Discovering IoT Physical Channel Vulnerabilities

2022 CCS

作者

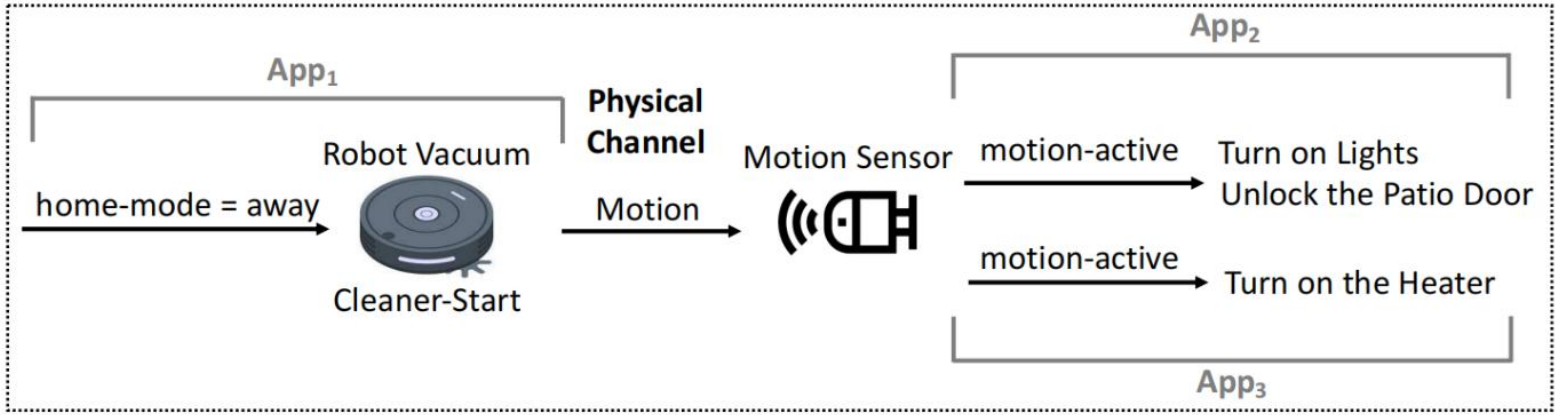
- Muslum Ozgur Ozmen[1], Xuansong Li[2], Andrew Chu[3], Z. Berkay Celik[1], Bardh Hoxha[4], Xiangyu Zhang[1]

- [1]Purdue University, [2]School of Computer Science and Engineering, Nanjing University of Science and Technology, [3]University of Chicago, [4]Toyota Research Institute North America

要解决的问题

- 攻击者可以利用应用程序之间的物理交互漏洞，将用户和环境置于危险之中，例如，攻击者打开加热器，触发一个应用程序，当温度超过阈值时打开窗户，从而闯入房屋。
- 现有的工作未能达到足够的广度和保真度，无法将应用程序代码转化为它们的物理行为，或提供不完整的安全策略，导致准确性差和误报警。

之前的工作基于设备的用例定义安全规则，以防止物理交互漏洞。但是这些规则不考虑意外交互，这些交互发生在设备和应用程序的预期使用之外，并在智能家居中意外触发动作。如：



本文的工作

- 将应用程序源代码中的驱动命令和传感器事件转换为物理执行模型，以定义它们的物理行为。
- 引入了一种新的复合物理执行模型体系结构，它定义了交互应用程序的联合物理行为。
- 使用预期的/非预期的物理通道标签开发新的安全策略。通过优化引导的证伪来正式验证CPEM上的策略。

架构设计

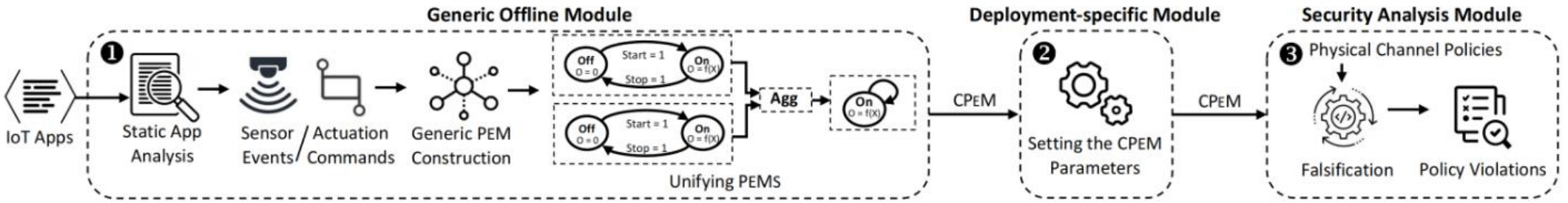
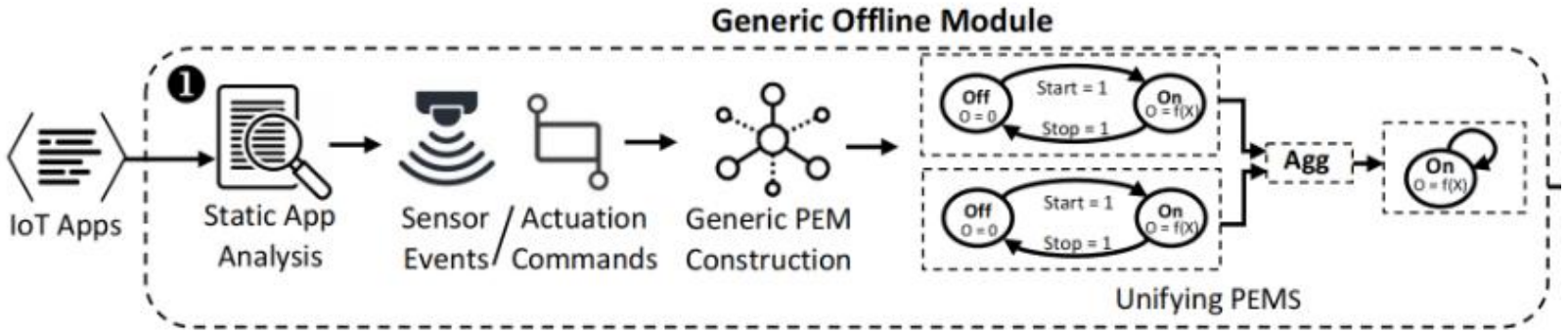


Figure 2: Overview of IoTSeer's architecture

思路：

1. 为IoT的App进行建模，形成Physical Execution Model(PEM, 是hybrid I/O automation)
2. 多个PEM结合，形成Composite Physical Execution Model(CPEM)
3. 给CPEM设置相关参数
4. 建立物理通道应当满足的规则
5. 利用设置好参数的CPEM检测，看是否违反相关规则。

1. Generic Offline Module



1.1. Static App Analysis

目的：需要应用程序的事件(event)、驱动命令(actuation command)和与每个命令关联的触发条件(trigger condition)。

挑战：物联网平台是多样化的，每个平台都为自动化提供了不同的编程语言。

解决方法：利用现有的物联网应用静态分析和解析工具（原文参考文献16,17,61）。这些工具可以模拟应用程序的生命周期，从程序间控制流图(interprocedural control flow graph, ICFG)包括入口点和事件句柄。同时还能提取出：

1. 设备和事件；
2. 每个事件会执行的动作(actuation)；
3. 执行动作的条件；

例如：从 `When the temperature is higher than 80° $F$, if the AC is off, then open the window` 中IotSEER可以获取到event: `temp > 80` , command: `window-open` , trigger condition: `AC-off` 。

1.2. 构建PEM

为Actuation Command构建PEM

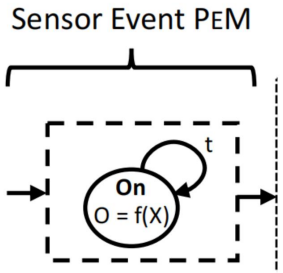
形式化定义为 `Ha = (Q, X, f, ->, U, O)`

其中：
Q是一组离散的状态（比如开、关）；
X是连续的变量（比如温度、音量）；
f是flow function，表示连续变量的演化；
->定义离散转换；
U/O表示输入/输出的变量。最终Ha的输出是对应命令产生的影响。

flow function:作者为每个物理通道定义一个单独的通用流函数。它们是连续物理通道(如温度)的微分方程和瞬时通道(如声音)的代数方程。流函数以两个参数作为输入：器件属性(device property)和到执行器的距离(distance from the actuator)，并输出执行器在该距离上对物理通道的影响。

为Sensor Event构建PEM

作者将事件的PEM定义为混合单状态的I/O自动机(with single state)，Q={on}，并且每经过时间t（传感器对其测量数据采样的频率）就会产生一次自跃迁。



Sensor Event PEM接受一个灵敏度级别参数，该参数定义物理通道中的最小更改量(threshold)，以改变传感器的读数。阈值函数输出传感器读数，指示物理通道水平是否等于或大于灵敏度水平。
如果传感器测量布尔类型的值(例如，是否运动)，PEM输出一个位表示“检测到”或“未检测到”事件；如果传感器进行数值读数(如温度)，则输出数值。

1.3. 聚合成CPEM

Algorithm 1 Composition of Physical Behavior of Apps

Input: Actuation Command PEMs (H_a), Sensor Event PEMs (H_s), Apps ($\mathcal{L}_{app}$)
Output: CPEM ($\mathcal{M}$)

```
1: function COMPOSITION(H_a, H_s, L_app)
2:     for H_i ∈ H_a, H_j ∈ H_s do
3:         UNIFY(H_i, H_j)                          ▷ Command to sensor event transitions
4:     end for
5:     for app_i ∈ L_app do
6:         ⟨L^i_command, L^i_condition, L^i_event⟩ = STATICANALYSIS(app_i)
7:         for s ∈ L^i_event, a ∈ L^i_command do
8:             if a.condition = true then
9:                 UNIFY(H_s, H_a)                  ▷ Sensor event to command transitions
10:            end if
11:        end for
12:    end for
13:    for a_1 ∈ L_command do
14:        if a_1 ∈ L^i_event and a_i.condition = true then
15:            UNIFY(H_a1, H_ai)                    ▷ Software channel transitions
16:        end if
17:    end for
18:    for H_j ∈ H_s do
19:        AGG(H_j.U)                               ▷ Aggregate inputs of the sensor events
20:        for H_k ∈ H_s do
21:            if H_j.O = H_k.U then DEP(H_j → H_k)  ▷ Dependency
22:            end if
23:        end for
24:    end for
25:    return M = ⋃(H_a, H_s)                       ▷ Return CPeM
26: end function
```

通过匹配Sensor Event和命令的物理通道来识别交互

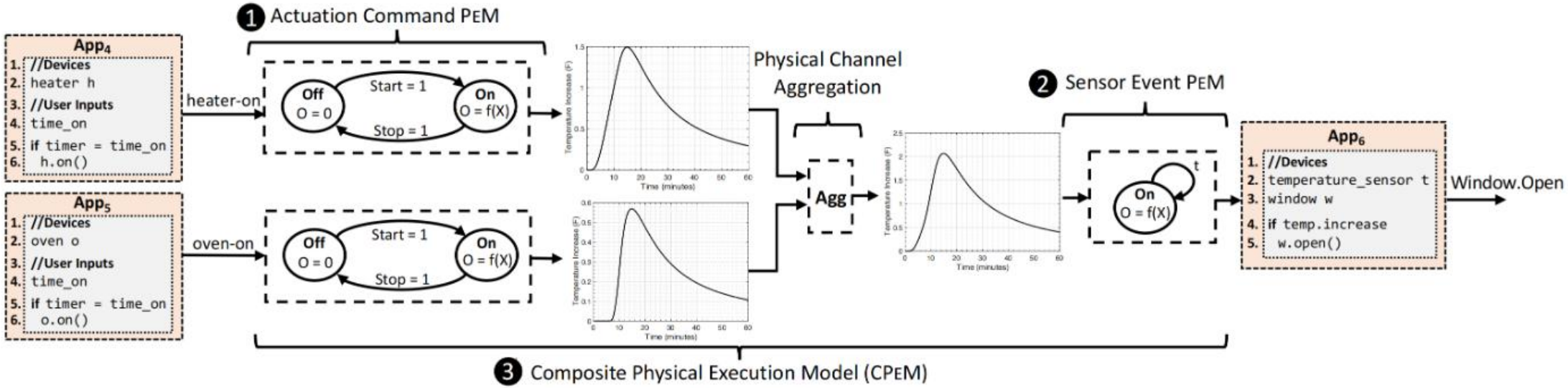
- 首先，如果传感器测量命令影响的物理通道，则添加从Command PEM (Ha)输出到Sensor Event PEM (Hs)输入的转换(第2-4行)。
- 其次，软件(software)和物理通道可以触发应用程序的事件处理程序，并在满足应用程序的条件时调用命令。

对于物理通道，作者添加了从Sensor Event PEM (Hs)到Commend PEM (Ha)的转换(第5-12行)。

对于软件通道，如果应用程序在a1发生时调用a2，则添加从Command PEM (Ha1)到另一个Commad PEM (Ha2)的转换(第13-17行)。

形式表现：

$$H_a \rightsquigarrow H_s,\ H_s \longrightarrow H_a,\ \text{and}\ H_a \longrightarrow H_a.$$



当App4和App5中的指令启动后，会产生如下转换

$$\mathtt{H_a\{heater\text{-}on\}} \rightsquigarrow\triangleright \mathtt{H_s\{temp\text{-}increase\}}$$
$$\mathtt{H_a\{oven\text{-}on\}} \rightsquigarrow\triangleright \mathtt{H_s\{temp\text{-}increase\}}$$

而温度的升高又会对App6产生影响，即如下转换：

$$\mathtt{H_s\{temp\text{-}increase\}} \rightarrow \mathtt{H_a\{window\text{-}open\}}$$

聚合写法：$\mathtt{Agg}(\mathtt{H_a^1\{heater\text{-}on\}}, \mathtt{H_a^2\{oven\text{-}on\}}) \rightsquigarrow\triangleright \mathtt{temp\text{-}increase}$

AGG运算符的输出是基于物理通道的单位定义的：

- 对于线性范围的通道（如温度），其输出为Command PEM输出的求和；
- 对于log范围的通道（如音量），其输出将会转换成线性后进行聚合，即

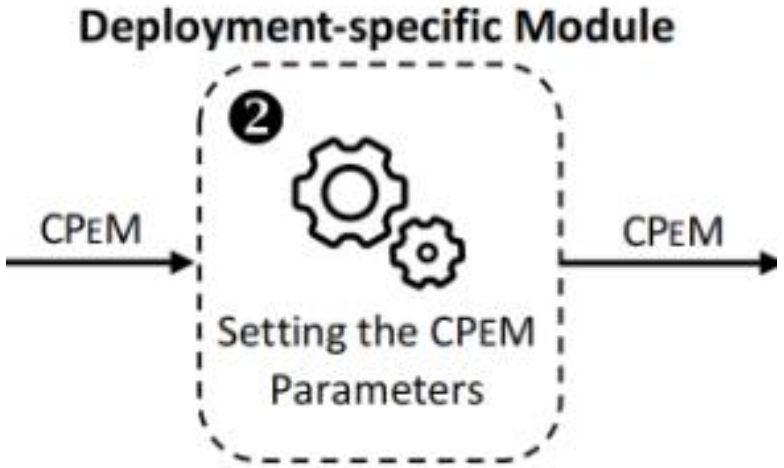
$$10 \times \log_{10}\left(\sum_{i=1}^{n} 10^{H_a^i/10}\right)$$

与此同时，物理通道的另一个特性是物理通道(pj)可能取决于另一个通道(pi)（例如，当环境温度升高时，空气-水容量增加，影响湿度传感器的读数）。如果pi的变化影响pj，一个Sensor Event PEM的输出可能会影响另一个Sensor Event PEM的读数。Generic PEM允许我们轻松地识别依赖性，通过迭代获取每个Sensor Event PEM并检查它是否用于另一个Sensor Event的阈值函数(第20-23行)。为了解决这个问题，作者使用

$$\mathrm{DEP}\left(H_s^i \rightarrow H_s^j\right)$$

表示从温度传感器的PEM输出到湿度传感器PEM的输入转换。

2. Deployment-specific Module



需要设置具体的设备参数，以确保CPEM精确模拟应用程序的物理行为。根据上一节所述，具体的参数有两个：

* 设备的特征值
* 距离。

2.1. Setting the Device Property Parameter

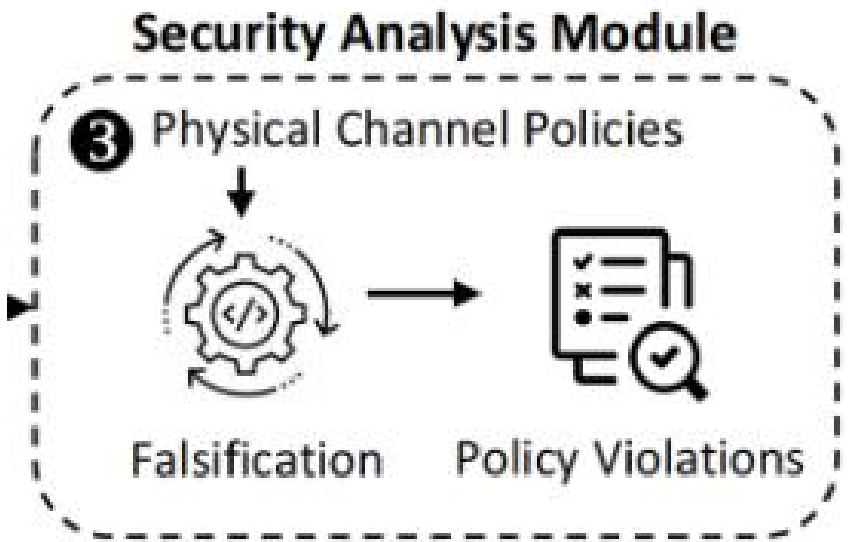
- 第一种方法是使用已安装设备的数据表。然而，在原型实现中，作者意识到数据表可能是不完整的，或者可能出现差异，比如设备老化。
- 第二种方法是使用System Identification(SI)，这是一种基于学习的方法，通常由控制工程师使用实验数据轨迹来估计物理过程的参数或模型。

    1. IoTSEER单独激活每个执行器并收集传感器测量数据。
    2. 运行带有设备属性参数的PEM，并获得传感器的trace。
    3. 计算实际设备和PEM的trace之间的(�, �)-closeness（这个closeness是保真度度量的一个量，该值决定了两个trace在timing和value之间的差异，其中�也叫作偏差分数）。
    4. 对设备属性参数进行二分搜索，以获得使偏差评分最小的最优值。

2.2. Setting the Distance Parameter

IoTSEER使用接收信号的强度(RSSI)的距离估计技术，这种技术利用距离和RSSI的反比来估计两个设备之间的距离。尽管这种方法可能会在距离参数中产生误差，但作者的评估表明，这种误差对IoTSEER的policy violation识别的影响是很小的。

3. Security Analysis Module



3.1. Identifying Physical Channel Policies

核心想法：打标签，设置Intended和Unintended两种标签

方法：使用SmartAuth（一种NLP技术）自动从其描述中提取与应用程序相关的活动，然后IoTSEER使用Word2Vec表示来计算活动和命令之间的语义距离。然后，它为距离低于阈值的命令分配intended（Int）标签，并为其他的分配unintended（UnInt）标签。同时IoTSEER还允许用户根据自己的需要来更改标签。

样例：”open the windows when you are cooking”，SmartAuth会检测到app和cooking activity有关，IoTSEER获取该应用程序的活动，并检查安装在智能家居中的任何执行器是否在语义上与该活动相关。例如，烹饪活动在语义上与打开烤箱和打开厨具上的命令有关，因此，IoTSEER将Int分配给它们。

3.1. Identifying Physical Channel Policies

如果一个应用程序的描述没有表明一个活动，或没有一个命令与该活动的语义相关，IoTSEER将根据应用程序的传感器事件分配标签。这些以运动或声音传感器事件为条件的应用程序被用于检测智能家居中用户和入侵者的存在。例如下图中：

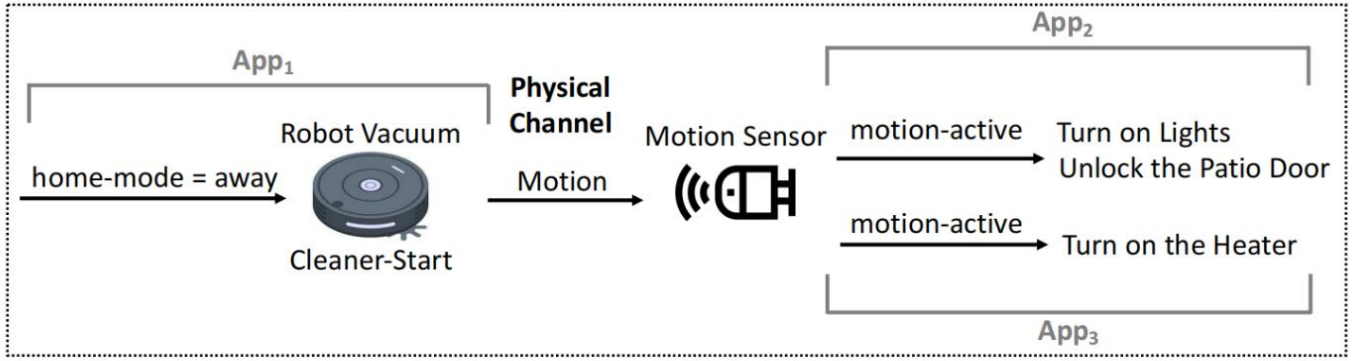


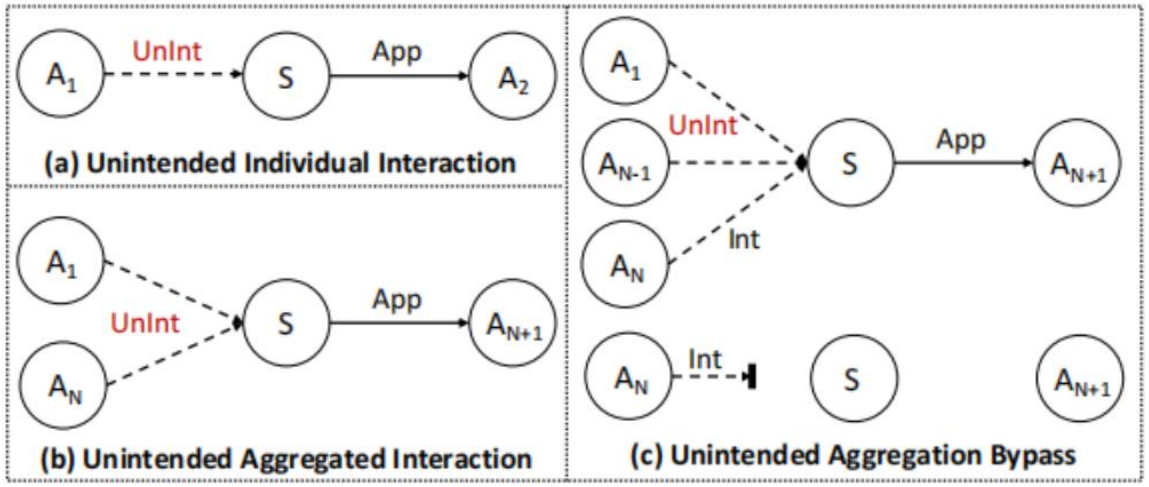
Figure 1: Illustration of physical interactions: the vacuum robot is activated when the user is not home, which unlocks the patio door and turns on the lights and heater.

App2当检测到运动时打开灯、开门，用于在用户存在时触发。IoTSeer为此类应用程序的所有命令分配了UnInt标签，因为只有对于用户和入侵者来说，这种操作才是intended。

安装有温度、湿度、烟道和照度通道条件的应用程序，并在物理通道状态达到特定条件时触发。例如，当温度高于一个阈值时，一个打开空调的应用程序会根据环境温度水平来控制空调。IoTSeer为此类应用程序的所有命令分配Int标签，因为它们的预期用途只取决于物理通道的条件，而不管哪些命令会影响它们。

3.1. Identifying Physical Channel Policies

作者定义了三个基于意图的策略，如下图所示，它们用于识别创建不希望和不安全的系统状态的意外物理交互。



(a) Actuation commands对物理通道的影响不能非预期地(Unintended)触发一个应用程序的传感器事件并调用其设备操作。

(b) 多个命令对物理通道的聚合影响不能非预期地(Unintended)触发一个应用程序的传感器事件并调用其设备操作。

(c) 如果命令的预期影响不会触发应用程序的传感器事件，他和其他的命令的聚合影响也不能非预期地(Untended)触发

- **Device-Centric Policies**

虽然基于意图的策略从非预期的物理交互中检测到不安全状态，但Int标记的物理通道也可能导致不安全状态。例如，加热器对温度传感器的预期影响可能会触发一个应用程序，当温度超过一个阈值时打开窗户。

为解决上述问题，提出了以设备为中心的策略（也就是在以上策略的基础上增加了时间限制），部分策略样例如下：

ID	Policy Description	Formal Representation
DC_2	When the home is in the away mode, the window must be closed.	$\square(\texttt{mode-away} \rightarrow \texttt{window-close})$
DC_3	A device must not open, then close and then reopen (actuation loop) within t seconds.	$\square\neg(\texttt{on} \wedge \bigcirc\Diamond_{[0,t]}(\texttt{off} \wedge \bigcirc\Diamond_{[0,t]}\texttt{on}))$
DC_5	The alarm must go off within t seconds after smoke is detected.	$\square(\texttt{smoke-detected} \rightarrow \Diamond_{[0,t]}\texttt{alarm-on})$
DC_6	The main door must not be left unlocked for more than t seconds.	$\Diamond_{[0,t]}\texttt{door-lock}$
DC_9	The door must always be locked and lights must be off when the home is in the away mode.	$\square(\texttt{mode-away} \rightarrow \texttt{door-lock} \wedge \texttt{light-off})$

3.2. Validating Policies on CPEM

在用Metric Temporal Logic (MTL) 表示确定的策略后，IoTSeer执行CPEM（混合I/O自动机），并收集执行器和传感器的trace以验证策略。在每次执行时，CPEM将应用程序的激活时间作为输入，即应用程序调用其命令和命令PEM转换到“on”状态的时间。CPEM模拟命令和传感器事件的统一物理行为，并输出PEM的轨迹。轨迹（v, t）由一个周期时间戳t和一个物理信道值v组成。每个命令PEM的v显示它对通道的影响程度，而每个传感器事件PEM的v显示它的测量值。这些跟踪还包括命令/事件的标签（Int/UnInt）和应用程序id。

- **Policy Validation Challenges**

物理通道值和应用程序激活时间是连续的；因此，CPEM的状态空间变得无限大，这使得CPEM上的正式验证方法（如模型检查）无法决定.

为了解决这个问题，作者首先实现了一种网格测试方法，网格测试确定CPEM是否满足有限的应用程序激活时间集下的策略——应用程序调用驱动命令的时间。如下算法提出了对CPEM进行策略验证的网格测试方法，作者将应用程序的激活时间设置为一个网格（t0：Δt：tend）（第3行）。该算法通过搜索激活时间组合来执行CPEM。然后，它使用鲁棒性度量（第4-6行）验证来自PEM的每个执行跟踪上的策略，其中负的鲁棒性值表示违反了策略。

Algorithm 2 Grid-Testing

Input: CPEM ($\mathcal{M}_{H_a,H_s}$) with command PEMs (H_a) and sensor event PEMs (H_s), parameters (x - distances among devices), inputs (U - apps' activation times $t_0 : \Delta t : t_{end}$), policy (ψ).
Output: P=(inputs, apps, dist, atime, y)

1: **function** GRID_TEST(H_a, H_s, x, U, $\mathcal{M}_{H_a,H_s}, \psi$)
2: **for** j $\in H_s, H_{OP} \subseteq H_a$ **do**
3: **for** Different activation times in H_{OP} **do**
4: **if** $\Phi(\mathcal{M}_{H_{OP},H_s}, x, u) \nvDash \psi$ **then**
5: $P \leftarrow P \cup \{x, u, \Phi(\mathcal{M}_{H_{OP},H_s}, x, u)\}$
6: **end if**
7: **end for**
8: **end for**
9: **return** P
10: **end function**

然而随着交互应用程序数量的增加，网格测试并不能扩大更大的分析，并且可能会由于输入离散化而错过policy voilation。

- **Optimization-Guided Falsification.**

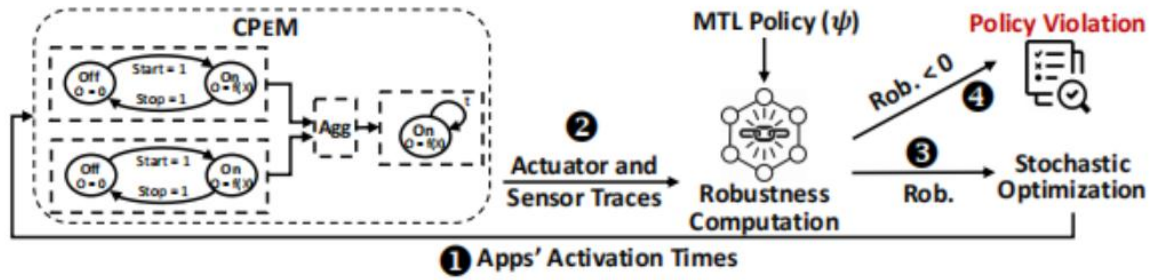


Figure 6: Overview of falsification to find policy violations.

具体来说：

1. 使用一种优化算法(Optimization algorithm)，通过采样激活时间来搜索策略违反；
2. 执行CPEM并记录PEM的执行器和传感器traces；
3. 从trace中，计算一个鲁棒性值，以量化MTL公式与策略违反有多接近；其中负值表示违反，正值表示满足；
4. 然后采样器在范围内将另一个输入喂到CPEM（类似fuzz中的mutation）。采样器的目标是最小化鲁棒性值，以寻找策略违反（和fuzz很像）。

输入生成的终止条件是：（1）发现违反策略或（2）满足用户定义的最大迭代次数。

以下是一个policy violation的样例结果：

```
"Policy Violation": {
  "Input": robot-vacuum-on,
  "Apps":
    "app_a": e:= timer, a:= robot.vacuum-on,
    "app_b": e:= mot-active, a:=patDoor.unlock,
  "Activation Time": 10,
  "Distance": 2,
  "Physical Channel Values":
    "t = 10": robot-vacuum.move,
    "t = 11": (UnInt) mot-active,
    "t = 11": patDoor.unlock}
```

Listing 1: An example output of policy validation.