Outline

- Research paper:
  - Security Analysis of Emerging Smart Home Applications
  - SmartAuth: User-Centered Authorization for the Internet of Things
  - Sensitive Information Tracking in Commodity IoT
Motivation

Emerging Smart Home Frameworks
Motivation

Potential Security Risks

Remote determination of prime time for Burglary [1,2]

OR

Flooding [1]

Current Vulnerabilities

Remote determination of prime time for Burglary [1,2]

These attacks are device-specific, and require proximity to the home

[2] FTC Internet of Things Report’15
Motivation

- In what ways are these emerging, programmable smart homes vulnerable to attacks?
- What do those attacks entail?
Motivation

- Why SmartThings?
  - relatively mature
  - 521 smartApps
  - 132 device types
  - share design principles with other existing nascent frameworks

- Methodology
  - examine security from 5 perspectives by constructing test apps to exercise SmartThings API
  - empirical analysis of 499 apps to determine security issue prevalence
  - proof of concept attacks that compose security flaws
## Results Overview

<table>
<thead>
<tr>
<th>Security Analysis Area</th>
<th>Finding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overprivilege in Apps</td>
<td>Two Types of <strong>Automatic Overprivilege</strong></td>
</tr>
<tr>
<td>Event System Security</td>
<td>Event <strong>Snooping and Spoofing</strong></td>
</tr>
<tr>
<td>Third-party Integration Safety</td>
<td>Incorrect OAuth Can Lead to Attacks</td>
</tr>
<tr>
<td>External Input Sanitization</td>
<td>Groovy <strong>Command Injection</strong> Attacks</td>
</tr>
<tr>
<td>API Access Control</td>
<td>No Access Control around SMS/Internet API</td>
</tr>
</tbody>
</table>

| Empirical Analysis of 499 Apps         | > 40% of apps exhibit overprivilege of atleast one type |

| Proof of Concept Attacks                | Pincode Injection and Snooping, Disabling Vacation Mode, Fake Fire Alarms |
SmartThings Primer
Capability System

Untrusted SmartApp

Send commands
Read/set attributes
Receive events

ZWave Lock SmartDevice

capability.lock
capability.lockCodes
capability.battery
...

<table>
<thead>
<tr>
<th>Capability</th>
<th>Commands</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>capability.lock</td>
<td>lock(), unlock()</td>
<td>lock (lock status)</td>
</tr>
<tr>
<td>capability.battery</td>
<td>N/A</td>
<td>battery (battery status)</td>
</tr>
</tbody>
</table>

Usability
Simpler Coarser Capabilities

Ease of Development
Expressive Functionality

Security
Very Granular Capabilities
Overprivilege in SmartApps

SmartThings Cloud Platform

- Groovy-Based Sandbox
  - SmartDevice
  - Capability System
    - [Cmd/Attr]
    - [Events]
  - SmartApp

Relationships:
- WiFi
- ZWave
- Internet API
- SMS API
- HTTPS
- GET/PUT

Configure
Control

SmartThings Companion App
Overprivilege in SmartApps

- Coarse-grained Capabilities
  - ‘auto-lock’ app from app store
  - only needs ‘lock’ command, but can also issue ‘unlock’

Overprivilege increases attack surface of the Home
Insufficient Event Data Protection
Once a SmartApp gains any capability for a device, it can subscribe to any event that device generates.

If a SmartApp acquires the 128-bit ID, then it can monitor all events of that device without gaining any of the capabilities the device supports.

Using the 128-bit ID, a SmartApp can spoof physical device events.
Other Potential Security Issues - OAuth

- Insecurity of Third-Party Integration:
  - SmartApps expose HTTP endpoints protected by OAuth;
- Incorrect implementation can lead to remote attacks [1]

[1] Chen et al., OAuth Demystified for Mobile Application Developers, CCS'14
Unsafe use of Groovy Dynamic Method Invocation:

- Apps can be tricked into performing **unintended actions**

```groovy
def foo() { ... }
def str = "foo"
"$str"()
```
Other Potential Security Issues – Unrestricted External Communication APIs

- Unrestricted Communication Abilities:
  - SMS and Internet
  - Can be used to *leak data arbitrarily*
Computing Overprivilege

Coarse-Grained Capabilities

Used
Cmds/Attrs

Coarse SmartApp-SmartDevice Binding

Used
Capabilities
Measuring Overprivilege in SmartApps

**Challenges**
- Incomplete capability details (commands/attributes)
- SmartThings is closed source; can’t do instrumentation
- Groovy is extremely dynamic; Bytecode uses reflection (Groovy Meta Object Protocol)

**Solutions**
- Discovered an unpublished REST endpoint, which, if given a device ID, returns capability details
- Study source code of apps from open-source app store instead
- Static analysis on AST
# Empirical Analysis Results

<table>
<thead>
<tr>
<th></th>
<th>Documented</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>65</td>
<td>93</td>
</tr>
<tr>
<td>Attributes</td>
<td>60</td>
<td>85</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reason for Overprivilege</th>
<th>Number of Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse-grained Capability</td>
<td>276 (55%)</td>
</tr>
<tr>
<td>Coarse SmartApp-SmartDevice Binding</td>
<td>213 (43%)</td>
</tr>
</tbody>
</table>

| Overprivilege Usage Prevalence (Coarse Binding) | 68 (14%) |
Exploiting Design Flaws in SmartThings

- Command Injection
- OAuth Compromise
- Overprivilege
- Unrestricted SMS API
- Event Spoofing

- Pincode Injection: Popular Existing SmartApp with Android companion app; *Unintended action of setCode() on lock*
- Pincode Snooping: Stealthy malware SmartApp; *ONLY requests capability.battery*
- Disabling Vacation Mode: Malware SmartApps with *no capabilities*; Misuses logic of existing SmartApps with fake events
- Fake CO Alarm

21
Potential Defense Strategies

- Achieving least-privilege in SmartApps
  - Risk asymmetry in device operations, e.g., oven.on and oven.off
  - Include notions of risk from multiple stakeholders, rank [1], and regroup

- Preventing information leakage from events
  - Provide a notion of strong identity for apps + access control on events
  - Make apps request access to certain types of events, e.g., lock pincode ACKs
SmartAuth: User-Centered Authorization for the Internet of Things

Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, XianZheng Guo and Patrick Tague

USENIX SEC 2017
Smart-home apps improve quality of life, but can be risky
Users have limited information about what is going on.
Users have limited information about what is going on
Can we notify users about the most important information?

This app doesn’t need to control the lock!
Challenges

- Security and privacy implications depend on context
  - Same sensor in bedroom vs. outside has very different implications

- Behaviors in code cannot be mapped directly to high-level functionality in description

- Need to support cross-device scenarios
Redesign the authorization system

● Goals:
  ○ Security and Privacy:
    ■ Share minimum data and capabilities for desired functionality
  ○ IoT specific:
    ■ Cross-device, context-based, automatic control
  ○ Usability:
    ■ Assist user to make well-informed decisions, minimize user burdens
  ○ Performance:
    ■ Lightweight and compatible
SmartAuth Overview

- Program Analyzer
- Content Inspector (NLP, e.g., [2])
- Policy Enforcer
- Consistency Checker
- Authorization creator

Example - Program Analyzer

section("Bathroom humidity sensor") {
  input "bathroom", "capability.relativeHumidityMeasurement",
  title: "Which humidity sensor?"
}

if (shower.value.toInteger() > relHum) {
  coffee.on()
Example - NLP and Behavior Correlation

Description analysis

Entity:
- Coffee machine
- Shower

Condition:
- Taking a shower

Triggers:

Action:
- Turn on the coffee machine

Program analysis

Entity:
- Switch
- Humidity sensor
- Lock

Context clue:
- Bathroom for the humidity sensor
- Coffee for the switch

Condition:
- Humidity reading > threshold

Triggers:

Action:
- Turn on the switch
- Unlock the door
Evaluation

- How effective is SmartAuth?
  - How accurate is the policy extraction?
  - How does SmartAuth impact users’ decisions?

- What is the performance overhead?

- How compatible is SmartAuth?
Evaluation: Effectiveness of extracting policies

- Manual analysis to verify all the cases
- 3.9% false positives
  - Limitations of NLP analysis
- No false negatives
Evaluation: User study

- Between-subjects, in-lab study

- 100 participants split into two groups:
  - SmartAuth
  - Current SmartThings interface (manifest-style)

- Five pairs of similar apps
  - Participant chooses one of the two
  - One has unexpected privileges
Example app pairs

**Lights Off**
- Turn lights off when no motion or presence detected for a period of time

  - Light switches to turn off
    - Choose light switches
      - Tap to set
    - Turn off when there is no motion and presence
      - Choose motion sensor
        - Tap to set
      - Choose presence sensors
        - Tap to set

**Darken Behind Me**
- Turn your lights off after a period of no motion being observed

  - When there’s no movement...
    - Where?
      - Tap to set
    - Monitor the temperature...
      - Which?
        - Tap to set
    - Turn off a light...
      - Which?
        - Tap to set
SmartThings VS SmartAuth
Users make better decisions with SmartAuth
Evaluation: Performance

- Run-time enforcement
Evaluation: Compatibility

- Observe behaviors and debug information
- None of the apps crash
- In the extreme case, 3.3% of apps lose functionality when we block all remote access
Sensitive Information Tracking in Commodity IoT


USENIX SEC 2018
Internet of Things (IoT) enables the future

Smart Homes

Smart Energy

30% saving

No with smart
With smart
Usage/month

Healthcare

Smart Farms
IoT is no magic
IoT enables the future (and a whole lot of problems)

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

Alexa beware! New smart home tests reveal serious privacy flaws

"Issues such as the fear of oversharing of data by commercial services, insufficient protection of stored personal data, and the possibility of interception of digital traffic by cybercriminals are significant."

When you live home ...

Whether the door is locked or not...
**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

Saint: Tracking Sensitive Data in IoT Apps
Saint: Tracking Sensitive Data in IoT Apps

- **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
Challenges

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

- Saint is integration of static taint tracking into the IoT apps.
From app source code to IR

```
Devices
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
subscribe(p, “present”, h1)

Computation
h1()
  s.on()
  d.unlock()
  if (between) {
    z()
  }
} y() {
  return timeOfDayIsBetween(fromTime, toTime)
} z() {
  sendSms(c, “…”)
}
```
Backward taint tracking: Identify Sensitive Data Flow Paths
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?
Who sees privacy-sensitive information?

Recipient

\[ \text{sendSMS}(\text{phoneNumber, “kids $presence”}) \]

\[ \text{HttpPost}(\text{URL, “kids $presence”}) \]

<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></td>
<td></td>
<td>User</td>
<td>Developer</td>
</tr>
<tr>
<td>Messaging</td>
<td>Official</td>
<td>154</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Third-party</td>
<td>67</td>
<td>0</td>
</tr>
<tr>
<td>Internet</td>
<td>Official</td>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Third-party</td>
<td>0</td>
<td>13</td>
</tr>
</tbody>
</table>
Thank you!

Questions?