Cross-App Interference Threats in Smart Homes: Categorization, Detection and Handling

Haotian Chi, Qiang Zeng, Xiaojiang Du, Jiaping Yu

TEMPLE UNIVERSITY

UNIVERSITY OF South Carolina
Home Automation

- Home-automation **apps**
  - Lock the door when all leave
  - When arriving home, if the room is too hot, turn on A/C

Single-device automation

- **Smoke detection**

Subsystem-level automation

- Heating control
- Lighting control
- Camera surveillance
- Smart kitchen

Home-wide automation
Rule Abstraction

A home-automation app contains one or more rules, each in the form of T-C-A

- **Trigger**: “when resident arrives home”
- **Condition**: “if room temperature < 18°C”
- **Action**: “turn on heater”

Rule = TCA template + Configuration

- Configuration: app-device binding relations, threshold, etc.
Cross-App Interference (CAI) Threats

When multiple rules interplay in a home, they may interfere with each other.
Research Questions

- How to systematically categorize CAI threats?
- How to precisely detect them?
- How to assist users to handle them?
## Categorization of CAI Threats

### Rule 1: T1-C1-A1; Rule 2: T2-C2-A2

<table>
<thead>
<tr>
<th>Category</th>
<th>Basic Pattern</th>
<th>Auxiliary Pattern¹</th>
<th>ID</th>
<th>Validation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action-Interference Threats</td>
<td>A₁ = ¬A₂</td>
<td>T₁ = T₂, C₁ ∨ C₂</td>
<td>A.1</td>
<td>✓</td>
<td>R₁ and R₂ are executed simultaneously to perform conflict actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T₁ = T₂, ¬(C₁ ∨ C₂)</td>
<td></td>
<td>x</td>
<td>R₁ and R₂ cannot be both executed although they are both triggered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T₁ ≠ T₂, C₁ ∨ C₂</td>
<td>A.2</td>
<td>✓</td>
<td>R₁ and R₂ may be executed within a short period to perform conflict actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T₁ ≠ T₂, ¬(C₁ ∨ C₂)</td>
<td></td>
<td>x</td>
<td>R₁ and R₂ are unrelated and have no interaction.</td>
</tr>
<tr>
<td>Trigger-Interference Threats</td>
<td>A₁ → T₂</td>
<td>C₁ ∨ C₂, ¬(A₂ → T₁), A₁ ≠ ¬A₂</td>
<td>T.1</td>
<td>✓</td>
<td>R₁ triggers R₂, which does not interfere with R₁ in turn.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C₁ ∨ C₂, ¬(A₂ → T₁), A₁ = ¬A₂</td>
<td></td>
<td>x</td>
<td>R₁ triggers R₂, which performs a conflict action and thus invalidate R₁.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C₁ ∨ C₂, A₂ → T₁, A₁ ≠ ¬A₂</td>
<td>T.2</td>
<td>✓</td>
<td>R₁ and R₂ trigger each other alternately.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C₁ ∨ C₂, A₂ → T₁, A₁ = ¬A₂</td>
<td>T.3</td>
<td>✓</td>
<td>R₁ and R₂ trigger each other and perform conflict actions alternately.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>¬(C₁ ∨ C₂)</td>
<td>T.4</td>
<td>x</td>
<td>R₂ fails its condition checking and cannot be executed.</td>
</tr>
<tr>
<td>Condition-Interference Threats</td>
<td>A₁ → C₂</td>
<td>T₁ = T₂</td>
<td>C.1</td>
<td>✓</td>
<td>R₁ turns a constraint in R₂’s condition to true, which increases the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T₁ ≠ T₂</td>
<td>C.2</td>
<td>✓</td>
<td>probability of R₂ being executed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T₁ = T₂</td>
<td>C.3</td>
<td>✓</td>
<td>R₁ turns a constraint in R₂’s condition to false, which decreases the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T₁ ≠ T₂</td>
<td>C.4</td>
<td>✓</td>
<td>probability of R₂ being executed.</td>
</tr>
</tbody>
</table>
Challenges

- Extract rules from app code precisely
- Obtain user configuration without co-operation of IoT platforms
- Automatic CAI threat detection
System Overview

- Rule Extractor
  - extracts rule semantics from app source code
- Configuration Collector
  - collects configuration upon a new app is installed
- Threat Detector
  - analyzes whether any pair of rules causes CAI threats
- Frontend (A Mobile App)
  - presents detection result
Rule Extraction - Symbolic Execution

Rule Representation

Trigger:
(:subject), (:attribute)
(:constraint)
Condition:
(:data constraints)
(:predicate constraints)
Action:
(:subject) => (:command) (:paras) (:when) (:period)
(:data constraints)

1. Symbolic Input Identification
2. Trigger Event Identification
3. Data Constraint Construction and API Modeling
4. Predicate Constraint Construction
5. Sink Analysis
6. Control Flow Analysis
Configuration Collection - Code Instrumentation

```java
//... //Specify the phone that runs HomeGuard
input "patchedphone", "phone", required: true, title: "Phone number?"
//...
def updated() {
    // inserted code
    def appname = "ComfortTV"
    def devices = [[devRefStr:"tv1", devRef:tv1],
                    [devRefStr:"tSensor", devRef:tSensor],
                    [devRefStr:"window1", devRef:window1]]
    def values = [[varStr:"threshold1", var:threshold1]]
    collectConfigInfo(appname, devices, values)
}
def collectConfigInfo(appname, devices, values) {
    def uri = "http://my.com/appname:${appname}/"
    devices.each { dev ->
        uri = uri + dev.devRefStr + ":" + dev.devRef.getId() + "/
    }
    values.each { val->
        uri = uri + val.varStr + ":" + val.var + "/
    }
    sendSmsMessage(patchedphone,uri)
}
```

Collect app-device binding info.
Collect user-input values
Pack all configuration info.
Send the configuration info. out through the supported SMS or HTTP APIs.
## CAI Threat Detection

When a new app is installed, our system detects CAI threats between every pair of rules:

- Each CAI type is encoded as a set of **symbolic constraints**.
- Thus, CAI detection is transformed into solving SMT (satisfiability modulo theory) problems.

### Table: CAI Threat Detection Patterns

<table>
<thead>
<tr>
<th>Category</th>
<th>Basic Pattern</th>
<th>Auxiliary Pattern</th>
<th>ID</th>
<th>Validation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action-Interference</td>
<td>$A_1 = \neg A_2$</td>
<td>$T_1 = T_2, C_1 \land C_2$</td>
<td>A.1</td>
<td>✓</td>
<td>$R_1$ and $R_2$ are executed simultaneously to perform conflict actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T_1 = T_2, (C_1 \land C_2)$</td>
<td></td>
<td>✓</td>
<td>$R_1$ and $R_2$ cannot be both executed although they are both triggered.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T_1 \neq T_2, C_1 \land C_2$</td>
<td>A.2</td>
<td>✓</td>
<td>$R_1$ and $R_2$ may be executed within a short period to perform conflict actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T_1 \neq T_2, (C_1 \land C_2)$</td>
<td></td>
<td>✓</td>
<td>$R_1$ and $R_2$ are unrelated and have no interaction.</td>
</tr>
<tr>
<td>Trigger-Interference</td>
<td>$A_1 \Rightarrow T_2$</td>
<td>$C_1 \land C_2, (A_2 \Rightarrow T_1), A_1 \neq \neg A_2$</td>
<td>T.1</td>
<td>✓</td>
<td>$R_1$ triggers $R_2$, which does not interfere with $R_1$ in turn.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C_1 \land C_2, (A_2 \Rightarrow T_1), A_1 = \neg A_2$</td>
<td>T.2</td>
<td>✓</td>
<td>$R_1$ triggers $R_2$, which performs a conflict action and thus invalidate $R_1$.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C_1 \land C_2, A_2 \Rightarrow T_1, A_1 \neq \neg A_2$</td>
<td>T.3</td>
<td>✓</td>
<td>$R_1$ and $R_2$ trigger each other alternately.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C_1 \land C_2, A_2 \Rightarrow T_1, A_1 = \neg A_2$</td>
<td>T.4</td>
<td>✓</td>
<td>$R_1$ and $R_2$ trigger each other and perform conflict actions alternately.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\neg (C_1 \land C_2)$</td>
<td></td>
<td>✓</td>
<td>$R_2$ fails its condition checking and cannot be executed.</td>
</tr>
<tr>
<td>Condition-Interference</td>
<td>$A_1 \Rightarrow C_2$</td>
<td>$T_1 = T_2$</td>
<td>C.1</td>
<td>✓*</td>
<td>$R_1$ turns a constraint in $R_2$’s condition to true, which increases the probability of $R_2$ being executed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T_1 \neq T_2$</td>
<td>C.2</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$A_1 \not= C_2$</td>
<td>$T_1 = T_2$</td>
<td>C.3</td>
<td>✓*</td>
<td>$R_1$ turns a constraint in $R_2$’s condition to false, which decreases the probability of $R_2$ being executed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T_1 \neq T_2$</td>
<td>C.4</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

1 The auxiliary pattern of each CAI type does not conform to the basic pattern in other categories if not explicitly specified. We elide the negation constraints for conciseness.
Frontend App Presents Detection Result

AutoLock – Rule 1
GoodNight – Rule 2
AutoMode – Rule 5
BurglarAlarm – Rule 6
CAI threats in Market Apps

Run our CAI detection on 146 SmartApps in SmartThings official app repository
- 663 CAI instances in total
- 101 out of 146 apps are involved into at least one type of CAI threat
Real-world Testbed

- 18 official apps
- 24 devices

Devices and their layout

Detected (and verified) CAI threats

<table>
<thead>
<tr>
<th>App Name</th>
<th>Rule and Configuration</th>
<th>Set #</th>
<th>CAI Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurlingIron</td>
<td>When motion (1) detected, turn on oven (2) and fan (3) for 30 minutes.</td>
<td>1</td>
<td>A.1</td>
</tr>
<tr>
<td>Virtual Thermostat</td>
<td>When motion (4) detected, if temperature (8) is lower than 72°F, turn off fan (5).</td>
<td>2</td>
<td>A.2</td>
</tr>
<tr>
<td>NFCPlayToggler</td>
<td>When the user touches on mobile app, toggle switch (2) and door lock (5).</td>
<td>3</td>
<td>T.1</td>
</tr>
<tr>
<td>LockToWiseLeave</td>
<td>When presence sensor (7) becomes “not present”, lock door (8).</td>
<td>4</td>
<td>T.2</td>
</tr>
<tr>
<td>CurlingIron</td>
<td>When motion (1) detected, turn on oven (2) and fan (3).</td>
<td>5</td>
<td>T.3</td>
</tr>
<tr>
<td>SwitchChangesMode</td>
<td>When oven (2) is turned on, set home to “party” mode.</td>
<td>6</td>
<td>T.4</td>
</tr>
<tr>
<td>MakeItSo</td>
<td>When change to “Party” mode, unlock door (4) and turn on thermostat (8).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>It’sProHot</td>
<td>When temperature (8) exceeds 80°F, turn on fan (5).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnergySaver</td>
<td>When power usage (6) exceeds 3000 W, turn off fan (5).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmartHumidifier</td>
<td>When humidity (6) is below 30%, turn on humidifier (15); when humidity (6) exceeds 50%, turn off humidifier (15).</td>
<td>7</td>
<td>C.1</td>
</tr>
<tr>
<td>HumidityAlert!</td>
<td>When humidity (6) exceeds 50%, turn on ventilator (8); when humidity (6) is below 30%, turn off ventilator (8).</td>
<td>8</td>
<td>C.2, C.4</td>
</tr>
<tr>
<td>LightUpTheNight</td>
<td>When illuminance (3) exceeds 50 lx, turn on light (2); when illuminance (3) gets below 10 lx, turn on light (2).</td>
<td>9</td>
<td>C.3 (C.1)</td>
</tr>
<tr>
<td>BrightenDarkPlaces</td>
<td>When door (5) is opened, if illuminance (3) is below 10 lx, turn on light (2).</td>
<td>10</td>
<td>C.4</td>
</tr>
<tr>
<td>LetThereBeDark</td>
<td>When door (5) is opened, turn off lights (2); when door (5) is closed, restore the state of lights (8).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ForgivingSecurity</td>
<td>When motion sensor (4) or (5) becomes “active”, if the home is in “Away” mode, send alarm (2).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduled Mode Change</td>
<td>Set home to “Away” mode at 10 am and set home to “Night” mode at 6pm.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ForgivingSecurity</td>
<td>When motion sensor (4) becomes “active”, if home is in “Work” mode, turn on light (2) after 1 second.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rise and Shine</td>
<td>When motion sensor (5) becomes “active”, set home to “AlertHome” mode (“Work” mode).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GoodNight</td>
<td>When motion (12) detected, if switches (13) and (14) are all off, set home to “sleep” mode.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Once a Day</td>
<td>Turn on fan (6) at 11 pm and turn off fan (6) at 12 am.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MakeItSo</td>
<td>When changed to “sleep” mode, lock door (8).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation of Performance

Rule extraction speed
- 1,341 millisecond per app on average

CAI detection speed
- Averaged 671 millisecond
Related Work

<table>
<thead>
<tr>
<th>Publication Date</th>
<th># of CAI Threat Types</th>
<th>Systematic Categorization?</th>
<th>Symbolic Threat Modeling?</th>
<th>CAI Threat Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT [21]</td>
<td>Apr 2015</td>
<td>1</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Surbatovish et al. [20]</td>
<td>Apr 2017</td>
<td>1</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>IoTA [22]</td>
<td>Oct 2017</td>
<td>3</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>IoTSan [18]</td>
<td>Oct 2018</td>
<td>2</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>IoTMon [16]</td>
<td>Oct 2018</td>
<td>1</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>SafeChain [23]</td>
<td>Oct 2019</td>
<td>1</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HOMEGUARD</td>
<td>Aug 2018</td>
<td>10</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Conclusion

- **First** comprehensive categorization of CAI threats (first posted on arXiv in **Aug 2018**)
- **First** symbolic representation of CAI threats
- **First** work that leverages SMT for CAI threat detection
- An end-to-end implementation without co-operation of IoT platforms
Thank You!

Q & A
Another Example

A SmartApp in SmartThings public repository:

```python
definition(
    name: "Light Up the Night",
    namespace: "smartthings",
    author: "SmartThings",
    description: "Turn your lights on when it gets dark and off when it becomes light again.",
    category: "Convenience",
    iconX2Url: "https://s3.amazonaws.com/smartapp-icons/Meta/light_outlet-luminance@2x.png"
)

def installed() {
    subscribe(lightSensor, "illuminance", illuminanceHandler)
}

def illuminanceHandler(evt) {
    def lastStatus = state.lastStatus
    if (lastStatus != "on" && evt.integerValue < 30) {
        lights.on()
        state.lastStatus = "on"
    } else if (lastStatus != "off" && evt.integerValue > 50) {
        lights.off()
        state.lastStatus = "off"
    }
}
```

![Diagram of SmartApp logic]
Intra-App Attacks

Intra-app attacks (a.k.a., malicious apps, or malware) is a well-known threat

- Gain unauthorized access to smart home devices
  - **Send malicious commands** to home IoT devices
    - Unlock a door when the owner is absent
    - Turn on the oven over a long time to cause a fire accident
    - DoS the devices to demand a ransom payment to restore them
  - **Exfiltrate sensitive data** from the sensor devices
    - Medical device data expose the health condition of the homeowner
    - A presence sensor reveals the occupation of a house
CAI Threat Detection (Cont’d)

\[ T_1 = T_2 \] and \( A_1 = \neg A_2 \) are relatively straightforward to determine by respectively looking at whether:

- Two rule triggers subscribe to the same event (same device, attribute and value)
- Two rule actions control the same device with contradictory commands

\[ A_1 \leftrightarrow T_2 \] has two cases:

- \( A_1 \) and \( T_2 \) are handling the same device and attribute
  - For example, turning on a ON/OFF switch triggers a rule that subscribes to the switch’s “ON” event
- \( A_1 \) controls an actuator that changes the reading of a sensor subscribed to by \( T_2 \) via a physical channel
  - For instance, turning on a heater triggers a rule that subscribes to the reading of a temperature sensor

\( C_1 \land C_2 \): condition overlapping detection

- We often need to detect if two rules’ conditions have overlaps. For example,
  - \( tSensor1.\text{temperature} > 70, tv1.\text{switch} == \text{“on”}, 13.00 < \text{time} < 19.50 \)
  - \( tSensor1.\text{temperature} < 75, tv1.\text{switch} == \text{“off”} \)
- The overlapping detection is transformed into a constraint satisfaction problem
  - We use the Java Constraint Programming (JaCoP) library as the solver
Risk Ranking - Help Users to Handle CAI Threats

A detected CAI instance is labeled with a risk level \( L \in \{\text{low, medium, high}\} \).

- **Two observations**
  - Each CAI threat type has specific effects on the involved rules:
    - Positive (+): unexpectedly triggers the execution of a rule
    - Negative (-): unexpectedly invalidates the execution of a rule
    - Loop (o): makes two rules trigger each other to form a loop execution
  - The risk of implication of three effects (+, -, o) depends on two factors:
    - The functionality of the involved rule
    - The sensitivity of the rule action

Consider an example. R1 (“unlock the door and open the window when smoke is detected”) and R2 (“open the window when air quality is low”)

- If R3 imposes negative effects on “open the window”, it imposes a higher risk to R1 than R2 since R1 is for safety purpose and R2 is for comfort purpose.
- If R3 imposes positive effects on “open the window”, it does not break the functionality of both rules but is still considered dangerous due to the sensitivity of the action “open the window” itself.
Risk Ranking (Cont’d)

CAI threat instance ← \( I(\text{type}, R_1, R_2) \), where \( R_1 = (T_1, C_1, A_1) \) and \( R_2 = (T_2, C_2, A_2) \);

A risk level ← \( L \in \{-1, 0, 1\} \) (low, medium, high);

The effect of \( I \) on each rule \( R_i \) ← \( e_i \), determined by type

The category of a rule ← \( c_i \)

The command type of \( R_i \)’s action ← \( cmd_i \)

The risk imposed by \( I \) on each rule \( R_i \) is defined as:
\[
L_i = \max(M_1(e_i, c_i), M_2(e_i, cmd_i)),
\]
where \( M_1 \) and \( M_2 \) are two mappings designed based on the two observations.

The mapping table of \( M_1 \):

<table>
<thead>
<tr>
<th>Category</th>
<th>Effect(+)</th>
<th>Effect(−)</th>
<th>Effect(o)</th>
</tr>
</thead>
<tbody>
<tr>
<td>safety rules</td>
<td>low</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>non-safety rules</td>
<td>low</td>
<td>medium</td>
<td>medium</td>
</tr>
</tbody>
</table>
Risk Ranking (Cont’d)

To establish the mapping table of $M_1$, i.e., the risk level regarding CAI effect $e_i$ and command type $cmd_i$, an algorithm is designed to extract such knowledge from 146 official SmartApps.

The simple insight in the algorithm is that security-sensitive commands are used more frequently in SmartApps of the “Safety & Security” category than those of other categories; with non-sensitive commands, the opposite is true.

### Algorithm 1: The algorithm for extracting device control sensitivity under different CAI effects from SmartApps

```
Input : Apps ← the source code of all SmartApps
Output: Risk knowledge model of capability-supported commands RM
foreach app ∈ Apps do
    Rules ← ExtractRuleSemantics (app)
    cmd ← rule.action.command
    capCmd ← Concatenate (capability, cmd)
    oppCapCmd ← FindOppositeCmd (capCmd)
    if category is “Safety & Security” then
        count[capCmd]['+'][]['low'][]++
        count[capCmd]['-'][]['high'][]++
        count[oppCapCmd]['+'][]['high'][]++
        count[oppCapCmd]['-'][]['low'][]++
    else
        count[capCmd]['+'][]['low'][]++
        count[capCmd]['-'][]['medium'][]++
        count[capCmd]['-'][]['medium'][]++
        count[capCmd]['+'][]['medium'][]++
    foreach e ∈ {‘high’, ‘medium’, ‘low’} do
        RM[capCmd][e] = max(count[capCmd][e]['high'],
                            count[capCmd][e]['medium'],
                            count[capCmd][e]['low'])
```

### Part of the results (out of 46 commands)

<table>
<thead>
<tr>
<th>Capability.command</th>
<th>Effect(+)</th>
<th>Effect(−)</th>
<th>Effect(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm.off</td>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>alarm.siren</td>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>light.off</td>
<td>L</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>location.setLocationMode</td>
<td>L</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>lock.lock</td>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>lock.unlock</td>
<td>H</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>switch.on</td>
<td>L</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>valve.close</td>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>valve.open</td>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
</tbody>
</table>