Security Analysis of Emerging Smart Home Applications

新兴智能家居应用的安全分析

2016 IEEE Symposium on Security and Privacy

目录

[Security Analysis of Emerging Smart Home Applications 1](#_Toc38043845)

[新兴智能家居应用的安全分析 1](#_Toc38043846)

[2016 IEEE Symposium on Security and Privacy 1](#_Toc38043847)

[摘要 2](#_Toc38043848)

[背景 2](#_Toc38043849)

[相关概念 2](#_Toc38043850)

[面临的挑战 4](#_Toc38043851)

[针对挑战的解决方案 5](#_Toc38043852)

[亟待解决的问题 5](#_Toc38043853)

[本文贡献 5](#_Toc38043854)

[相关工作 6](#_Toc38043855)

[编程平台背景 6](#_Toc38043856)

[SmartThings背景 6](#_Toc38043857)

[主要工作 8](#_Toc38043858)

[1.过度授权问题 8](#_Toc38043859)

[1.1粗粒度的特权分离机制 8](#_Toc38043860)

[1.2粗SmartApp-SmartDevice 9](#_Toc38043861)

[2.事件对敏感数据的保护不足 9](#_Toc38043862)

[2.1通过基于能力的访问发生泄漏 9](#_Toc38043863)

[2.2通过事件泄漏智能设备标识符 10](#_Toc38043864)

[2.2.1注册事件的方法： 10](#_Toc38043865)

[2.3事件欺骗 10](#_Toc38043866)

[3.第三方集成的不安全性 10](#_Toc38043867)

[4.不安全的Groovy动态方法调用 11](#_Toc38043868)

[4.API 控制 11](#_Toc38043869)

[分析 11](#_Toc38043870)

[1. 数据规模 11](#_Toc38043871)

[2. 完整的功能列表 11](#_Toc38043872)

[3. Groovy代码的静态分析 12](#_Toc38043873)

[4. 粗粒度的功能 12](#_Toc38043874)

[5. 粗粒度的SmartApp-SmartDevice绑定 12](#_Toc38043875)

[6. 越权率 13](#_Toc38043876)

[PCA攻击 13](#_Toc38043877)

[1．后门Pin码注入攻击 13](#_Toc38043878)

[2．门锁密码窥探攻击 14](#_Toc38043879)

[3．禁用休假模式攻击 15](#_Toc38043880)

[4．假警报攻击 15](#_Toc38043881)

[可拓展点 15](#_Toc38043882)

[教训1:非对称设备操作和基于风险的能力 15](#_Toc38043883)

[教训2: 任意事件和身份机制 16](#_Toc38043884)

[合作，审查应用程序商店 16](#_Toc38043885)

[总结 16](#_Toc38043886)

[心得 16](#_Toc38043887)

## 摘要

本文针对于智能家居编程平台进行安全分析，有：

1. 虽然*SmartThings*实现了一个特权分离模型，特权分离模型过于粗粒度，导致明显的过度特权
2. 一旦安装了智能应用程序，它就可以完全访问设备，即使他只需要不放呢权限
3. 事件子系统没有足够的保护携带敏感信息(如锁码)的事件

最后 构建4个PAC攻击以测试猜想。

## 背景

早期的智能家居系统具有陡峭的学习曲线，复杂的设备设置程序，适合于DIY，几家公司推出了新的系统，这些系统易于用户设置，受云支持，用于第三方开发人员构建可实现智能家居收益的应用程序的编程框架。很少或没有先前的研究来调查智能家居应用程序或应用程序本身的编程框架的安全性。（本文工作的突破性）

## 相关概念

**Capability:** 一种名为“*功能*”的特权分离机制，可以指定智能应用程序可能向兼容的智能家居设备发出的操作集

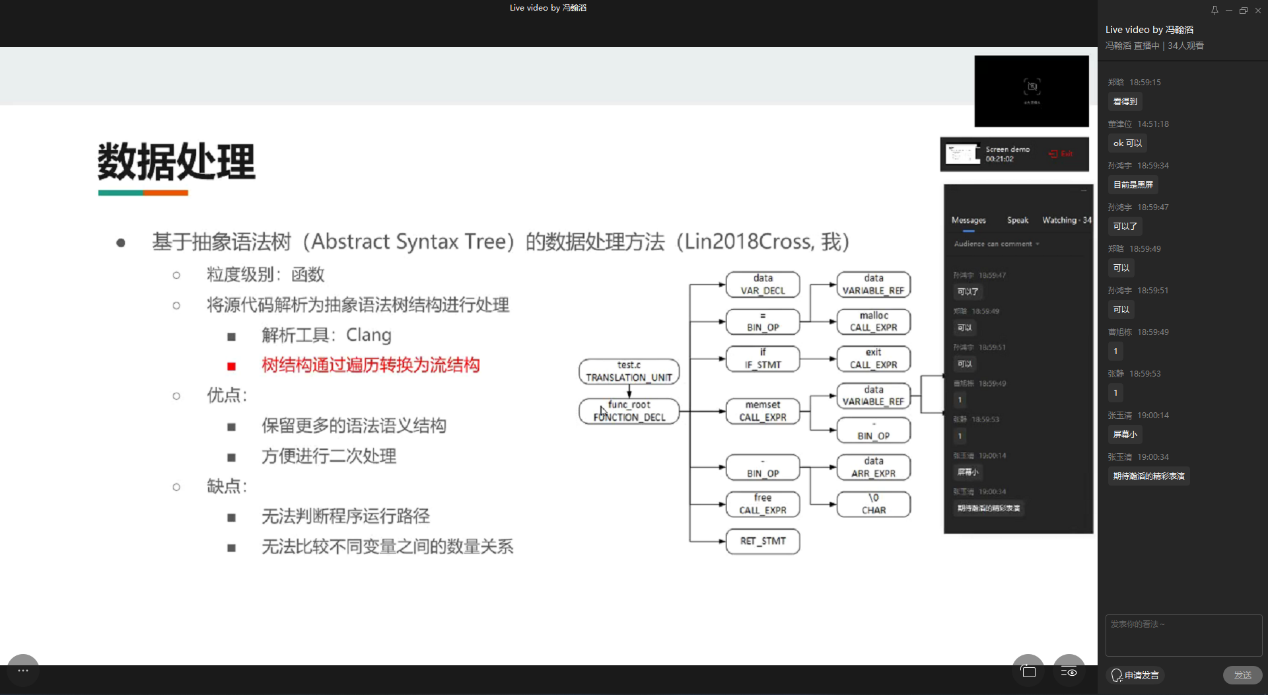
**Groovy：**一种基于*JVM*的敏捷开发语言，融合了*pyrhon*/*Ruby*/*Samaltalk*的许多特性（诸如闭包脚本语言没有类定义的解决方案可以有效地记录局部变量/动态类型等）*Groovy* 代码能够与 *Java* 代码很好地结合，也能用于扩展现有代码。由于其运行在 *JVM* 上的特性，*Groovy*也可以使用其他非Java语言编写的库。

**反射**：*Java*的反射是指程序在运行期可以拿到一个对象的所有信息。

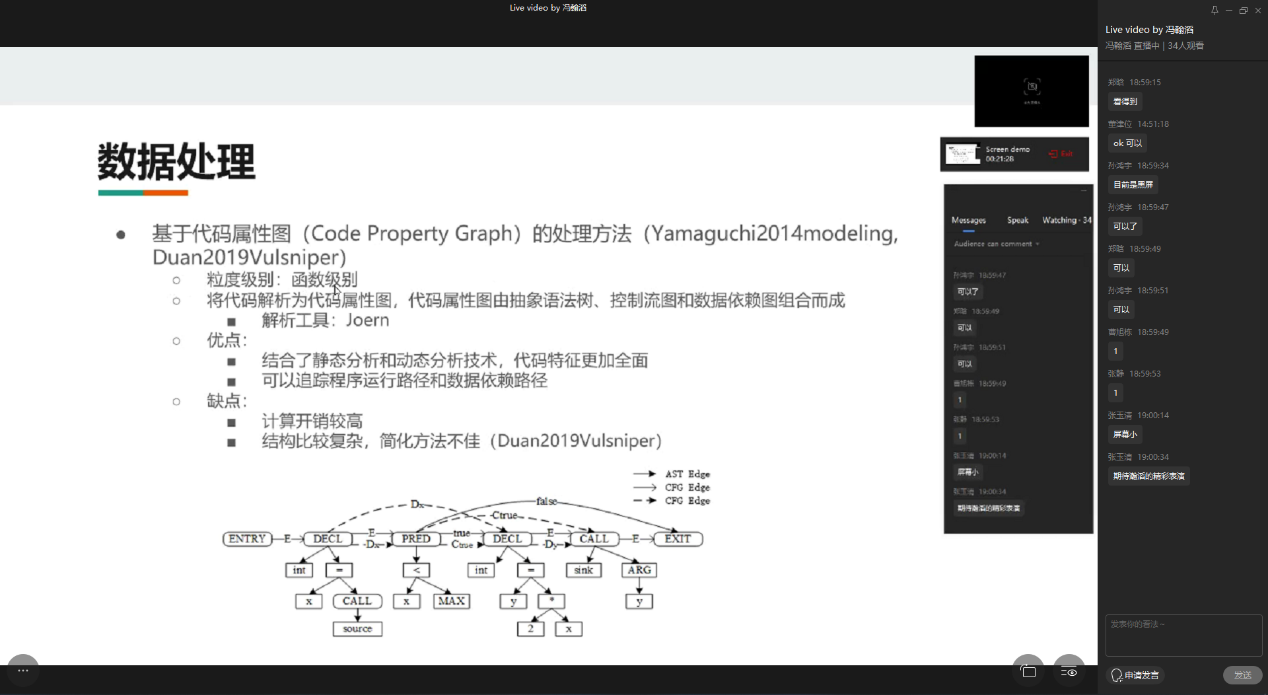
**0Auth 协议:** 为用户资源的授权提供了一个安全的、开放而又简易的标准。0Auth的授权不会使第三方触及到用户的帐号信息（如用户名与密码），即第三方无需使用用户的用户名与密码就可以申请获得该用户资源的授权，因此0Auth是安全的

**静态分析技术：**不执行测试软件，对需求分析说明书。软件著作，源程序结构检查等方法找出漏洞。

**抽象语法树(AST):--目标为了提取代码结构 语法意义文本信息 没有办法**

****

**有语法流程 但是没有比较清晰的控制流图**

****

**结合抽象语法树 数据依赖图+ 控制流图的关系 提取信息更多**

**但是计算开销过大 图结构复杂**

****

**代码切片：**

**基于API调用来丰富**

## 面临的挑战

我们的安全分析探讨了*SmartThings*编程框架的上述面向安全的方面

1. SmartThings平台是一个闭源系统。
2. 此外，SmartApps只在私有的、SmartThings托管的云环境中执行，这使得基于工具的动态分析变得困难。
3. 由于没有公开可用的API来获取SmartApp二进制文件，所以二进制分析技术也不适用

## 针对挑战的解决方案

静态分析工具、运行时测试和手工分析结合起来，同时获取智能APP的源码

## 亟待解决的问题

大量的研究针对于设备本身和通信协议，很少或没有先前的研究来调查智能家居应用程序或应用程序本身的编程框架的安全性。

## 本文贡献

在两个方面发现关键的安全设计缺陷：

* SmartThings功能（特权分离机制）模型
  + 过度授权
* 事件子系统
  + 如果授予该应用程序至少一项设备支持的功能，则该应用程序不需要任何特殊特权即可读取该设备生成的所有事件；
  + 无特权的应用程序可以仅使用一个泄漏的设备标识符读取任何设备的所有事件；
  + 伪造事件

设计4种概念验证攻击：

* 利用SmartAPP从未要求的lockCodes功能
* 假冒自己是个电池监视器，只要求电池的功能，同时窃听事件子系统，从而得到相应的PIN code
* 利用欺骗事件终止休假模式
* 使用一个模拟的设备事件发出假警报，没有特权的应用可以升级它的设备来控制它从没有授权访问的设备

最后总结了构建安全可编程智能家居框架的关键经验

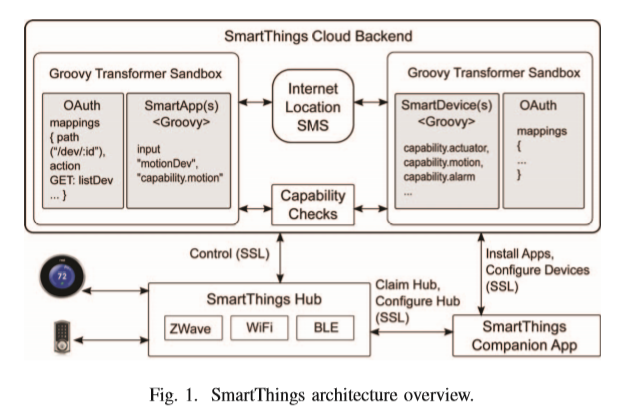
## 相关工作

* Denning等人概述了由于智能设备[12]的快速而稳定的引入而对智能家庭产生的一系列紧急威胁
* 设备方面的研究
  + MyQ车库系统可以变成一个监视工具，在房屋可能空着时通知防盗
  + Wink Relay触摸控制器的麦克风可以打开以窃听对话
  + Tuxedo触摸控制器具有通过身份验证和跨站点请求伪造的认证功能
  + 导致紧凑型荧光灯快速重启，可能会导致癫痫患者癫痫发作
  + 无法共享设备
* 协议方面的研究
  + ZigBee和Z-Wave协议在智能家居设备[9]、[21]上的缺陷
* 类似方法
  + Veracode对包括SmartThings [32]在内的几个智能家居hubs进行了安全性分析，但分析重点在于基础结构保护，例如是否使用SSL / TLS，是否存在重放攻击保护以及是否强密码被使用
* 越权和最小特权
  + 最小特权原则/实践困难性
  + 我们发现，大多数特权不是由于开发人员的困惑，而是由于框架设计本身
* 许可/功能模型设计
  + 引入了使用户保持在循环中的用户驱动的访问控制，当应用程序正在使用敏感资源[24]，[25]

## 编程平台背景

### SmartThings背景

* + 组成：hubs/SmartThings 云后端/智能手机配套应用



* **SmartApps and SmartDevices：**SmartApps和SmartDevices被发布到SmartThings应用商店，可以通过SmartThings伴侣应用程序访问。除了这个主应用程序商店，还有一个辅助商店，开发者可以在那里以源代码的形式使用他们的软件。
  + 在底层，SmartApps并不直接与物理设备通信。相反，它与封装物理设备的SmartDevice实例进行通信。SmartDevice使用较低级别的协议(例如Z-Wave和ZigBee)管理物理设备，并将物理设备暴露给SmartThings生态系统的其余部分
* **Capabilities & Authorization：**一个安全架构，可以管理智能应用程序可以访问哪些设备，我们称之为SmartThings能力模型。功能由一组命令(方法调用)和属性(属性)组成。命令可以控制或驱动设备。属性表示设备的状态信息。
  + **当用户安装smartApp时，这些功能会触发一个设备枚举过程，扫描当前与用户HUB匹配的物理设备，每个输入语句用户都可以看到支持的设备，然后用户选择一个。**
* **Events and Subscriptions：**
  + **订阅事件，当开关被打开触发事件关闭则触发其他**
* **WebService SmartApps:**
  + 可以选择公开Web服务端点，以响应来自外部应用程序的HTTP GET、PUT、POST和DELETE请求
  + 为了保护Web服务端点，云后端提供了基于Oauth的身份验证服务选择提供Web服务的智能应用程序在云后端注册，并发出两个128位的随机值:客户端ID和客户端secret
  + 通过HTTPS进行多步交换后，外部应用程序获得一个限定作用域的OAuth承载令牌，该令牌授予对特定SmartApp的访问权
* **Sandboxing：**
  + 隔离了SmartApps和SmartDevices只允许在Groovy程序中成功地执行白名单方法调用
  + 应用程序不能创建它们自己的类、加载外部jar、执行反射或创建它们自己的线程。每个智能应用程序和智能设备都有一个私人数据存储。

## 主要工作

我们的工作重点是系统地发现和利用SmartThings编程框架设计漏洞

研究了SmartThings框架在五个方面的安全性

* 最小特权原则的坚持
* 事件对敏感数据的保护
* 外部第三方集成安全性
* 外部输入处理：与数据库系统和Web应用程序类似，智能家居应用程序也需要清除不可信的输入 防止命令注入
* 外部通信api的访问控制

### 1.过度授权问题

### 1.1粗粒度的特权分离机制

例如，SmartThings应用商店提供的auto-lock智能应用程序，只需要lock功能的命令capability.lock，但也可以访问解锁命令，从而增加了攻击表面

**方法：**

计算{需求命令和属性} - {使用命令和属性}

### 1.2粗SmartApp-SmartDevice

当用户安装SmartApp时，SmartThings平台会列举所有在app的 preferences部分声明的功能支持的物理设备，用户会选择一组授权给SmartApp的设备。不幸的是，用户不会被告知所请求的功能，只会看到与至少一个所请求功能兼容的设备列表。此外，一旦用户选择要授权给SmartApp使用的设备，SmartApp就可以访问所选设备的device handler实现的所有功能的所有命令和属性. 我们发现开发人员无法避免这种特权，因为这是SmartThings框架设计的结果。开发人员无法避免这个问题

**方法：**

计算{授予的功能}-{使用的功能}

### 2.事件对敏感数据的保护不足

平台支持回调，每个智能设备在与hubs配对时被分配一个128位的设备标识符

* 一旦一个智能应用程序在功能请求后被批准访问某个智能设备，该智能应用程序还可以监控该智能设备发布的任何事件数据
* 一旦智能应用程序获得了一个智能设备的128位标识符，它就可以监控该智能设备的所有事件，而不需要获得设备所支持的任何功能
* 某些事件可以被伪造

### 2.1通过基于能力的访问发生泄漏

一旦用户批准SmartApp请求访问SmartDevice以获得任何支持功能，SmartThings框架就会允许SmartApp订阅SmartDevice的所有活动.但智能设备广泛的使用事件来传递敏感数据。

### 2.2通过事件泄漏智能设备标识符

一旦智能应用程序被授权与智能设备通信，它就可以读取设备。获取128位SmartDevice标识符的id值.

### 2.2.1注册事件的方法：

subscribe(deviceObj, attrString, handler)

deviceObj: 在输入语句执行时注入的设备的引用

attrString: 正在订阅的属性，检查属性的改变以触发事件

handler：事件调用

如果一个应用有智能设备的标识符，那么可以用标识符去替换deviceObj，从而注册一个事件即使它没有被授权与该智能设备对话。但是id标识很容易窃取

### 2.3事件欺骗

**原因:**

没有对引发事件进行访问控制,也没有提供验证事件完整性或事件来源的方法

我们发现攻击者可以使用正确的标识符创建合法的事件对象，并放置任意的状态信息。当这样的事件被触发时，SmartThings会将事件传播到所有订阅的SmartApps，就好像是SmartDevice本身触发了事件一样.

获取标识符很容易，中心和位置ID对所有智能应用程序都是自动可用的

框架提供了一个共享位置对象,那么任何一方都可以修改位置从而触发度假模式下的漏洞. 可以引发受欺骗的位置事件

### 3.第三方集成的不安全性

**概念:**平台可以为第三方应用提供HTTP端点来连接SmartThings。这些WebService智能应用程序可以响应HTTP GET、PUT、POST和DELETE请求.端点通过OAuth协议进行保护，所有远程方在调用WebService SmartApp HTTP端点时必须为每个请求附加一个OAuth承载令牌。

OAuth令牌选择提供WEB服务的智能应用程序在云后端注册, 并发出两个128位的随机值:客户端ID和客户端secret.从而实现Oauth的身份验证服务.

有些应用没有遵循SmartThings的推荐,而是选择了一种更短但不安全的方法硬编码客户端ID和客户端secret

### 4.不安全的Groovy动态方法调用

动态调用 : 考虑一个方法

def foo()

def str = "foo"，

那么可以通过发出"$str"()来调用foo方法,这在内部使用了JVM反射, 因此，动态方法可以方便地为Web服务端点开发处理程序。通常，通过HTTP接收命令的字符串表示形式，并使用动态方法调用直接执行该字符串。

### 4.API 控制

该框架并没有对SmartApps的出站互联网通信设置任何限制

## 分析

### 数据规模

**目的:**由于SmartAPP是在专有的云后端运行,平台二进制文件不会被推送到中心进行本地执行。因此，如果不绕过后台的安全机制，我们就无法获得二进制形式的SmartApps

**方法**:我们发现还有一个副市场,开发者可以托管自己的源码.由此获取代码进行分析. 我们进行网络协议分析发现了一组未发布的REST url，它们与后端交互，检索SmartApps的源代码以供显示.

### 完整的功能列表

智能应用程序定义了64种功能.

**目的**为了获取完整的功能列表,从而进行越权分析

**方法**我们发现了一种方法来查询编译系统一个未发布的REST端点，该端点接受一个设备处理程序ID并返回一个JSON字符串，该字符串列出了设备处理程序实现的一组功能以及所有组成的命令和属性. 因此，我们简单地自动创建64个框架device handle(通过Python脚本)，每个实现一个单独的功能。对于每个自动创建的device handler，我们查询SmartThings后端并接收完整的命令和属性列表。

### Groovy代码的静态分析

**原因:** Groovy极其动态的特性使二进制分析具有挑战性, Groovy编译器将每个直接方法调用转换成反射调用

**方法:** 使用SmartApp的**抽象语法树(AST)**表示来计算过特权. 分阶段执行代码,我们编写了一个自定义程序它访问所有方法调用和属性访问站点，以确定在SmartApp中访问的所有方法和属性。然后，我们使用完整的功能文档过滤这个列表，以获得程序中使用的命令和属性集。

**问题:发现了两个分析错误来源:** 动态方法调用和同名的方法/属性

**解决方案: 1.**动态调用方法检测到时,工具标记一个smartAPP进行手动分析.

**2.**SmartApps中自定义的方法和属性，它们的名称与已知的SmartThings命令和属性相同. 在这些情况下，我们的工具无法区分调用的是实际的命令或属性，还是自定义的方法或属性之一.此时生成标记并在之后进行手动分析。

### 粗粒度的功能

基于前面的功能，我们有276/499个因为粗粒度的功能而被滥用

### 粗粒度的SmartApp-SmartDevice绑定

粗糙的SmartApp-smartdevice绑定越权意味着SmartApp获得了完全未使用的功能，例如门锁/解锁智能应用程序也可以访问门锁的pin code.

但是，我们不知道什么device handler设备句柄将静态地与物理设备相关联, 因此，我们的方法是使用132个设备处理程序的数据集，并尝试不同的关联组合

### 越权率

我们发现，在499个智能应用程序中，有68个(13.6%)使用的命令和属性来自功能，而不是preferences部分显示要求的

## PCA攻击

### 1．后门Pin码注入攻击

从智能应用程序第三方Android对手那里窃取的OAuth访问令牌，命令注入涉及到通过OAuth远程发送命令字符串，以诱导SmartApp执行它在其UI中不支持的操作

**利用：**不安全的Groocy动态方法调用，过度特权和与SmartThings集成的第三方OAuth的不安全实现

**总步骤：** (1)为受害者的SmartThings部署获取OAuth令牌;

(2)确定WebService SmartApp是否使用不安全的Groovy动态方法调用，如果使用，则在OAuth上注入一个适当格式化的命令字符串。

**获取Oauth令牌：**需要获得OAuth令牌的客户端ID和客户端机密被硬编码。使用客户端ID和secret，攻击者可以将OAuth授权URL的redirect\_uri部分替换为由攻击者控制的域来拦截重定向。一般来说，攻击的这一部分包括让受害者点击一个链接，该链接指向真正的SmartThings域，只是链接的redirect\_uri部分被攻击者控制的域所取代。此时，攻击者可以使用从第三方应用程序的字节码获得的码字、客户端ID和密钥对独立地完成OAuth流。

**注入命令：**

**子步骤： （1）**确定与第三方Android应用程序关联的WebService SmartApp是否使用Groovy动态方法调用

**（2）**确定激活SmartApp端点所需的命令字符串的格式。

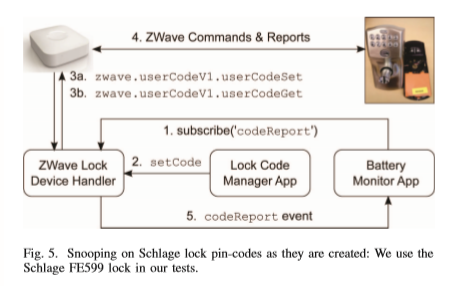
**确定是否使用Groovy动态方法调用比较困难**

### 2．门锁密码窥探攻击

这种攻击使用了一款电池监控器智能应用程序，该程序在源代码级别隐藏了其恶意意图。

我们发现，一旦Device handler从中心获得成功的确认，它将创建一个包含各种数据项的codeReport事件对象。其中之一是刚刚创建的明文pin码。

因此，我们所需要做的就是让我们的电池监控器SmartApp注册所有它被授权访问的设备上的所有类型的codeReport事件。在接收到一个特定的事件后，我们的电池监控器会在事件数据中搜索一个特定的项，该项标识锁代码。



**如上图所示的步骤 清晰明确**

**原因分析：1.**由于粗糙的SmartApp-SmartDevice绑定而导致的特权过度。即使电池监视器SmartApp看起来是良性的，并且只要求电池功能，它也获得了对其他功能的授权，因为相应的Z-Wave锁device handler支持lock, lockCodes, and refresh等其他功能。

**2.** smart things提供的device handler将明文pin码放入事件数据中，任何授权与处理程序通信的smartthings都可以访问这些数据。

**方法：**通过使用Groovy动态方法调用，我们隐藏了SmartApp中的恶意代码片段，让它看起来像是SmartApp将电池电量发送给一个提供图表和历史记录功能的远程服务。根据攻击者控制的Web服务接收到的字符串的值，电池监控器应用程序可以读取电池级别并将其发送到远程服务，也可以窥探锁定pin码并通过SMS将其发送给攻击者

**使用事件泄露加强攻击：**使用事件泄露来获取128位设备标识符

**即使攻击者没有功能，他也可以使用128位的设备标识符来实现上诉攻击**

### 3．禁用休假模式攻击

**休假模式：** 一种流行的家庭自动化体验，它模拟了关灯和其他设备，让房子看起来像有人住一样，而实际上是空的。

**运行：**APP的依赖于location对象的mode属性。当mdoe被设置为一个想要的值时，一个事件触发，智能应用程序激活它的模拟占用

SmartThings在send Location Event API周围没有任何安全控制，可能引起一个错误的模式改变事件。攻击智能应用程序会干扰入住模拟智能应用程序，并使其停止，从而使度假模式设置的保护功能失效

### 4．假警报攻击

一个没有特权的智能应用程序如何使用欺骗的物理设备事件来升级它的特权和并进一步的控制它没有被授权访问的设备。为CO检测器引发一个虚假物理设备事件，导致警报发出，从而使得无特权的攻击智能应用程序滥用了良性警报面板应用程序的逻辑

## 可拓展点

### 教训1:非对称设备操作和基于风险的能力

一种更安全的设计是根据设备操作的风险对功能相似的设备操作进行分组。然而，评估风险是一个挑战，一个开关操作对灯泡比相同的操作对警报风险更小。

### 教训2: 任意事件和身份机制

**事件子系统存在的问题：**SmartApps无法验证事件来源的身份，SmartThings也没有选择性传播敏感事件数据的方法。

### 合作，审查应用程序商店

## 总结

本文对流行的可编程智能家居SmartThings框架进行了安全评估。发现风险有，SmartThings提供的粗粒度功能；SmartThings授予SmartApp对设备的完全访问权；SmartThings事件子系统没有足够的安全控制。基于以上的漏洞，本文给出了4种攻击。

## 心得

智能家居编程平台的确是与众不同的切入角度，值得进一步深挖，但是为了支撑所需要的技术水准要求很高，需要对漏洞分析等有比较深入的学习，同时仍应该利用一些web传统的固有的安全漏洞来扩展自己的攻击面。