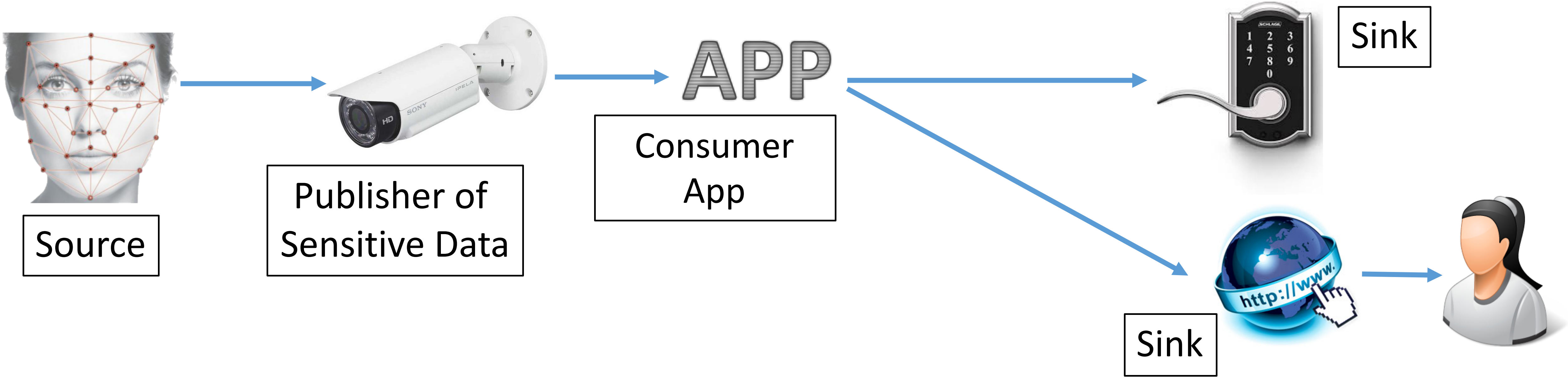
HomeKit

SAMSUNG SmartThings

amazon alexa

android wear

Emerging IoT App Frameworks



- Unlock door if face is recognized
- Home-owner can check activity from Internet

- App needs to compute on sensitive data to provide useful service
- But has the potential to leak data

Fernandes et al., Security Analysis of Emerging Smart Home Applications, S&P 2016

How can we enable apps to compute on the *sensitive* data the IoT generates while *mitigating data abuse*?

Existing IoT frameworks only have permission based access control

Smart home API

[Smart Homes]

Google Fit API

[Wearables]

Android Sensor API

[Quantified Self]

e.g., capability.lockCodes in SmartThings (pincodes), FITNESS_BODY_READ scope in Google Fit (heart rate)

- Permissions control what data an app can access
- Permissions do not control how apps use data, once they have access

Instruction-Level Flow Analysis Techniques

Dynamic Taint Tracking

+ Fine granularity
+ No developer effort
- High computational overhead
- May need special h/w for acceleration
- Implicit flows can leak information
- Limited OS/Language flexibility

IoT devices (and hubs) have constrained hardware

OS and Language Diversity; [Supports Rapid Development]

Static Taint Tracking

+ Fine granularity
+ No developer effort
- Implicit flows can leak information
- IPC and async. code can leak information

Fundamental Trigger-Action Nature of IoT apps = Lots of async. code

FlowFence

Flow-control is a first-class primitive

Label-based flow control

- Component-level information tracking
- Flow enforcement through label policies

+

Language-based flow control

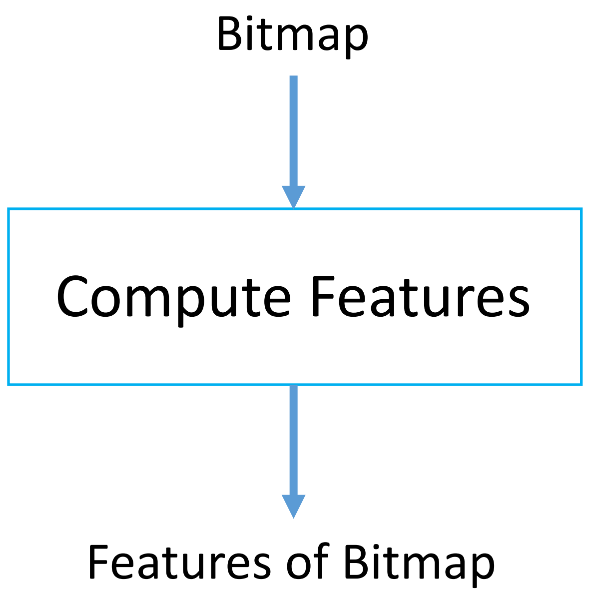
- Restructure apps to obey flow rules
- Developer declares flows

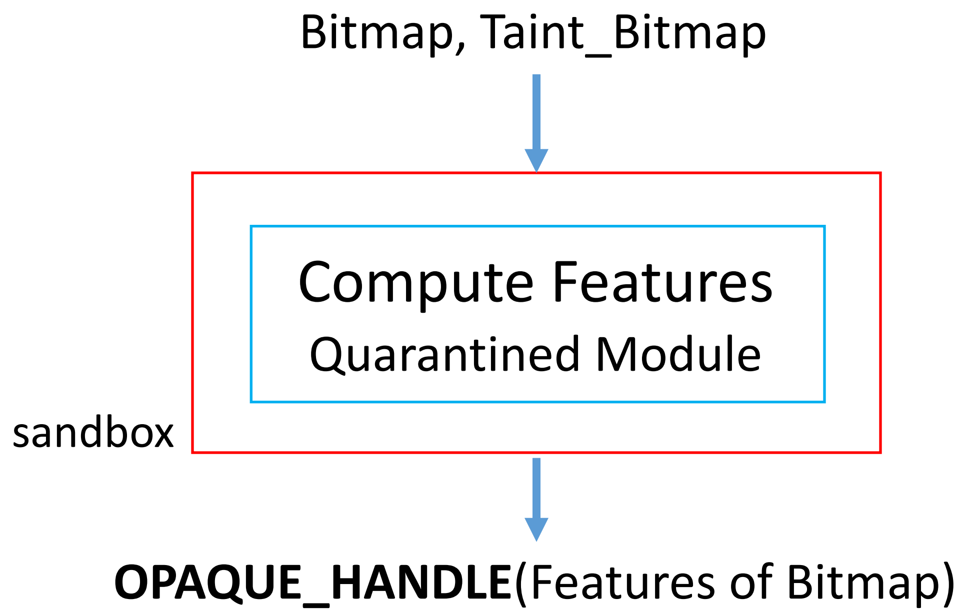
=

FlowFence

- Support of diverse publishers and consumers of data, with publisher and consumer flow policies
- Allows use of existing languages, tools, and OSes

FlowFence Primitives – Quarantined Modules and Opaque Handles

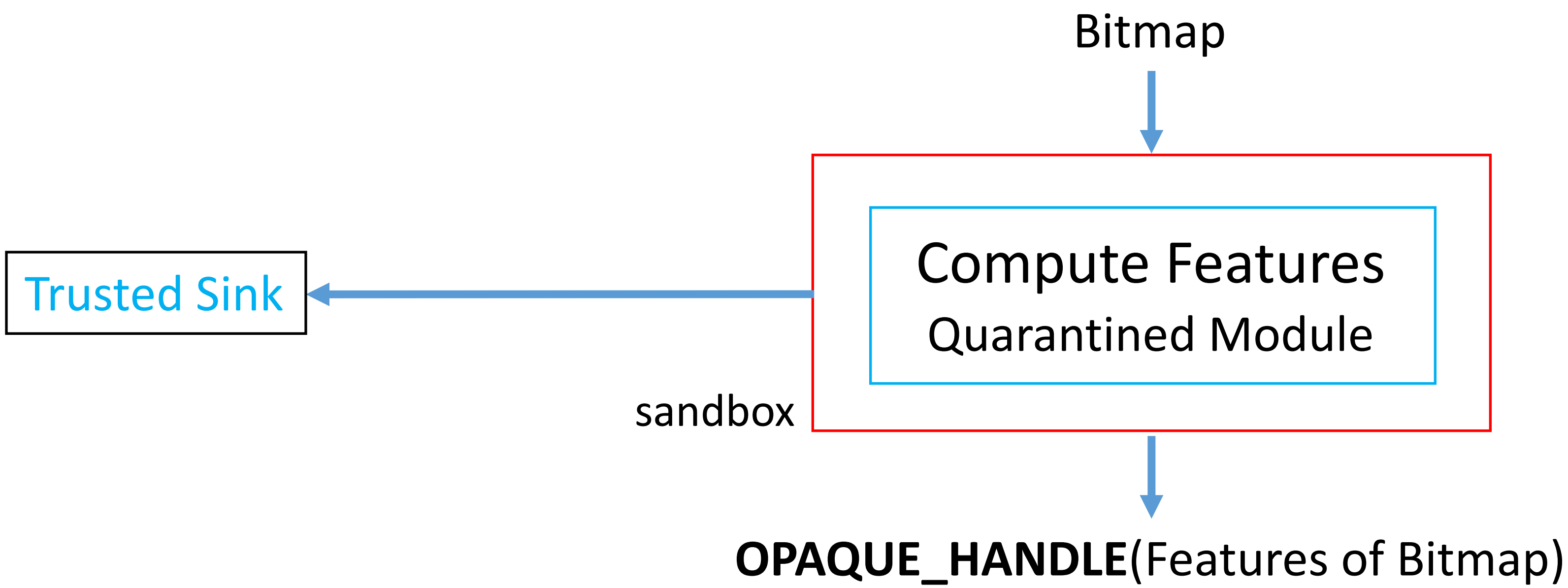




- The computation runs with the rights to access sensitive bitmap data

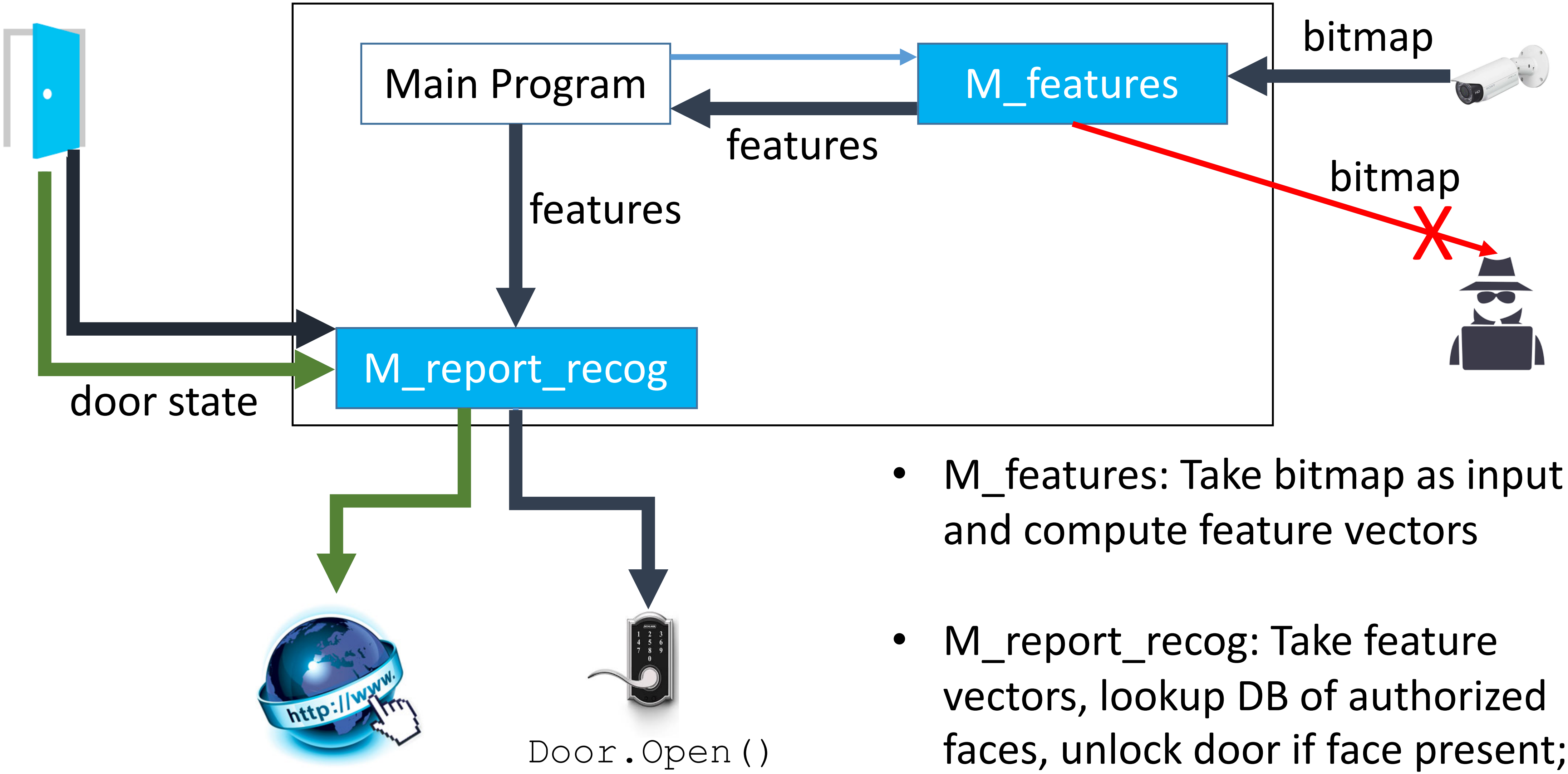
- Submit a computation that runs in a sandbox
- All sensitive data is available only in sandboxes

FlowFence Primitives – Quarantined Modules and Opaque Handles



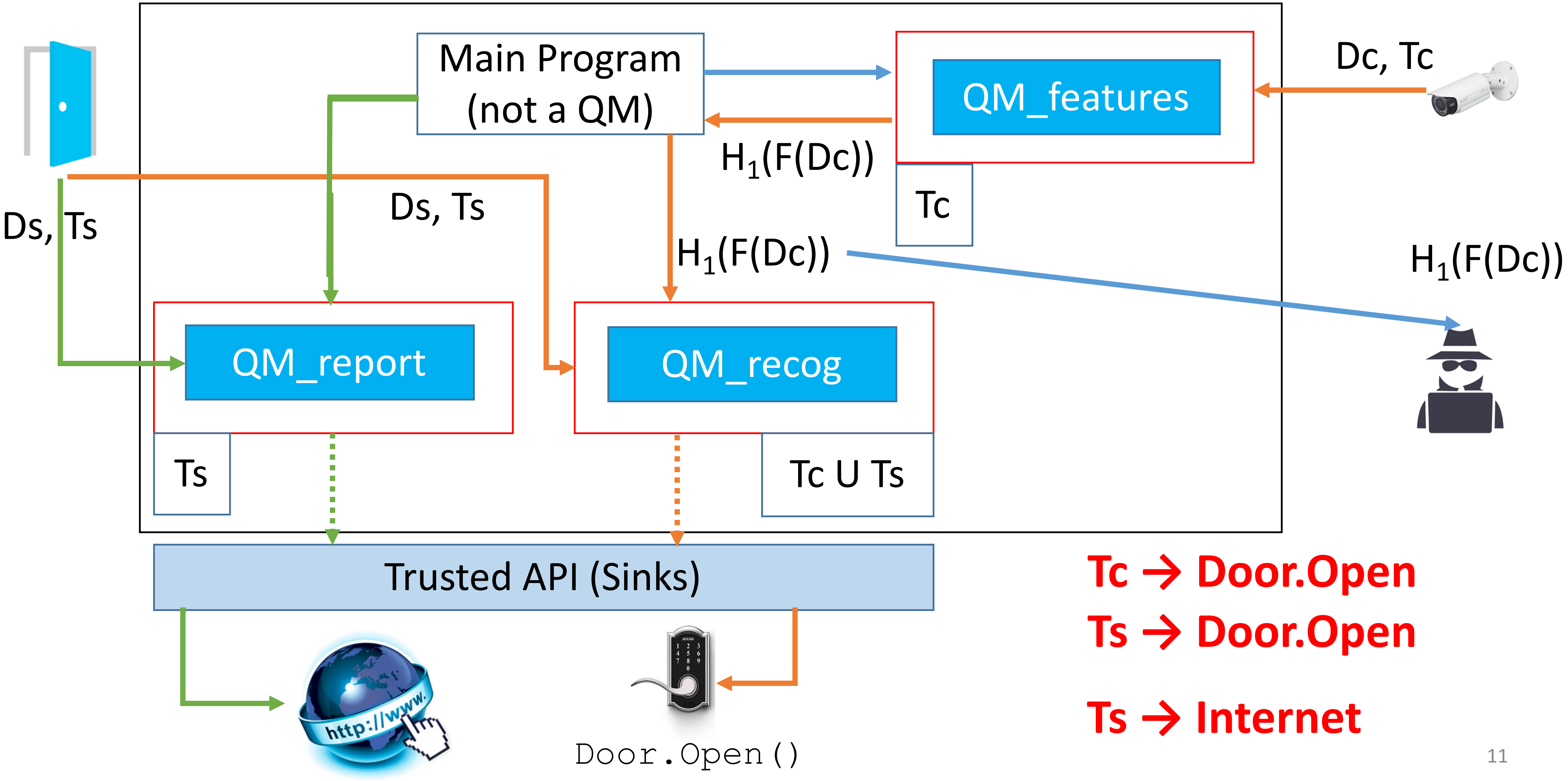
- Quarantined Modules can also access FlowFence-provided Trusted Sinks
- Trusted Sinks check the taint labels of the caller against a flow policy

Face Recognition App Example



- M_features: Take bitmap as input and compute feature vectors
- M_report_recog: Take feature vectors, lookup DB of authorized faces, unlock door if face present; Report door state

FlowFence – Refactored App



Tc → Door.Open

Ts → Door.Open

Ts → Internet

Taint Labels and Flow Policies

H_1	{T1, T2, …}	F(D1)

App_ID	Label_Name
com.camera.publisher	Taint_Camera

Example Policy

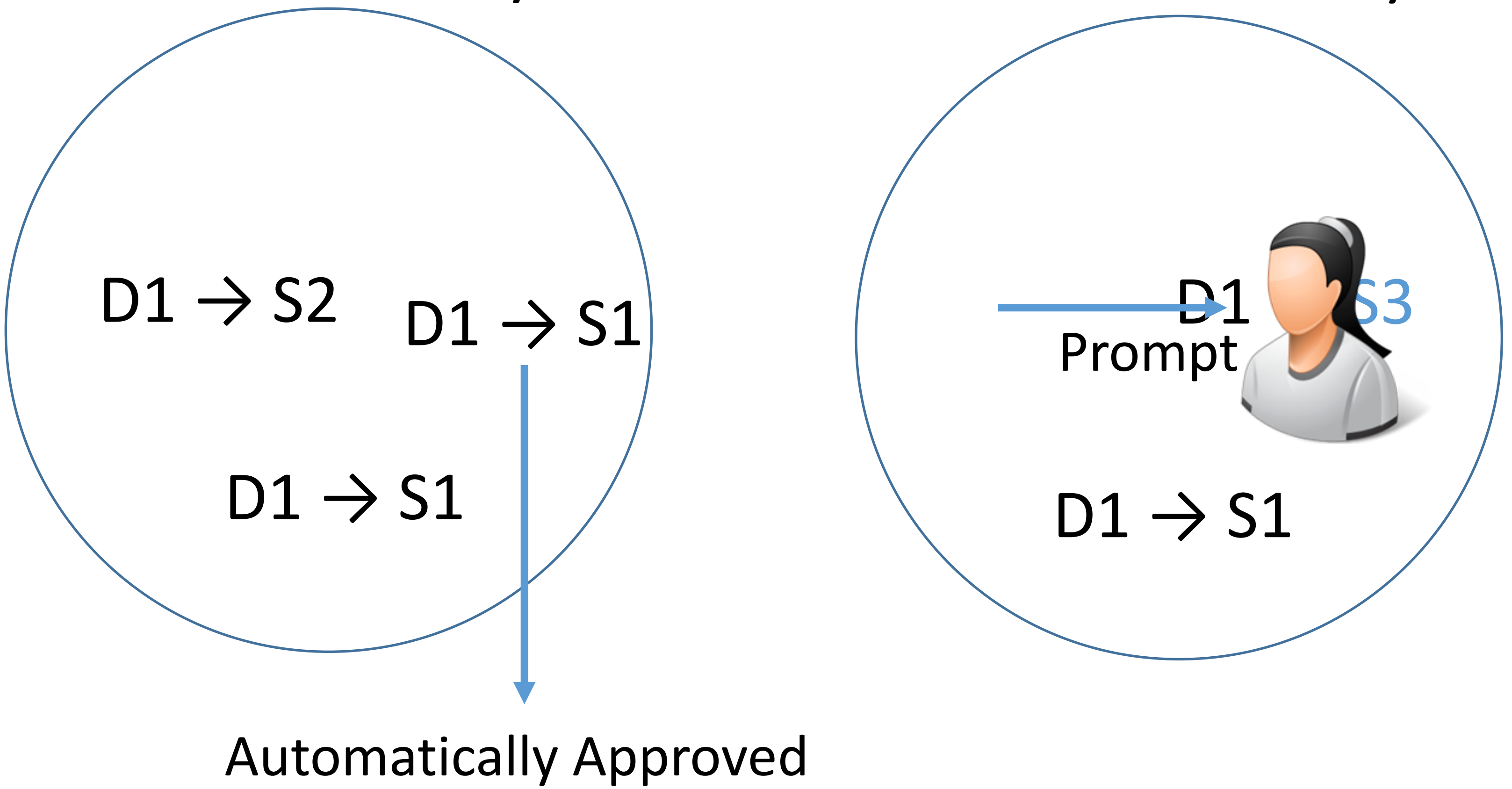
{
Taint_Camera → UI,
Taint_HeartR → Internet
}

- App_ID – unique application identifier on the underlying OS
- Label_Name – well-known string that identifies the type of data

Publisher and Consumer Flow Policies

Publisher Policy

Consumer Policy



{ Publisher;
  Taint_Camera → UI
}

{ Consumer;
  Taint_Camera → Door.Open
  Taint_DoorState → Door.Open
  Taint_DoorState → Internet
}

Data Sharing Mechanisms in Current IoT Frameworks

Smart home API

[Smart Homes]

Google Fit API

[Wearables]

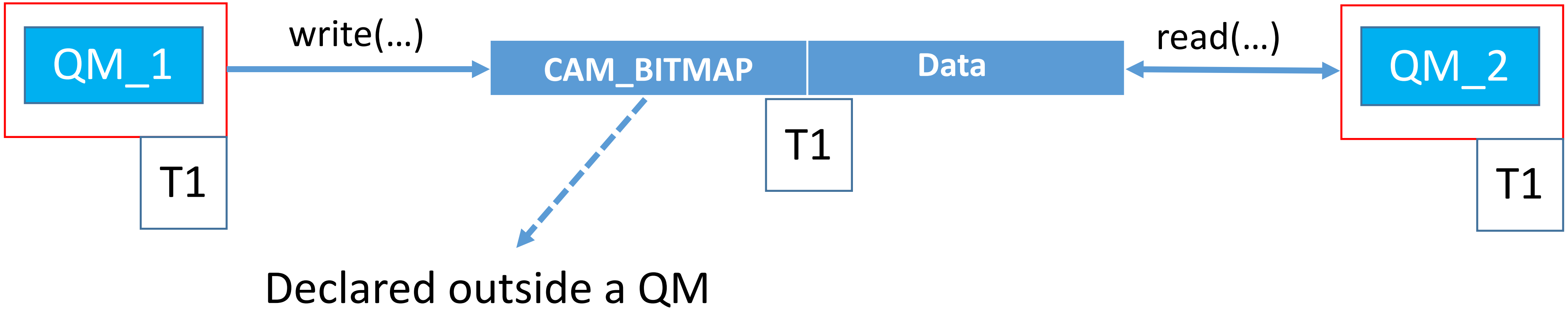
Android Sensor API

[Quantified Self]

- Polling Interface
  - App checks for new data
- Callback Interface
  - App is called when new data available
- Device Independence
  - E.g., many types of heart rate sensors produce “heart beat” data
  - Consumers should only need to specify “what” data they want, without specifying “how”

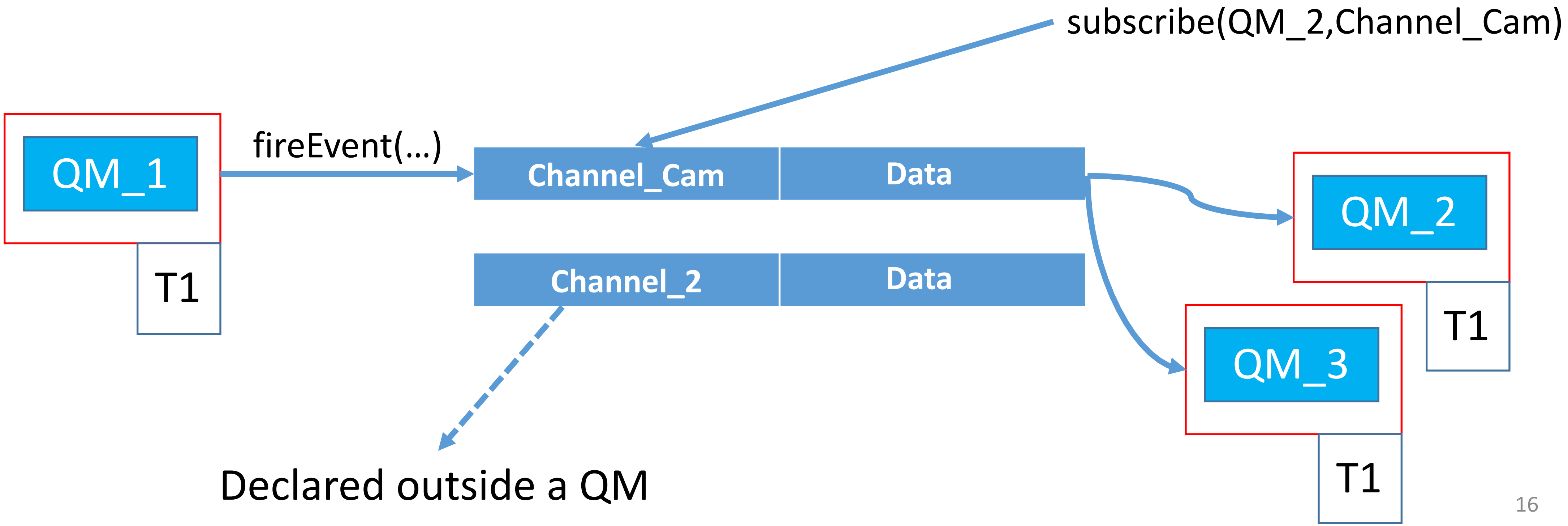
Key-Value Store – Polling Interface/Device Independence

- Each app gets a <u>single</u> Key-Value Store
- An app can <u>only write to its own</u> Key-Value Store
- Apps can read from any Key-Value Store
- Keys are <u>public information</u> because consumers need to know about them



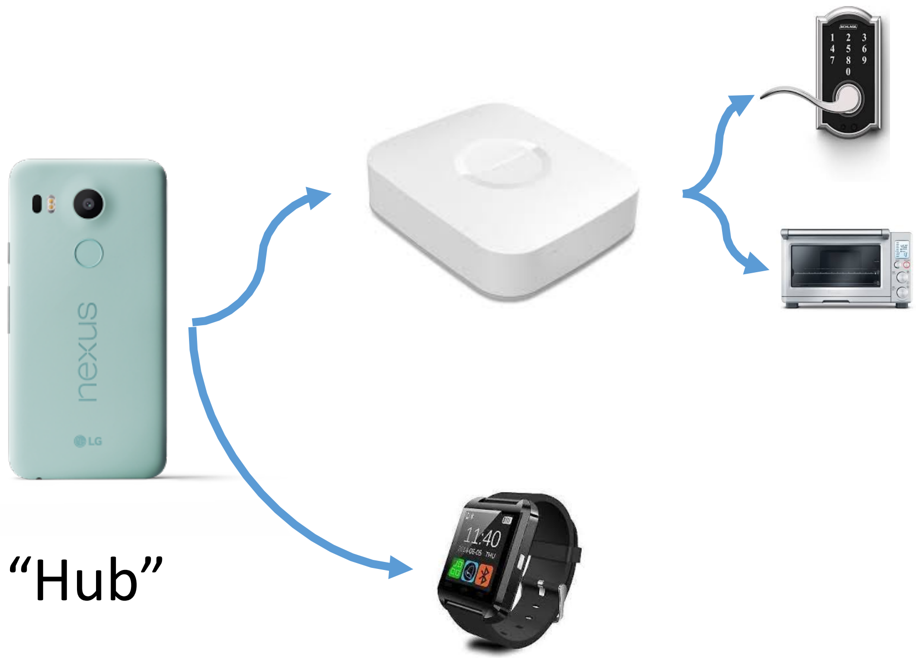
Event Channels – Callback Interface/Device Independence

- Apps can declare statically in code, their intended channels
- Only the owner of a channel can fire an event
- Channel name is public information

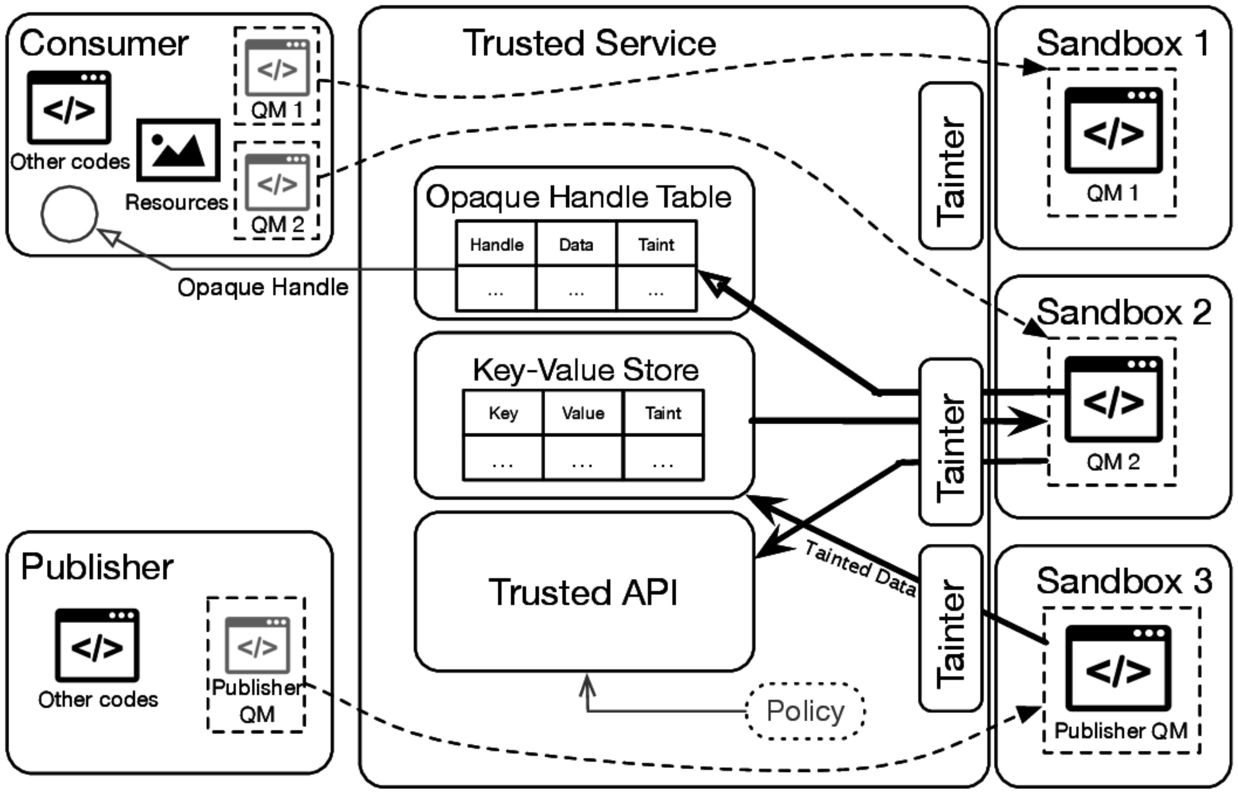


FlowFence Implementation

- IoT Architectures
  - Cloud
  - Hub
- isolatedProcess = true for sandboxes
- Supports native code



“Hub”



Evaluation Overview

- What is the overhead on a micro-level in terms of computation and memory?

Per-Sandbox Memory Overhead	2.7 MB
QM Call Latency	92 ms
Data Transfer b/w into Sandbox	31.5 MB/s

Comparable to IoT device ops over wide-area-network, e.g., Nest, SmartThings

Nest cam peak bandwidth is 1.2 Mb/s

- Can FlowFence support real IoT apps securely?

Ported 3 Existing IoT Apps: SmartLights, FaceDoor, HeartRateMonitor	Required adding less than 140 lines per app; FlowFence isolates flows

- What is the impact of FlowFence on macro-performance?

FaceDoor Recognition Latency	5% average increase
HeartRateMonitor Throughput	0.2 fps reduction on average
SmartLights end-to-end latency	+110 ms on average

Porting IoT Apps to FlowFence

App	Data Security Risk	Original LoC	FlowFence LoC	Flow Request
SmartLights	Can leak location information	118	193	Loc → Switch
FaceDoor	Can leak images of people	322	456	Cam → Lock, Doorstate → Lock, Doorstate → Net
HeartRateMon	Can leak images and heart rate	257	346	Cam → UI

SmartLights, FaceDoor – 2 days of porting effort each, HeartMon – 1 day of porting effort

Macro-performance of Ported Apps

FaceDoor Enroll Latency

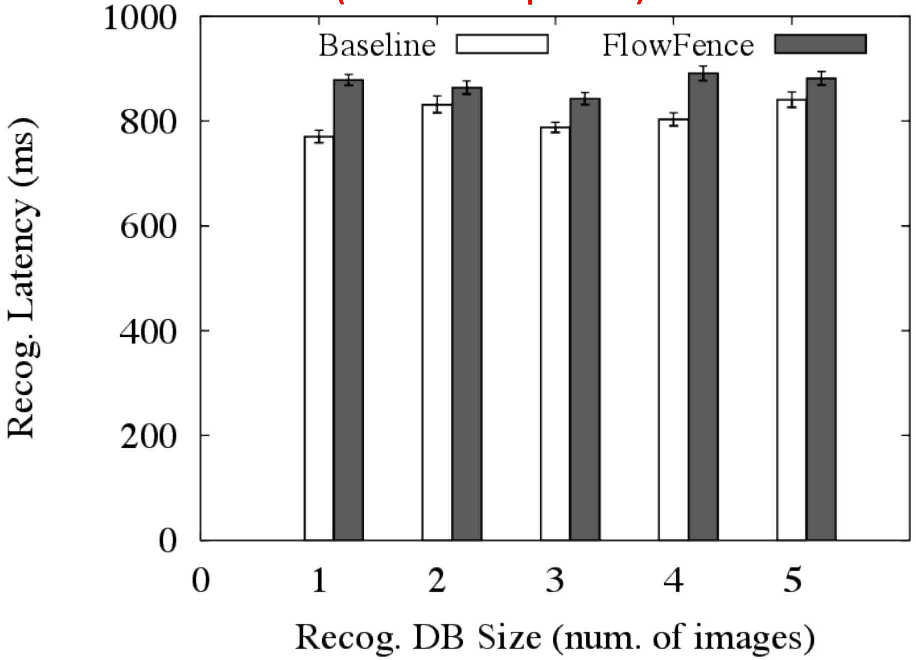
Baseline	811 ms (SD = 37.1)
FlowFence	937 ms (SD = 60.4)

SmartLights End-To-End Latency

Baseline	160 ms (SD = 69.9)
FlowFence	270 ms (SD = 96.1)

FaceDoor Recognition Latency

(612x816 pixels)



HeartRateMon Throughput

Throughput w/o Image Processing	23.0 (SD=0.7) fps	22.9 (SD=0.7) fps
Throughput w/ Image Processing	22.9 (SD=0.7) fps	22.7 (SD=0.7) fps

Summary

- **Emerging IoT App Frameworks only support permission-based access control**: Malicious apps can steal sensitive data easily
- **FlowFence explicitly embeds control and data flows within app structure**; Developers must split their apps into:
  - Set of communicating Quarantined Modules with the unit of communication being Opaque Handles – taint tracked, opaque refs to data
  - Non-sensitive code that orchestrates QM execution
- FlowFence supports publisher and consumer flow policies that enable building secure IoT apps
- We ported 3 existing IoT apps in 5 days; Each app required adding < 140 LoC
- Macro-performance tests on ported apps indicate FlowFence overhead is reasonable: e.g., 4.9% latency overhead to recog. a face & unlock a door

FlowFence: Practical Data Protection for Emerging IoT Application Frameworks

UNIVERSITY OF MICHIGAN

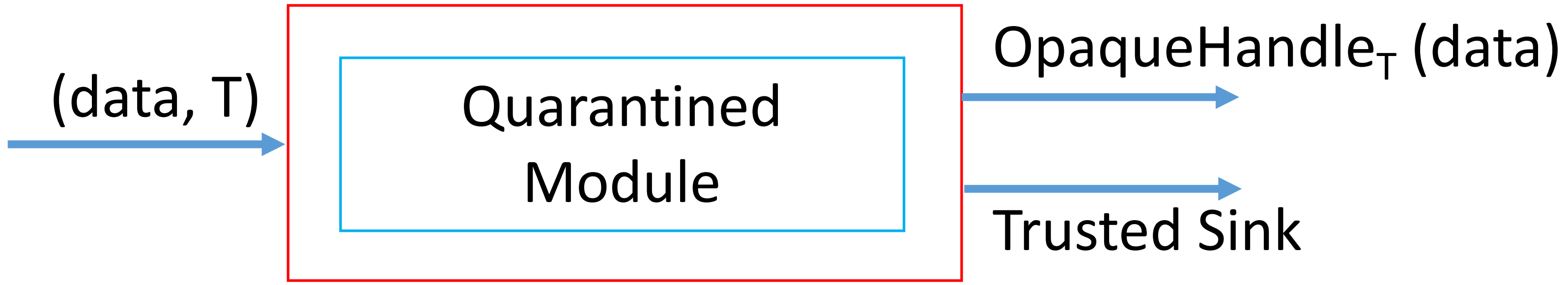
UNIVERSITÀ DEGLI STUDI DI PADOVA

- **Emerging IoT App Frameworks only support permission-based access control**: Malicious apps can steal sensitive data easily
- **FlowFence explicitly embeds control and data flows within app structure**; Developers must split their apps into:
  - Set of communicating Quarantined Modules with the unit of communication being Opaque Handles – taint tracked, opaque refs to data
  - Non-sensitive code that orchestrates QM execution
- FlowFence supports publisher and consumer flow policies that enable building secure IoT apps
- We ported 3 existing IoT apps in 5 days; Each app required adding < 140 LoC
- Macro-performance tests on ported apps indicate FlowFence overhead is reasonable: e.g., 4.9% latency overhead to recog. a face & unlock a door

https://iotsecurity.eecs.umich.edu

Earlence Fernandes

FlowFence Primitives – Quarantined Modules and Opaque Handles



- A developer-written Quarantined Module (QM) runs in a sandbox and computes on sensitive data

- Sandbox controls the ways in which data can enter and exit; FlowFence offers Key-Value Store and Event Channels for data sharing

- An Opaque Handle does not reveal information about:
  - Raw Data
  - Data Type
  - Taint Label
  - Data Size
  - Exceptions

  to non-QM code

Over-tainting

- Poor app decomposition
  - Developer should refactor app to more accurately reflect flows
  - FlowFence only taints QMs; not complete app code
- Poison Pill attacks due to malicious publisher
  - Publishers must define Taint Bound TMc whenever a KV store or event channel is created for that store or channel
  - Publishers cannot add taints beyond TMc
  - Consumers can check the taint bound, and then decide whether they want to interact with that publisher
  - TMc cannot be modified once set

Side Channels

- Best effort at closing some side channels (e.g., KV store keys and event channel names which are declared outside a QM at install time), but we do not handle all side channels

- For example, time to return handle can be modulated by sensitive data
  - Can make QMs return immediately and then execute async. w.r.t. caller
  - Similar to LIO [61]
  - Timing channels can also be bounded using predictive techniques [72]

Challenges in Applying Taint-based Flow Control

Time/Space Overhead
[Efficiency]

Implicit Flows
[Security]

Concurrency Attacks
[Security]

TaintDroid, DTA++, DyTAN, …

Amandroid, FlowDroid, …

Jif, JFlow, …

Diverse Producers/Consumers
[IoT-specific, Practicality, Security]

New Languages/No Rapid Dev. With Tools
[IoT-specific, Practicality]

Existing Problems while applying Dynamic & Static Flow Analysis

Time/Space Overhead [1]

```
if (sensitive_data == 0x1) {
        var = ‘A’
} else if (sensitive_date == 0x2) {
        var = ‘B’
}
```

Implicit Flows [3]

Specialized hardware for acceleration [2]

SmartThings

```
subscribe(dev, callback)
```

Miss flows due to multithreading/events [4]

[1] Paupore et al., HotOS’15
[2] Ruwase et al., SPAA’08
[3] Sarwar et al., SECRYPT’13
[4] Myers et al., POPL’99

IoT-specific Challenges in applying Dynamic/Static Information Flow Analysis

- **Asynchronous, multithreaded and event-based environment**

`subscribe(dev, callback)`

<u>Language based techniques may not apply directly</u>

- **Diverse Publishers and Consumers (data labels not known apriori)**

<u>We don’t know which devices are present in any given IoT configuration (and hence which types of data)</u>

- **OS and Language diversity**

<u>Some techniques take advantage of OS/Language structure</u>

Existing and IoT-specific problems with applying Dynamic/Static Instruction-level Flow Analyses

- Instruction-level Taint Tracking
  - Tainting app code or OS leads to computational and space overhead [1] – IoT devices/hubs are often constrained/low-powered without special hardware
  - Requires knowledge of taint labels beforehand – IoT has diverse device types; We do not know which taint labels are flowing through a program beforehand

- Static Analyses and Language techniques
  - Implicit Flows [2], IPCs, Asynchronous code (which is common in IoT apps i.e., Trigger-Action programming is ubiquitous [3]) can cause under-tainting
  - Developers must use specialized languages restricting flexibility

- Reliance on particular language or OS structure for security
  - IoT exhibits OS and language diversity

[1] Paupore et al., HotOS’15
[2] Sarwar et al., SECRYPT’13
[3] Ur et al., CHI’14, CHI’16