Internet of Things (IoT) enables the future

- Smart Homes
- Healthcare
- Smart Energy
- Smart Farms

Power consumption

<table>
<thead>
<tr>
<th></th>
<th>Usage/month</th>
</tr>
</thead>
<tbody>
<tr>
<td>No with smart</td>
<td></td>
</tr>
<tr>
<td>With smart</td>
<td></td>
</tr>
</tbody>
</table>

30% saving
IoT is not magic
IoT enables the future (and a whole lot of problems)

IoT Devices Are Hacking Your Data & Stealing Your Privacy - Infographic

When you live home ... Whether the door is locked or not...
Sensitive data in IoT apps

**Problem:** Users lack visibility into who sees their sensitive information

- Look inside of IoT apps to determine how they use privacy sensitive data
  - Device states
  - Device information
  - User inputs
  - Location (physical and geo-location)
Static taint analysis

**Goal:** Analyze app source code to determine when privacy sensitive information leaves the IoT app

- Static taint analysis is a technique that tracks information dependencies from an origin
- Conceptual idea:
  - Taint source
  - Taint propagation
  - Taint sink

```python
c = taint_source()
...
a = b + c
...
network_send(a)
```
Challenges

- IoT programming platforms are diverse
- Identifying sensitive sources in IoT apps is quite subtle
- Each IoT platform has its idiosyncrasies that require special treatment

• **Current data tracking tools are insufficient to address these challenges**
Saint

- Saint is integration of static taint tracking into the IoT apps

![Diagram of Saint's workflow]

1. Saint’s taint sources/sinks
2. IoT App source code
3. Saint analyzer
4. Report discovered data flows

Saint's analyzer

- Obtain IR (source, sink and entry point detection)
- Perform data flow analysis
- Report discovered data flows

**Analysis Output**

- **Taint sink**: Internet --- httpPUT() in Line 123
- **Dataflow path 1**: sendSms --> $DeviceName [Device Information]
- **Finding #**: Device Information transmitted [Developer-defined URL]

**Stacktrace**

```python
def initialize()
    ecobee.poll()
    subscribe(app, appTouch)
```

**Analysis Console**

- http://saint-project.appspot.com/
From app source code to IR

Devices

```groovy
input (p, presenceSensor, type:device)
input (s, switch, type:device)
input (d, door, type:device)
input (toTime, time, type:user_defined)
input (fromTime, time, type:user_defined)
input (c, contact, type:user_defined)
```

Events

```groovy
subscribe(p, "present", h1)
```

Computation

```groovy
h1(){
  s.on()
  d.unlock()
  def between = y()
  if (between){
    z()
  }
}
y(){
  return timeOfDayIsBetween(fromTime, toTime)
}
z(){
  sendSms(c, "...")
}
```
Backward taint tracking

- Identify sensitive data flow paths

```python
def foo():
    temp = ther.latestValue("temperature")
    tempCel = convert(temp) + thld
    bar(tempCel)

def convert(t):
    return ((t - 32) * 5) / 9

def bar(t):
    ther.setHeatingSetpoint(t)
    sendSMS(adversary, "set to ${t}")
```

1: input (ther, thermostat, device)
2: input (thld, number, user_defined)
3: def initialize():
   4:     subscribe(app, appHandler)
5: }
6: def appHandler(evt):
   7:     foo()
8: }
13: def foo():
    14:     temp = ther.latestValue("temperature")
    15:     tempCel = convert(temp) + thld
    16:     bar(tempCel)
17: }
18: def convert(t):
    19:     return ((t - 32) * 5) / 9
20: }
21: def bar(t):
    22:     ther.setHeatingSetpoint(t)
    23:     sendSMS(adversary, "set to ${t}")
24: }
```

Dependence relation:
- (15: temp, 16: [temp, thld])
- (16: tempCel, 15: [temp, thld])
- (23: t, 16: [tempCel])
- (23: t)
Analysis Sensitivity and Implicit Flows

- **Path-sensitivity**
  - Collects the evaluation results of the predicates
  - Discards infeasible paths

- **Context-sensitivity**
  - Implements depth-one call-site sensitivity
  - Discards paths not matching calls and returns

- **Implicit flows**
  - Determines whether predicates at conditional branches depends on a tainted value
  - Taints all elements in the conditional branch
Algorithms for IoT-specific idiosyncrasies

- On-demand algorithms for analysis precision
  - State variables
    - Field-sensitive analysis
  - Web service apps
    - Allows external entities to access devices
  - Call by reflection
    - Add all methods as possible call targets

```python
counter = state.switchCounter  # state variable
if (counter):
    device actions

mappings {  # web-service apps
    path("/switches") {
        action: [GET: "listSwitches"]
    }
}
def listSwitches() {
    return it.currentValue("switch")
}

"$methodName"()  # call by reflection
def foo() {  # add as possible call target }
def bar() {  # add as possible call target }
```
Application Study

- Implemented Saint for SmartThings IoT platform
- Selected 168 official and 62 third-party market apps
- 92 official and 46 third-party apps expose at least one kind of sensitive data

<table>
<thead>
<tr>
<th>Apps</th>
<th>Internet</th>
<th>SMS</th>
<th>Both</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Official</td>
<td>24</td>
<td>63</td>
<td>5</td>
<td>92</td>
</tr>
<tr>
<td>Third-party</td>
<td>10</td>
<td>36</td>
<td>-</td>
<td>46</td>
</tr>
</tbody>
</table>
What type of privacy-sensitive information leaves IoT apps?

<table>
<thead>
<tr>
<th>% Apps</th>
<th>Official apps</th>
<th>Third-party apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device state</td>
<td>60</td>
<td>50</td>
</tr>
<tr>
<td>Device info.</td>
<td>70</td>
<td>40</td>
</tr>
<tr>
<td>User input</td>
<td>80</td>
<td>30</td>
</tr>
<tr>
<td>Location</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>State variable</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>
### Who sees privacy-sensitive information?

**sendSMS(phoneNumber, “kids $presence”)**

**httpPost(URL, “kids $presence”)**

<table>
<thead>
<tr>
<th>Taint Sinks</th>
<th>Apps</th>
<th>Recipient defined by</th>
<th>Content defined by</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Messaging</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Official</td>
<td>User</td>
<td>154</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Developer</td>
<td>0</td>
<td>149</td>
</tr>
<tr>
<td></td>
<td>External</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Third-party</td>
<td>User</td>
<td>67</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Developer</td>
<td>0</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>External</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Internet</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Official</td>
<td>User</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Developer</td>
<td>48</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>External</td>
<td>44</td>
<td>40</td>
</tr>
<tr>
<td>Third-party</td>
<td>User</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Developer</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>External</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
Summary

- Introduced Saint, a static analysis tool that identifies sensitive data flows in IoT apps
- Evaluated Saint on 230 SmartThings apps
- Found %60 of the analyzed apps includes sensitive data flows
- Consumers and developers can use Saint to identify potential privacy risks
- Saint console is available: http://saint-project.appspot.com/
IoTBench

27 data leaks

28 security/safety violations

15 attacks migrated from mobile phone security

500+ official and third party apps

V.1.0.1 Released May 2018

https://github.com/IoTBench/

IoTBench-test-suite
A micro-benchmark suite to assess the effectiveness of tools designed for IoT apps

iot-platform smartthings openhab malicious-behaviors data-leaks

Groovy ★ 10 🆙 2 Updated on May 12
Thank you for listening!