Evasion Attacks and Defenses on Smart Home Physical Event Verification

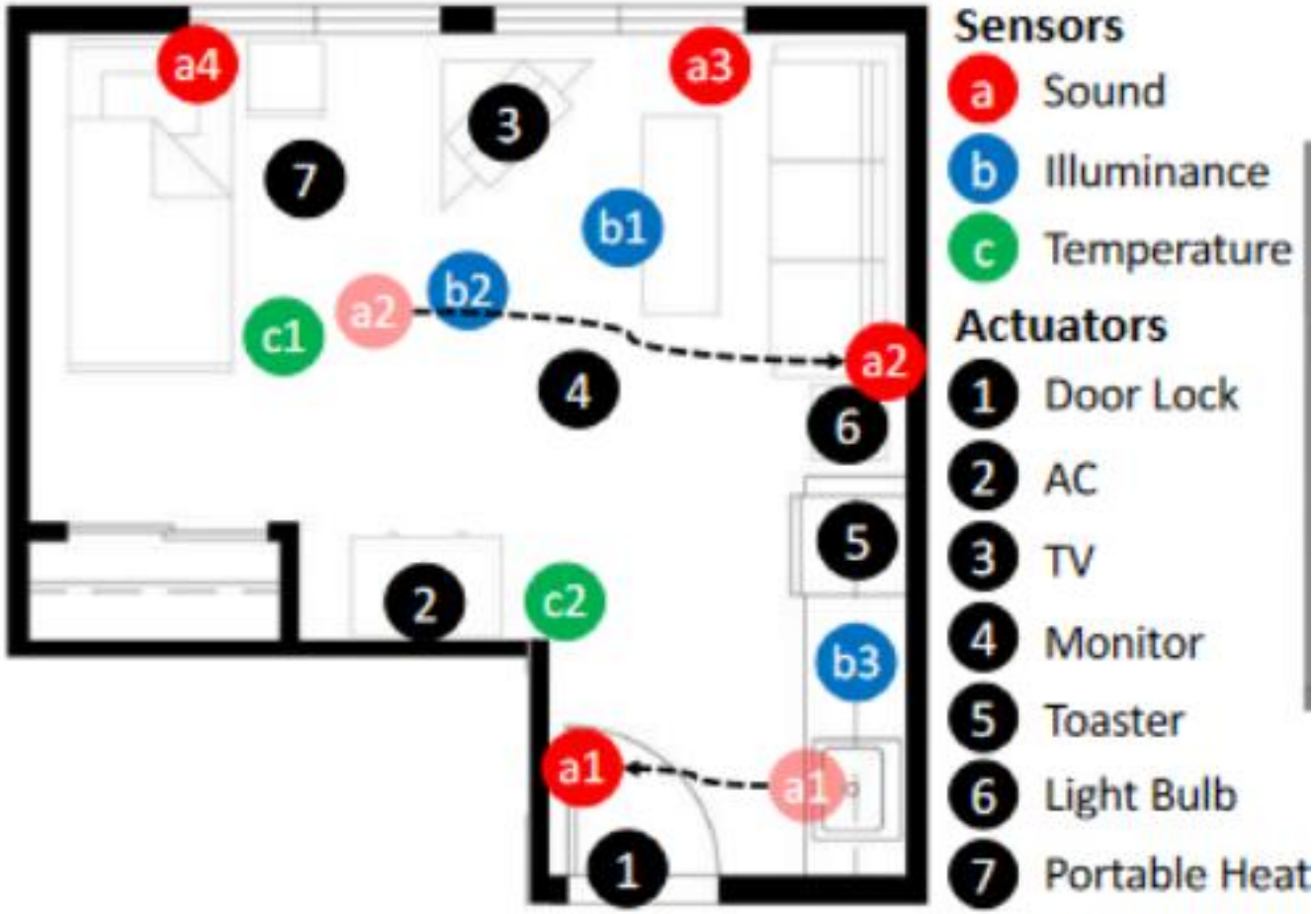
2023 NDSS

作者

- Muslum Ozgur Ozmen, Ruoyu Song, Habiba Farrukh, and Z. Berkay Celik

- Purdue University

BackGround


a4
a3
3
7
b1
b2
a2
c1
a2
4
6
5
2
c2
b3
a1
a1
1
Sensors
a Sound
b Illuminance
c Temperature
Actuators
1 Door Lock
2 AC
3 TV
4 Monitor
5 Toaster
6 Light Bulb
7 Portable Heat

BackGround

两种攻击方式

* Event Spoofing
* Masking

BackGround

现有防御机制：Event Verification System (EVS)

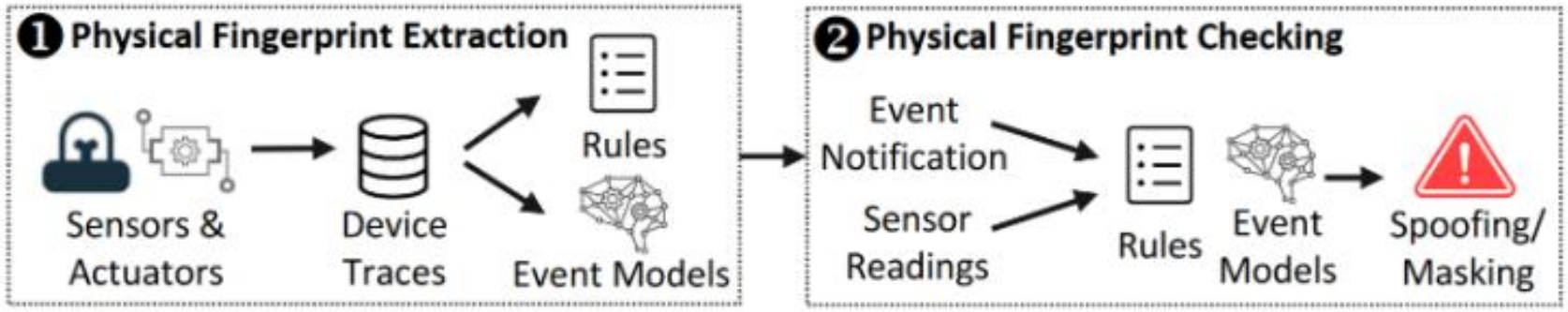


Fig. 1: Overview of event verification systems (EVS)

EVS有两类：Rule-based EVS (R-EVS), ML-EVS

```
Rule: light-on --- {S1.Illum = High, S2.Illum = High}
```

```
Rule: light-on --- {S1.Illum = High, S2.Illum = High}
```

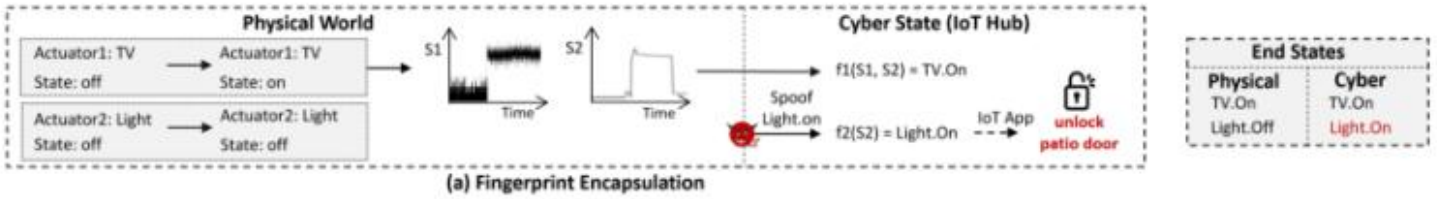
- R-EVS: 当收到相应的event notification时检查是否符合相应的rule。

  如：为了检查spoofing，当收到了 `light-on` 消息时，R-EVS会检查光线传感器S1和S2的读数看是否受到攻击（如果读数有偏差，则认为实际上应该没有执行 `light-on` ，也就是遭到了spoofing）。为了检查masking，R-EVS会定期查看sensor的读数，如果S1、S2读数变成了High，但是没有收到 `light-on` 的消息，则会认为是masking attack。

- ML-EVS: 为了检查spoofing，ML-EVS根据学习到的事件模型，从运行时传感器读数预测物理事件是否发生。如果收到通知，但模型预测这个事件没有发生，则认为是spoofing；为了检查masking，他们持续从传感器读数中提取特征。如果事件模型预测发生了一个物理事件，但是没有收到它的通知，则认为是masking。

针对EVS，有三种逃逸攻击（Evasion Attack）

5.1. Fingerprint Encapsulation



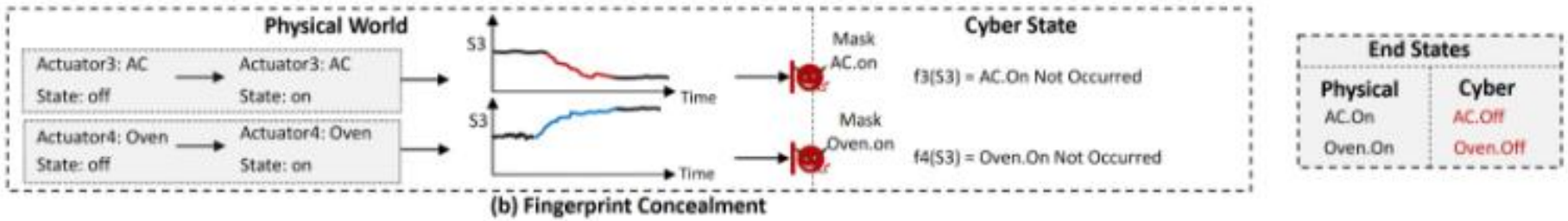
(a) Fingerprint Encapsulation

产生攻击的根本原因是一个事件(Ei)对传感器的影响包含了另一个事件(Ej)的影响。从形式上讲，当满足如下条件时R-EVS很容易受到这种攻击:

$$\mathrm{Rule}_1 : \mathrm{E_i} \leftrightarrow \{\mathrm{S_{i,1}} = \mathrm{High}, \mathrm{S_{i,2}} = \mathrm{High}, \ldots, \mathrm{S_{i,n}} = \mathrm{High}\}$$
$$\mathrm{Rule}_2 : \mathrm{E_j} \leftrightarrow \{\mathrm{S_{j,1}} = \mathrm{High}, \mathrm{S_{j,2}} = \mathrm{High}, \ldots, \mathrm{S_{j,m}} = \mathrm{High}\}$$
$$\{\mathrm{S_{j,1}}, \mathrm{S_{j,2}}, \ldots, \mathrm{S_{j,m}}\} \subseteq \{\mathrm{S_{i,1}}, \mathrm{S_{i,2}}, \ldots, \mathrm{S_{i,n}}\}$$

对于ML-EVS，如果Ej的指纹模型预测Ej已经在Ei物理发生时发生，则会发生这种攻击。

5.2. Fingerprint Concealment

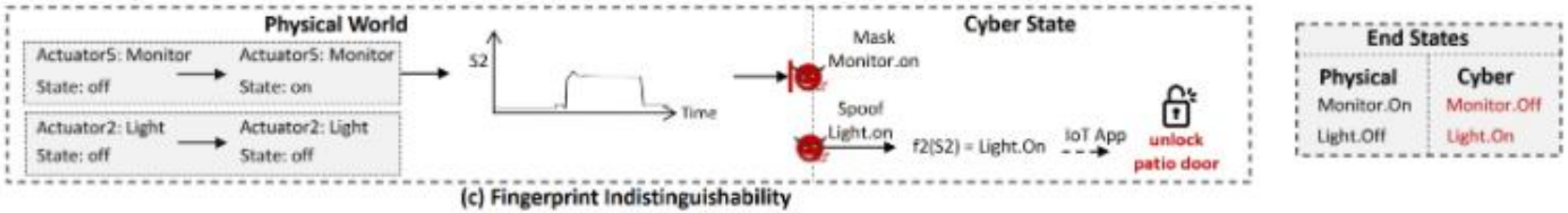


(b) Fingerprint Concealment

如果R-EVS中有两个规则，其中这两个规则对相同的sensor产生相反的影响。当对手掩盖这些事件的任何通知时，R-EVS无法检测到，因为规则中的预期传感器值没有得到满足。虽然可以布置多个传感器来测量事件，但R-EVS通常不会在其规则得到部分满足时发出警报，以尽量减少误报。

在ML-EVS中，由于两个事件的影响相互抵消，则可能会产生对一个或者两个的事件都预测的是 `event-not-occured` ，此时就会存在威胁。

5.3. Fingerprint Indistinguishability



类似于第一种攻击方式，此时二者影响完全相同，此时R-EVS无法区分两个events。

$$\mathrm{Rule}_1 : \mathrm{E_i} \leftrightarrow \{\mathrm{S_{i,1}} = \mathrm{High}, \mathrm{S_{i,2}} = \mathrm{High}, \ldots, \mathrm{S_{i,n}} = \mathrm{High}\}$$
$$\mathrm{Rule}_2 : \mathrm{E_j} \leftrightarrow \{\mathrm{S_{j,1}} = \mathrm{High}, \mathrm{S_{j,2}} = \mathrm{High}, \ldots, \mathrm{S_{j,m}} = \mathrm{High}\}$$
$$\{\mathrm{S_{j,1}}, \mathrm{S_{j,2}}, \ldots, \mathrm{S_{j,m}}\} = \{\mathrm{S_{i,1}}, \mathrm{S_{i,2}}, \ldots, \mathrm{S_{i,n}}\}$$

对于ML-EVS也类似。

Threat Model

攻击者的目标是通过欺骗和掩盖事件而不被发现来逃避EVS。为此，作者假设攻击者可以被动地嗅探智能家居通信，而不拦截或注入任何信息。攻击者可以通过现有的物联网网络分析工具，通过未加密或加密的设备通信实时识别事件，这使得攻击者确定实施攻击的时间，最大限度地提高他们躲避EVS的可能性。为了确定要欺骗或掩盖某个事件，攻击者可以预测事件对传感器测量的物理影响，并相应地欺骗或掩盖事件。

本文解决方法：打补丁

* Software Patching
* Location Patching

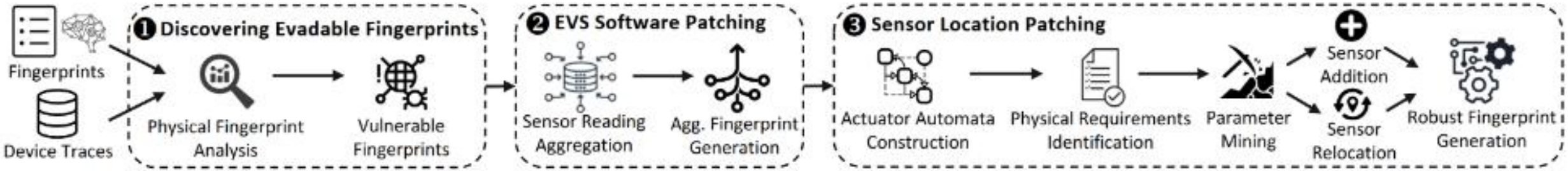


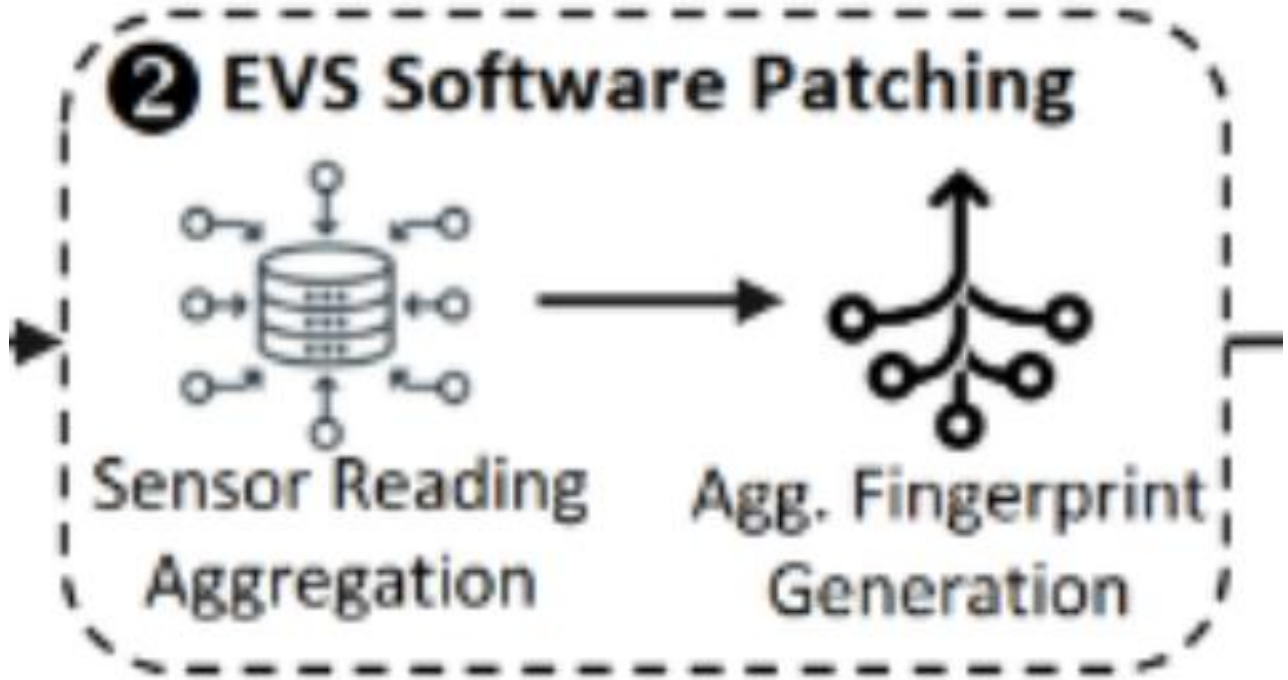
Fig. 3: Overview of physical event fingerprint patching

Algorithm 1 Discovering Evadable Physical Fingerprints

Input: Event List ($\mathtt{L_e}$), Sensor List ($\mathtt{L_s}$), Event Rules ($\mathtt{Rule_e}$), Event Models ($\mathtt{f_e(X)}$ where $\mathtt{X}$ is an event's feature matrix)

Output: Evadable Fingerprints ($\mathtt{L}_{evade}$)

function VULNERABILITYANALYSIS($\mathtt{L_e}, \mathtt{L_s}, \mathtt{Rule_e}, \mathtt{f(e)}$)
  for $\mathtt{E_i} \in \mathtt{L_e}$, $\mathtt{E_{in}} \subset \mathtt{L_e}$ **do**
    if $\mathtt{Rule_{E_i}} \subseteq \mathtt{Agg_r(Rule_{E_{in}})}$ **then** $\mathtt{L}_{evade} \leftarrow \mathtt{L}_{evade} \cup \mathtt{Rule_{E_i}}$
    end if
    if $\mathtt{f_{E_i}(Agg_f(X_{E_{in}})) = 1}$ **then** $\mathtt{L}_{evade} \leftarrow \mathtt{L}_{evade} \cup \mathtt{f_{E_i}}$
    end if
    if $\mathtt{Opp(Rule_{E_i}, Rule_{E_{in}})}$ **then** $\mathtt{L}_{evade} \leftarrow \mathtt{L}_{evade} \cup \mathtt{Rule_{E_i}}$
    end if
    if $\mathtt{f_{E_i}(Agg_f(X_{E_i}, X_{E_{in}})) = 0}$ **then** $\mathtt{L}_{evade} \leftarrow \mathtt{L}_{evade} \cup \mathtt{f_{E_i}}$
    end if
  end for
  return $\mathtt{L}_{evade}$
end function


❷ EVS Software Patching
Sensor Reading Aggregation
Agg. Fingerprint Generation

Software Patching

```
E1 - {S1 = High, S2 = Med}
```

其中S1的High应当是>100，Med应当是[50, 100]

```
1 m1 = sensor1.read()
2 m2 = sensor2.read()
3 if m1 < 100 or (m2 < 50 or m2 > 100):
4     print("Spoofing Attack Detected")
```

```
E1 - {S1 = High, S2 = Med}
```

```
E2 - {S1 = High, S2 = Med, S3 = High}
```

如E1的范围是[100, 150]，E2的范围是[100, 200]，那么他们的聚合范围就是[200, 350]，此时High的范围是[100, 200]，Agg_High是>200。

```
m1 = sensor1.read()
m2 = sensor2.read()
m3 = sensor3.read()
if E2 == 0 and (m1 < 100 or (m2 < 50 or m2 > 100)):
    print("Spoofing Attack Detected")
elif E2 == 1 and (m1 < 200 or m2 < 100 or m3 < 100)
    print("Spoofing Attack Detected")
```

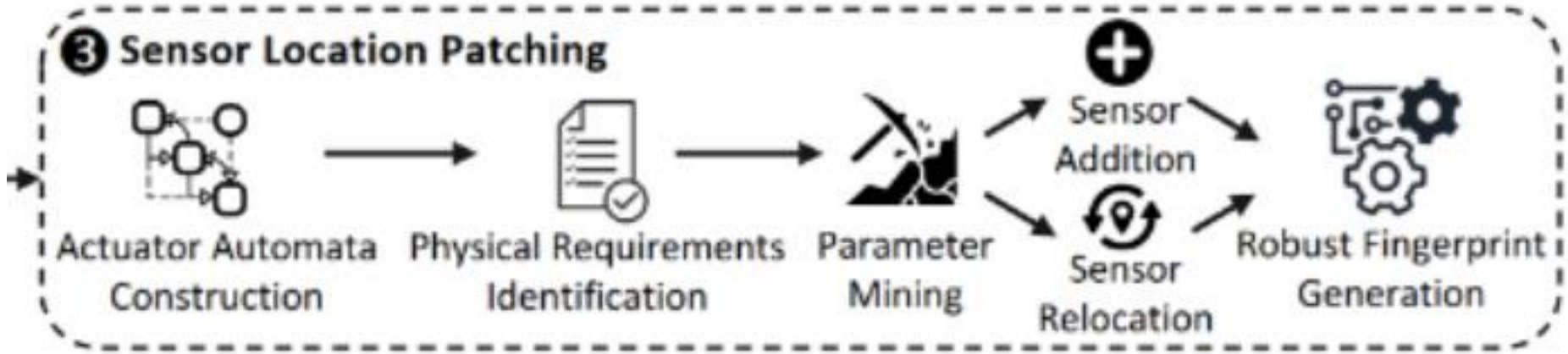
如果E2没发生，按照之前的检测就行；如果E2发生了，将按照聚合后的进行检测。如果两个事件对传感器的测量有相反的影响，则该传感器将从聚合规则中删除，以防止规避。

ML-EVS的想法类似，聚合时送入所有feature就行

Limitations of EVS Software Patching.

这种补丁方法对于Evasion3和一些特殊的Evasion2（两个或更多的events完全隐藏了互相之间的影响）没有作用。

此外，软件补丁不能用于具有布尔类型属性的传感器(例如，声音检测)或传感器饱和(例如，湿度传感器达到100%)。在这种情况下，来自多个事件的聚合影响与事件的单个影响变得难以区分，其中传感器输出布尔值或最小/最大值。


❸ Sensor Location Patching
Actuator Automata Construction
Physical Requirements Identification
Parameter Mining
Sensor Addition
Sensor Relocation
Robust Fingerprint Generation

Actuator Automata Construction：虽然可以通过物理实验的方式确定位置，但是这种需要的工作量太大，并且不方便试验。因此决定采用自动机的方式，将每个事件映射到它们影响的物理通道，并为每个事件构造一个自动机。

1. **Mapping Events to Physical Channels.**建立一个二维表格Map[p, e]，其中Map[p, e]表示事件e会对物理通道p产生影响，反之0表示无影响。

2. **Constructing the Automata.**形式上，自动机表现为 `H = (Q, X, f, ->)` ，Q是执行器的离散状态(例如，开/关)，X是定义事件对通道的影响(例如，温度变化)的连续变量。流量函数(f)有三个参数:到执行器的距离、最小/最大输出值和描述执行器特性(例如，其光强)的设备属性参数，并随时间计算X。每个物理通道的流函数都是惟一的。对于连续的影响(如温度)，用微分方程来定义;对于瞬时的影响(如声音)，用代数方程来定义。

参数设置：

- 首先，我们为距离参数设置不同的值，以测试不同可能的传感器位置。
- 其次，根据传感器数据表输出的最高和最低测量值设置最小/最大输出参数，以处理传感器饱和，并确保自动机不输出超出这些范围的值。
- 利用(T, E)-closeness理论（用实际设备的trace的timings和values测量自动机的保真度）设置设备的特征参数。

  利用采集到的EVS指纹生成数据来确定设备的属性参数，这些参数能使得(T, E)-closeness计算得到最大值。特别地，利用在设备特征参数二分搜索的方式执行自动机，并收集代表Event对物理通道影响的自动机trace。之后记录能够使得自动机和真实设备trace偏离最小的参数。

Physical Requirements Identification:通过观察发现，为了阻止Evasion attack，sensor必须要放到只会受到一个事件影响的位置。这样才能保证从sensor中提取的fingerprint是唯一的。

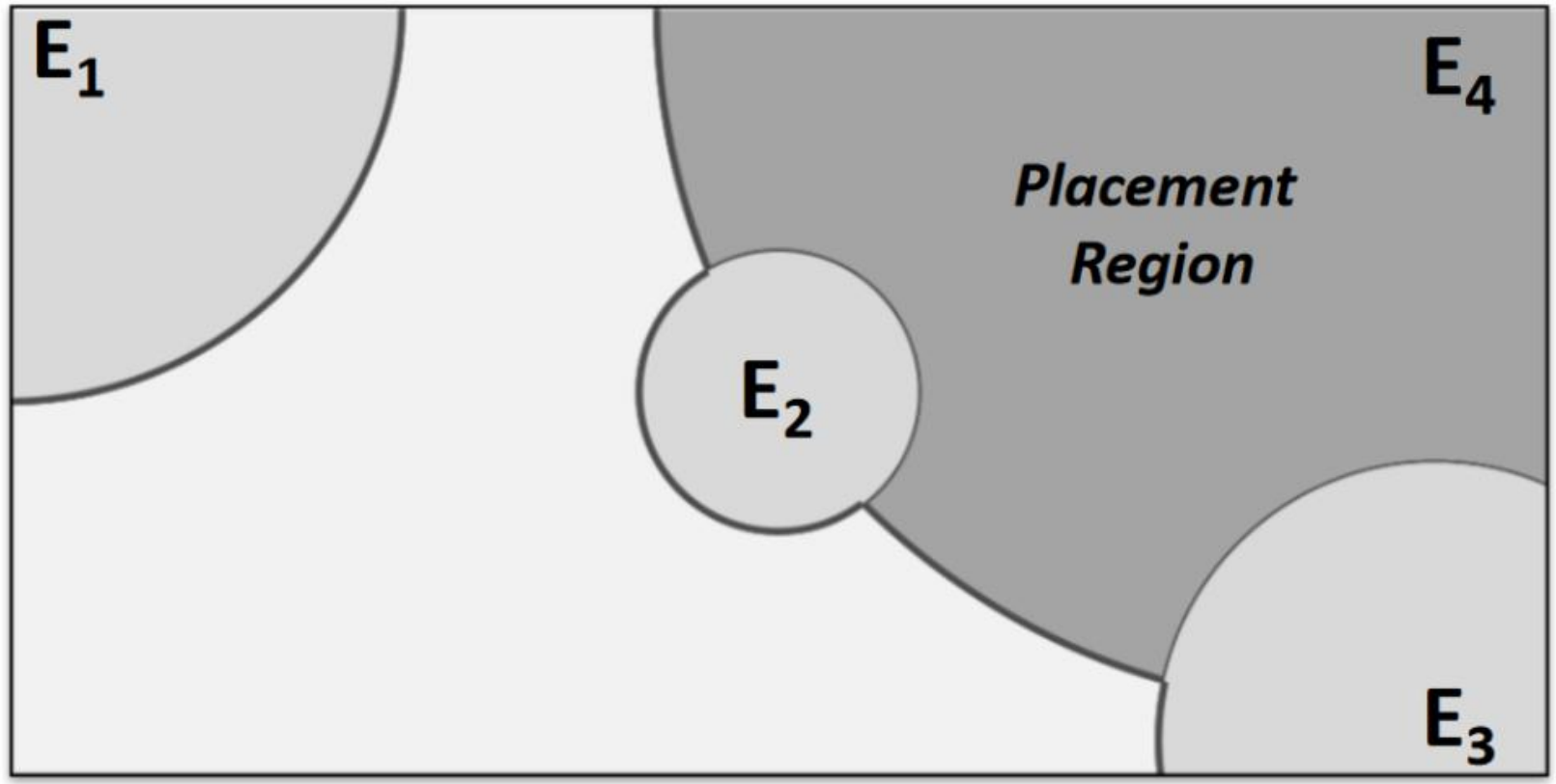
基于上述观察，作者将要求形式化：使影响同一传感器读数的事件数量最小化，同时使检测事件影响的传感器数量最大化。作者用linear temporal logic（LTL）表示要求，以推导出防止躲避攻击的传感器位置。

Algorithm 2 Physical Requirements Formalization

Input: Event List ($\mathtt{L_e}$), Physical Channel List ($\mathtt{L_p}$), Detection Thresholds ($\tau_\mathtt{p}$), Event - Physical Channel Mapping ($\mathtt{Map[p, e]}$)
Output: LTL Constraint List for each model ($\mathtt{L}_{\phi_{\mathtt{p,e}}}$)

1: **function** REQ_MODELING($\mathtt{L_e}, \mathtt{L_p}, \tau_\mathtt{p}, \mathtt{Map[p, e]}$)
2: $\quad \mathtt{L}_{\phi_{\mathtt{p,e}}} = \emptyset$
3: $\quad$ **for** $\mathtt{j} \in \mathtt{L_p}$ **do**
4: $\quad\quad \mathtt{N} = \mathtt{Sum(Map[i, :])}$
5: $\quad\quad$ **for** $\mathtt{i} \in \mathtt{L_e}$ **do**
6: $\quad\quad\quad$ **if** $\mathtt{Map[i, j]} == 0$ **then** Break
7: $\quad\quad\quad$ **else**
8: $\quad\quad\quad\quad \phi_{\mathtt{i,j}} = \Box(\mathtt{imp}(\langle \mathtt{i, j} \rangle) > \tau_\mathtt{i})$
9: $\quad\quad\quad\quad \bar{\phi}_{\mathtt{i,j}} = \Box(\mathtt{imp}(\langle \mathtt{i, j} \rangle) < \tau_\mathtt{i}/(\mathtt{N} - 1))$
10: $\quad\quad\quad$ **end if**
11: $\quad\quad\quad \mathtt{L}_{\phi_{\mathtt{p,e}}} = \mathtt{L}_{\phi_{\mathtt{p,e}}} \cup \phi_{\mathtt{i,j}} \cup \bar{\phi}_{\mathtt{i,j}}$
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: $\quad$ **return** $\mathtt{L}_{\phi_{\mathtt{p,e}}}$
15: **end function**

- **Maximal Influence Formula.**第一个LTL公式（第8行）保证事件e对于物理通道p的影响总是会大于阈值。由于事件对传感器读数的影响取决于距离，因此该公式的满足度取决于传感器和事件之间的距离。阈值定义了一个事件改变传感器读数所需的最小影响。我们根据传感器数据表中定义的灵敏度级别来确定阈值。
- **Minimal Influence Formula.**第二个LTL公式（第9行）可以得到最小的影响区域。考虑到对于同一个物理通道会有多个事件影响，假设有N-1个事件，那么当每一个事件对通道的影响值 < 阈值/(N-1)时，其求和就一定会小于阈值。


E_1
E_4
Placement Region
E_2
E_3

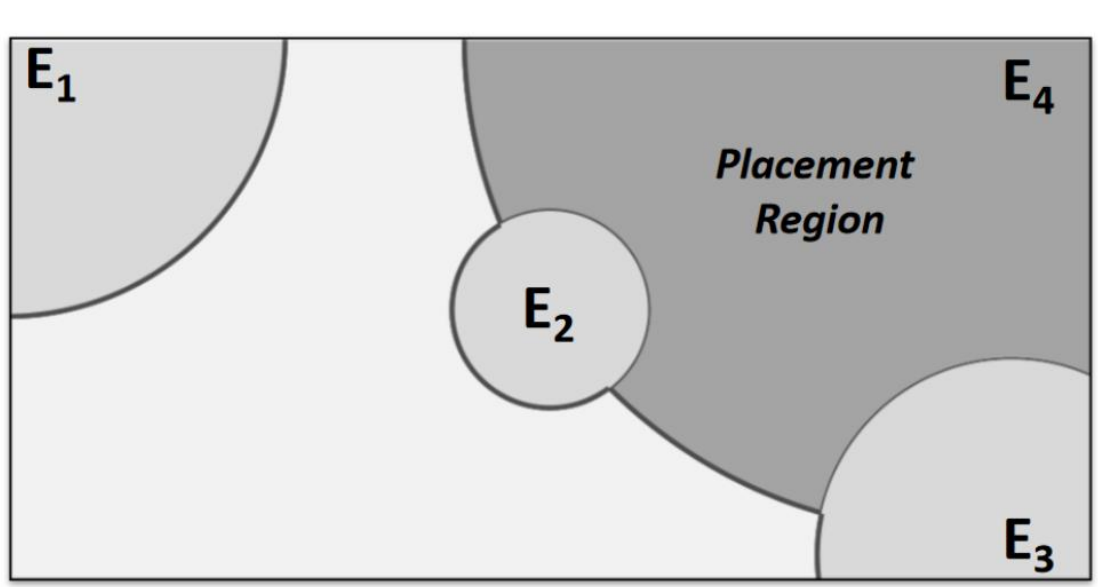
Algorithm 3 Sensor Location Identification

Input: Event List (L_e), Physical Channels (L_p), Actuator Locations ($E_{x,y,z}$), LTL Formulas ($L_{\phi_{p,e}}$), Automata ($H_{p,e}$)
Output: Sensor Location Regions (L_0)

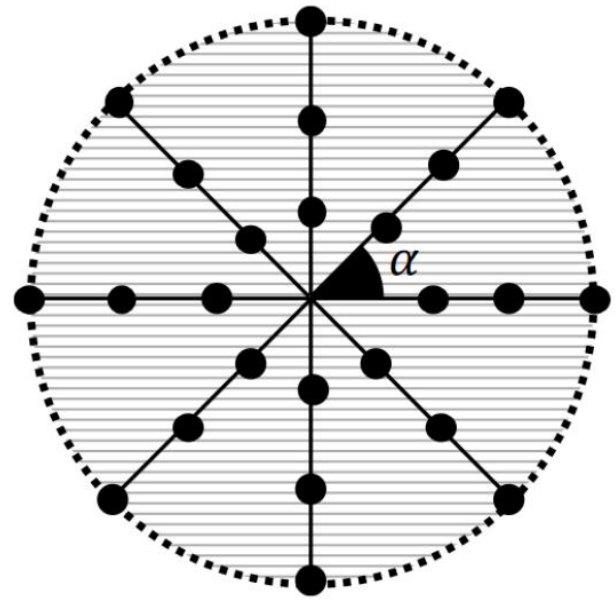
function LOCATIONGENERATION($L_e, L_p, E_{x,y,z}, L_{\phi_{p,e}}, H_{p,e}$)
  $L_0 = \emptyset$
  for $i \in L_p$ **do**
    $ct = 0$
    for $j \in L_e$ **do**
      if $\phi_{i,j} \neq \emptyset$ **then**
        $r_{i,ct} = \text{Parameter\_Mining}(H_{i,j}, \phi_{i,j})$
        $C_{i,ct}: (x - i_x)^2 + (y - i_y)^2 + (z - i_z)^2 = (r_{i,ct})^2$
        $\bar{r}_{i,ct} = \text{Parameter\_Mining}(H_{i,j}, \bar{\phi}_{i,j})$
        $\bar{C}_{i,ct}: (x - i_x)^2 + (y - i_y)^2 + (z - i_z)^2 = (\bar{r}_{i,ct})^2$
        $ct = ct + 1$
      end if
    end for
    for $j = 0 : ct - 1$ **do**
      $Reg = C_{i,j} \setminus \{\bar{C}_{i,j}\}, k = 0 : ct - 1, k \neq j$
      if $Reg = \emptyset$ **then**
        for $Loc \in C_{i,j}$ **do**
          $G = imp(i,j)/(\sum_{k=0}^{k=ct-1,k\neq j}(imp(i,k))$
        end for
        $L_0 = L_0 \cup \langle p, G \rangle$
      else $L_0 = L_0 \cup \langle p, Reg \rangle$
      end if
    end for
  end for
  return L_0
end function

该算法的输入是Actuator的位置、之前生成的自动机结构、LTL公式，输出是sensor应当放到的位置。

为了得到Actuator位置信息，作者使用的是Lumos工具进行定位，该工具使用移动设备传感器和无线信号强度来定位。



(a) Parameter mining output



(b) Grid search

Robest Physical Fingerprint Generation

1. 增加新的sensor。
   优点：允许保持现有传感器的位置不变
   缺点：增加耗费

2. 将之前的sensor重新放置。
   优点：只利用现有传感器的成本效益方法
   缺点：重新定位一个传感器在某些情况可能会导致另一个事件的指纹变脆弱，比如重定位的这个传感器原本是用来区分某个事件的。

在传感器添加/迁移后更新EVS指纹，以防止逃避攻击。
* 首先移除从重新定位的传感器读数中提取的指纹，因为它们会根据新的位置发生变化。
* 接下来修改指纹，以反映添加或重新定位的传感器的预期传感器读数。与EVS指纹提取类似，这需要在事件发生时收集传感器读数。这可以通过进行实验或使用自动机来实现，自动机模拟每个事件对传感器读数的影响。第一种方法（添加设备）使用真实世界的数据来学习新的指纹，但这需要额外的实验。第二种方法（重新定位）不需要实验，而是依靠自动机正确性。