Towards Secure and Reliable IoT Applications

Gang Tan, CSE, Penn State
Nov 15th, 2019
@ 2nd IoT Security and Privacy Workshop
Internet of Things (IoT) enables the future

Smart Homes
Source: Samsung

Smart Energy
Source: LG

Healthcare
Source: John Hopkins

Smart Farms
Source: Microsoft

IoT is not magic

Connected devices

Mobile app

IoT application

Automation
IoT enables the future (and a whole lot of problems)

Many of these failures are traditional security problems:

**Software bugs, user errors, poor configuration, or faulty design**

Their code is an automaker’s nightmare: software that lets hackers send commands through the Jeep’s entertainment system to its dashboard functions, steering, brakes, and transmission, all from a laptop that may be across the country.

While wandering in his neighborhood, he noticed a lot of Bluetooth locks popping up and decided to do some sniffing of those "security" gadgets (read: capturing packets being sent between devices). "I discovered plain-text passwords being sent that anybody could read.

Mirai botnet adds three new attacks to target IoT devices
This new version of the botnet uses exploits instead of brute force attacks to gain control of unpatched devices.
IoT app interaction (via the physical world)

- fire -> sprinkler activation -> water leak detection -> shut off water valve -> no water for sprinkler!
- Problem: the interaction between IoT apps cause unreliability and insecurity.
IoT app interaction: another example

Temp-control app: open the window, when temp > 80 °F

Heater-control app: time at 6pm -> turn the heater on -> temperature rise

* Example by Ding & Hu [CCS 2018]
In this talk...

How to **increase security and reliability** of IoT Apps and their interaction?

### IoT Safety and Security

|---|

### IoT Privacy


### IoT Fault Tolerance


### Surveys:

|---|
Collaborators

- Z. Berkay Celik (Penn State -> Purdue)
- Patrick McDaniel (Penn State)
- Michael Norris (Penn State)

- Other collaborators
  - Penn State: Prasanna Venkatesh, Shulin Zhao, and Anand Sivasubramaniam
  - Florida International University: Leo Babun, Amit Sikder, Hidayet Aksu, and Selcuk Uluagac
Agenda

Soteria*

Automated IoT Safety and Security Analysis

[USENIX ATC 2018]

* Greek goddess protecting from harm

IoTRepair

Systematically Addressing Device Faults in Commodity IoT

[On-going Work]

Projecting into the Future
Automated IoT Safety and Security Analysis

[Soteria*]

* Greek goddess protecting from harm
Soteria

Problem: IoT platforms cannot evaluate whether an IoT app or a collection of apps is safe, secure, and operates correctly

- Soteria performs **model checking** on IoT apps to see whether they conform to a set of safety/security properties
From IoT apps to finite-state machines

Model checking: Does the sprinkler system activate when there is a fire?
Soteria components

1. IoT app source code
2. Obtain IR
3. Property identification
4. Temporal logic
   - individual
   - union
   - State-model extraction

Property verification (Model checker)

Pass
Fail
State-model extraction from source code

- What is a state model?
  - States and transitions
  - In IoT applications...
    - States: Device attributes
    - Transitions: Labeled by events that trigger the attribute changes

- Challenges of extracting state models
  - IoT programming platforms are diverse
  - Transitions may be guarded by conditions
  - State-explosion problem
Coping with diverse IoT platform languages

- App source -> IR -> state model
- We can reuse the part from IR to state-model extraction, for a new source language
An example toy app

- The app: when users are back at home, turn on the light, unlock the door, and send a notification email
  - Between fromTime and toTime
The IR of the example app

Devices

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

Event subscription

- `subscribe(p, "present", handler)`

Computation

```groovy
handler(){
    def between = inBetween()
    if (between){
        s.on()
        d.unlock()
        notify()
    }
}

inBetween(){
    return timeOfDayIsBetween(fromTime, toTime)
}

notify(){
    sendSms(c, "...")
}
```

* Extracted from Groovy code for Samsung’s SmartThings
def inBetween()
    return timeOfDayIsBetween(fromTime, toTime)

def notify()
    sendSms(c, "...")
Conditional device attribute changes

- Perform path exploration and accumulate path conditions
  - Add a transition using end states and path conditions

```java
subscribe(presence, present, handler)

// Entry point
handler(){
    above = 50
    below = 5
    power = get_power()

    if(power > above){
        switch.off()
    }
    if(power < below){
        switch.on()
    }
}

get_power(){
    latest_pow = power_meter.currentValue("power")
    return latest_pow
}
```

Source code of Energy-control IoT app

Without path exploration:
- Present: S0
- Power: power > 50
- Action: switch-off

With path exploration:
- Present: S0
- Power: power < 5
- Action: switch-off
- Power: power < 5
- Action: switch-on
Coping with state explosion

- Reduce states by aggregating numerical-valued attributes

```
1: def modeChangeHandler(evt){
2:   def temp = 68
3:   setTemp(temp)
4: }
5: def setTemp(t){
6:   ther.setHeatingPoint(t)
7: }
```

Thermostat-control IoT app

Without state reduction
Thermostat temperature

- t=50
- t=51
- ...
- t=95

With state reduction
Thermostat temperature

- t=68
- t<>68

Worklist

- (2: temp = 68)
- (6: t, 3: temp)
- (6: t)
Microbenchmarks

Setup: Intel i5 Core 2 Duo, Java Runtime 1.8, NuSMV 2.6, Graphviz 2.36

- **State-reduction efficacy**
  - 10 numerical-valued devices in 14 apps

- **State model extraction overhead**
  - An app with 180 states, on avg. ~ 17 secs
IoT safety/security property identification

- **Property** is a system artifact formally expressed via **temporal logic** and validated on the state model
  - Relied on use/misuse case requirements engineering for discovering IoT Properties

- **5 general properties**
  - Device-independent

- **30 application-specific properties**
  - Identifies use cases of one or more devices

1. The door must always be locked when the user is not home
2. The refrigerator and security system must always be on
3. The water valve must be closed if a leak is detected
   ... 
4. The alarm must always go off when there is smoke
   ... 

- **5 general properties**
  - Conflicting state changes
  - Race condition of events
Property validation

- Individual applications
  - Properties verified at validated through a model checker

- Multiple applications
  - Create a union state model* of interacting apps

* Union state model represents the complete behavior when the multiple apps running together

![Diagram showing states and transitions for App1, App2, and App3.]

\[ \text{AG } \neg (\text{smoke-detected} \land \text{door-locked}) \]

- Is door always unlocked when there is smoke at home?

Violation

Initial States

Safe

Soteria
Soteria in action...

Soteria – a system for formal verification of IoT apps through model checking

Source code

```python
section("Turn on a pump...") {
    input "valve_device", "capability.valve", title: "Which?",
    required: true }

def installed() {
    subscribe(valve_device, "water.wet",
    waterWetHandler)
}
```

IR

```plaintext
// Permissions block
input (water_sensor, waterSensor, type:device)
input (valve_device, valve, type:device)
```

State-model

```
[water.dry, valve.close]
[water.wet, valve.close]
water.dry
[water.dry, valve.open]
water.wet
water.wet
[water.wet, valve.open]
[water.wet, valve.close]
```  

Model Checking

Property

```
water.wet ⇒ (AX valve.on)
```

SMV format of the state-model

```

Output

Stacktrace

Using NuSMV symbolic model checker...
General properties failed at state-model construction: none
NuSMV >> read model ...
NuSMV >> check property
NuSMV >> true
```
Follow-up Work: IoTGuard [NDSS 19]

Soteria

- Static approaches overapproximate and report false positives

Limitations:

- **IoTGuard**: A dynamic **property enforcement** system on IoT device behaviors
  - Use the same properties as Soteria
  - But enforce the properties at runtime
  - In addition, extend the scope to cover trigger-action platform apps

- Trigger-action platform apps
  - connect digital services with IoT apps
  - 300+ online services (IoT and non-IoT)

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>E (Smart home): user-present</td>
</tr>
<tr>
<td>A (Google): log user’s presence to a google doc file</td>
</tr>
</tbody>
</table>
Evaluation

- Implemented Soteria and IoTGuard for Samsung’s SmartThings platform
- Selected 65 SmartThings market apps: 35 official and 30 third-party apps
- For IoTGuard, further selected about 30 IFTTT trigger-action apps
  - Did evaluation on a simulated smart home including 29 devices (20 device types)
• **14%** individual apps violate some properties (10 properties in total)
• All found violations were in third-party apps

<table>
<thead>
<tr>
<th>App ID</th>
<th>Violation Description</th>
<th>Violated Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP1</td>
<td>The music player is turned on when user is not at home</td>
<td>P.13</td>
</tr>
<tr>
<td>TP2</td>
<td>The door is unlocked on sunrise and locked on sunset</td>
<td>P.1</td>
</tr>
<tr>
<td>TP3</td>
<td>The location is changed to the different modes when the switch is turned off and when the motion is inactive</td>
<td>S.4</td>
</tr>
<tr>
<td>TP4</td>
<td>The flood sensor sounds alarm when there is no water</td>
<td>P.29</td>
</tr>
<tr>
<td>TP5</td>
<td>The music player turns on when the user is sleeping</td>
<td>P.28</td>
</tr>
<tr>
<td>TP6</td>
<td>The lights turn on and turn off when nobody is at home</td>
<td>P.13, S.1</td>
</tr>
<tr>
<td>TP7</td>
<td>The lights turn on and turn off when the icon of the app is tapped</td>
<td>S.1</td>
</tr>
<tr>
<td>TP8</td>
<td>The switch turns on and blinks lights when no user is present</td>
<td>P.12</td>
</tr>
<tr>
<td>TP9</td>
<td>The door is locked multiple times after it is closed</td>
<td>S.2</td>
</tr>
</tbody>
</table>

TP = Third-party

P = App-specific properties
S = General properties
Soteria Findings - Multi-app analysis

- Several groups of apps had property violations

<table>
<thead>
<tr>
<th>Gr. ID</th>
<th>App ID</th>
<th>Event/Actions</th>
<th>Violated Pr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>O7, TP3</td>
<td>switch off, change location mode, motion inactive, switch-on</td>
<td>P.12, P.13, P.14, P.17, S.1, S.2</td>
</tr>
<tr>
<td></td>
<td>O30, TP21</td>
<td>location mode change, switch-off</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O31, TP22</td>
<td>location mode change, switch-on</td>
<td></td>
</tr>
<tr>
<td></td>
<td>O12, TP19</td>
<td>location mode change, set thermostat heating, set thermostat cooling</td>
<td></td>
</tr>
</tbody>
</table>

Switch (O7/TP3) change mode (vacation, sleeping)

- mode change (O31/TP22) switch on
- mode change (O30/TP21) switch off
- mode change (O12/TP19) set thermostat heating
- mode change (O12/TP19) set thermostat cooling
V.1.1 Released in Sept 2018

IoTBench

https://github.com/IoTBench/

27 data leaks

28 security/safety violations

500+ official and third party Smartthings and OpenHAB apps
IoTRepair

Systematically Addressing Device Faults in Commodity IoT

[Ongoing Work]
Motivation

- IoT devices are prone to faults

Sprinkler stops working because of loss of power or software bug

Flood sensor always reports flood because of device error (stuck-at errors)

Faults lead to unreliable/insecure physical environments
Fault types

• Fail-stop faults
  ▶ When devices stop responding to remote commands
  ▶ E.g., power loss, communication loss, software/hardware errors that stop devices

• Non fail-stop faults
  ▶ When devices continue to operate, but function incorrectly
  ▶ E.g., stuck-at faults, outlier faults, spike faults, high-variance faults

• Cascading faults
  ▶ When a faulty device managed by one app triggers an event in another app
A survey of IoT platforms on how faults are handled

- Undetected: the platform does not detect this type of fault
- Silent: the platform detects faults but does not notify applications
- Generic Error: the platform gives a generic error to applications accessing a faulty device
- Detailed Error: the platform specifies information about the fault type in the error

**IoT platforms do not provide developers with sufficient mechanisms to handle faults:** none provides info about non-fail-stop faults; only AndroidThings provides info about fail-stop faults, but it does not provide developers means to handle faults
Prior research

• IoT fault identification
  ▶ E.g., Sympathy [Ramanathan et al., Sensys 05]; DICE [Choi et al., DSN 18]; [SHARMA et al., TOSN 10]; ...
  ▶ They detect faults, but do not perform fault handling

• Little work on IoT fault handling
  ▶ Existing work lacks generality and focuses on specific environments or specific recovery techniques
  ▶ E.g., UAV sensor fault isolation [Tu et al., arXiv 18]
  ▶ E.g., edge device removal in Rivulet [Ardekani et al., ACM Middleware 17];
  ▶ E.g., transactions in Transactuations [Sengupta et al., ATC 19]

A need for a general IoT fault-handling system across a diverse set of deployments
Design of IoTRepair

• Focus on fault handling, not fault detection
  ▶ Assume a fault-detection module, which detects faults and gives faulty device IDs

• Provide a library with a set of fault-handling functions
  ▶ E.g., activate duplicate, retry, restart, checkpoint/rollback

• Developers use the library for flexible fault handling
  ▶ Through an API and a config file

• Provide an automated fault handler
  ▶ Try fault-handling functions using some scheme (order, ...)
  ▶ With an auto-generated and dynamically adjusted config file
IoTRepair system architecture

1. Initialization
   - Configurations for Connected Devices
   - Configurations for Installed Apps
   - General System Configurations

2. Fault Identification

3. Faulty Device ID

4. Read/Update Configurations
   - Automated Fault Handling
   - Developer APIs
   - Fault Handling Schemes
   - Fault Handling Functions

Real-time
A toolkit of fault-handling functions

• Device-based functions: fix the state of a single device
  ▸ Activate a redundant device
  ▸ Retry actuation (effective for fixing transient faults)
  ▸ Software and hardware restart

• Environment-based functions: fix the state of an IoT environment (multiple devices)
  ▸ Checkpoint/rollback
  ▸ Transaction: perform a series of actuations in an all-or-none fashion
System configuration

- Specify parameters used in fault-handling functions
  - E.g., what is the ID of a duplicate device
  - E.g., how many restart attempts
  - E.g., what type of rollback should be used
- Developers can write this manually or adjust an auto-generated one
Automated fault handler

• Invoking fault-handling functions according to some *scheme*
  ▶ A scheme controls the selection, ordering, and parameters of the fault-handling functions
  ▶ If some function is able to recover from the fault, stop
  ▶ If none can, notify the user

![Table 5: Execution order of functions in developed schemes. (1) Retry, (2) Replicate, (3) Software Restart, (4) Hardware Restart, (5) Rollback, (6) Notify User)
History based checkpoint/rollback

• The automated handler
  ▶ Continuously takes checkpoints of device states (sensor and actuator states) after an actuation

• Three kinds of rollbacks, when a fault is detected
  ▶ **Fail-recent**: rollback to the most recent checkpoint
  ▶ **Fail-norm**: rollback to the most frequent checkpoint that matches the current states of non-faulty sensors
  ▶ **Fail-safe**: first filter checkpoints in the history using fail-safe config of devices; then fail-norm
### Fail-Norm Rollback

<table>
<thead>
<tr>
<th>Motion</th>
<th>Presence</th>
<th>Door lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>Faulty (stuck at home)</td>
<td>Unlocked</td>
</tr>
</tbody>
</table>

**Door-lock-app:** unlock the door iff presence sensor says user is home

#### History of checkpoints:

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Motion</th>
<th>Presence</th>
<th>Door lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>On</td>
<td>Home</td>
<td>Unlocked</td>
</tr>
<tr>
<td>2</td>
<td>On</td>
<td>Away</td>
<td>Locked</td>
</tr>
<tr>
<td>3</td>
<td>Off</td>
<td>Home</td>
<td>Unlocked</td>
</tr>
<tr>
<td>50</td>
<td>Off</td>
<td>Away</td>
<td>Locked</td>
</tr>
</tbody>
</table>

- These two match the motion sensor state and the second one has higher frequency; so rollback the door to be locked
- When user is away, unlikely to detect motion
Config file: auto generation and dynamic adjustments

- During system initialization
  - Obtain a list of connected devices, their types and capabilities
  - Generate default configurations for devices based on their types/capabilities
- Some config info is adjusted at runtime
  - E.g., it detects duplicate devices based on runtime sensor/actuator states and put that info into the config file
Evaluation setup

• Simulated Home: 17 devices and 11 apps
  ▶ Trace-driven simulation runs based on generated events
  ▶ Events generated randomly, do not ensure full coverage

• Fault Injection: injected faults overwrite device states with faulty states
  ▶ Each injection determines false states, length of the fault, and fault type
### Evaluation: devices and apps

#### Devices used for evaluation

<table>
<thead>
<tr>
<th>ID</th>
<th>Device</th>
<th>Power (mW)</th>
<th>Read (ms)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.1</td>
<td>Motion sensor</td>
<td>66</td>
<td>0.1</td>
<td>double, 8B</td>
</tr>
<tr>
<td>S.2</td>
<td>Contact sensor</td>
<td>0.1</td>
<td>0.1</td>
<td>int, 4B</td>
</tr>
<tr>
<td>S.3</td>
<td>Temperature, pressure, altitude sensor</td>
<td>19.5</td>
<td>37.5</td>
<td>double, 8B</td>
</tr>
<tr>
<td>S.4</td>
<td>Presence sensor</td>
<td>1.3</td>
<td>0.5</td>
<td>int*3,12B</td>
</tr>
<tr>
<td>S.5</td>
<td>Smoke detector</td>
<td>30</td>
<td>0.96</td>
<td>double, 8B</td>
</tr>
<tr>
<td>S.6</td>
<td>Leak detector</td>
<td>80</td>
<td>0.1</td>
<td>double, 8B</td>
</tr>
<tr>
<td>A.1</td>
<td>Door lock, coffee machine, light</td>
<td>0.01</td>
<td>0.1</td>
<td>-</td>
</tr>
<tr>
<td>A.2</td>
<td>Alarm, air conditioner, heater, window, valve</td>
<td>100</td>
<td>0.1</td>
<td>-</td>
</tr>
</tbody>
</table>

† S is for Sensor and A is for Actuator.

#### Apps developed with devices for smart home

<table>
<thead>
<tr>
<th>ID</th>
<th>App Name</th>
<th>Description</th>
<th>S/A†</th>
</tr>
</thead>
<tbody>
<tr>
<td>App1</td>
<td>Motion-Activated-Lights</td>
<td>When motion detected, turn on lights. Turn off lights when motion not active.</td>
<td>S.1, A.1</td>
</tr>
<tr>
<td>App2</td>
<td>Smoke-Alarm</td>
<td>When smoke is detected sound alarm and unlock doors. When no smoke is detected turn off alarm.</td>
<td>S.5, A.2</td>
</tr>
<tr>
<td>App3</td>
<td>Temperature-Control</td>
<td>Keep temperature between 70-80 degrees (°F) by turning heater and air conditioner on and off</td>
<td>S.3, A.2</td>
</tr>
<tr>
<td>App4</td>
<td>Water-Leak-Detector</td>
<td>When leak is detected sound alarm. When no leak is detected turn off alarm.</td>
<td>S.6, A.2</td>
</tr>
<tr>
<td>App5</td>
<td>Welcome-Home</td>
<td>When the user arrives home, unlock doors and turn on coffee machine</td>
<td>S.4, A.1</td>
</tr>
<tr>
<td>App6</td>
<td>Secure-Patio</td>
<td>When user is not present and contact is detected, send text message to user</td>
<td>S.2, S.4, A.2</td>
</tr>
<tr>
<td>App7</td>
<td>Energy-Saver</td>
<td>If window is open and either heater or air conditioner are on, close window.</td>
<td>A.1, A.2</td>
</tr>
<tr>
<td>App8</td>
<td>Secure-Home</td>
<td>when user is not present home, lock doors and windows.</td>
<td>S.4, A.1</td>
</tr>
<tr>
<td>App9</td>
<td>Intruder-Detector</td>
<td>When user is not present home and motion is detected, send text message to user</td>
<td>S.1, S.4, A.2</td>
</tr>
<tr>
<td>App10</td>
<td>Alarm-Safety</td>
<td>When alarm is activated, turn on lights</td>
<td>A.1, A.2</td>
</tr>
<tr>
<td>App11</td>
<td>Morning-Air</td>
<td>Unlock and open windows, and close and lock windows at specific times</td>
<td>A.1, A.2</td>
</tr>
</tbody>
</table>

† S is for Sensor and A is for Actuator device (The sensors and actuators are listed in Table 2.)
Evaluation (still ongoing)

- Timing: How fast fault handling functions and schemes can resolve faults
  - How long each fault takes to run averaged between devices, faults, and config
  - How long each scheme takes to resolve fault types averages between devices, faults, and config
- Effectiveness: How effective our system can reduce fault effects
  - Single and Multiple faults injected in a given time
  - Cascading Faults
  - Implications of faults on Safety and Security
- Power Consumption: How much power our automated fault handler consumes
  - Measure events, actuations, and restarts to capture how much device-power is consumed
Effectiveness

• **Incorrect states** in a faulty execution: those that differ from the corresponding states in an identical but faultless execution

• Average the results of 100 runs, each with unique trace and faults
  ▶ New traces are generated using random events in sensor devices
  ▶ New fault injection changes the random elements of all faults, which is false state, duration, and repairability

• Baselines:
  ▶ Injected faults and no handling
  ▶ Injected faults and only device suppression
Automated Fault Handler Effectiveness

- Single: 63.51% decrease from NoHandle to Transient Resistant
- Multiple: 36.68% decrease from NoHandle to Transient Resistant
Projecting into the Future
What’s unique about IoT security?

- Interaction between digital and physical worlds
  - “Heater on -> temperature rise -> window open”
  - Soteria: build models from the apps and bake in rules for digital and physical world interaction (e.g., “heater on -> temperature rise”)
  - IoTMon [Ding & Hu, CCS 18]: use NLP techniques to find physical-world connection between apps
  - Future: need integration of better physical models (timing, velocity, etc.); side channels
What’s unique about IoT security?

- Distributed systems security
  - IoT is a network of devices (sensors and actuators)
  - Centralized solutions unlikely to scale to large IoT networks
  - Future: push security functions into devices (a la edge computing)
  - Challenge: implement security in lower-resourced devices
  - Challenge: concurrency
    - synchronization between devices; multiple security functions running concurrently
  - Challenge: IoT devices made by a vast number of manufacturers
    - A security solution needs to accommodate diversity and easy to incorporate new devices
Hyppönen’s law, and IoT safety and security

Security expert Mikko Hyppönen posited that...

"Whenever an appliance is described as being "smart", it's vulnerable."

As everything that can be smart will be smart, and interact with each other, they will become targets of adversaries.
Questions?