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](#_Toc37940985)

[新兴智能家居应用的安全分析 1](#_Toc37940986)

[2016 IEEE Symposium on Security and Privacy 1](#_Toc37940987)

[摘要 2](#_Toc37940988)

[一、介绍 3](#_Toc37940989)

[我们的贡献 5](#_Toc37940990)

[第二章、相关工作 6](#_Toc37940991)

[智能家居安全. 6](#_Toc37940992)

[超过特权和最低特权 7](#_Toc37940993)

[许可/功能模型设计 8](#_Toc37940994)

[第三章SmartThings的背景和威胁模型 8](#_Toc37940995)

[A.SmartThings 背景 8](#_Toc37940996)

[B.威胁模型 15](#_Toc37940997)

[四、SMARTTHINGS框架的安全分析 16](#_Toc37940998)

[1). 最小特权原则坚持 16](#_Toc37940999)

[2).敏感事件数据保护 16](#_Toc37941000)

[3).外部、第三方集成安全 16](#_Toc37941001)

[4).外部输入处理: 16](#_Toc37941002)

[5).外部通信api的访问控制: 17](#_Toc37941003)

[A. 智能应用程序中出现的过度特权 17](#_Toc37941004)

[粗粒度的功能（特权分离机制）： 17](#_Toc37941005)

[粗SmartApp-SmartDevice绑定 18](#_Toc37941006)

[B. 敏感事件数据保护不足 19](#_Toc37941007)

[通过基于能力的访问发生泄漏 19](#_Toc37941008)

[通过事件泄漏智能设备标识符 20](#_Toc37941009)

[事件欺骗 20](#_Toc37941010)

[C. 第三方集成的不安全性 21](#_Toc37941011)

[D. 不安全地使用Groovy动态方法调用 22](#_Toc37941012)

[E. API访问控制:不受限制的通信能力 22](#_Toc37941013)

[五、智能应用的经验安全分析 23](#_Toc37941014)

[A. 我们数据集的总体统计信息 23](#_Toc37941015)

[B. 越权测量 24](#_Toc37941016)

[功能完整列表 24](#_Toc37941017)

[Groovy代码的静态分析 25](#_Toc37941018)

[粗粒度的功能 26](#_Toc37941019)

[粗SmartApp-SmartDevice绑定 26](#_Toc37941020)

[C. 越权使用患病率 27](#_Toc37941021)

[六、概念验证攻击PCA 27](#_Toc37941022)

[A. 后门Pin码注入攻击 28](#_Toc37941023)

[窃取一个Oauth 令牌 29](#_Toc37941024)

[注入命令来利用过度特权 30](#_Toc37941025)

[B. 门锁密码窥探攻击 31](#_Toc37941026)

[从任何设备泄漏事件 33](#_Toc37941027)

[C. 禁用休假模式攻击 34](#_Toc37941028)

[D. 假警报的攻击 34](#_Toc37941029)

[E. SmartThings用户调查研究 35](#_Toc37941030)

[七、挑战和机遇 38](#_Toc37941031)

[教训1:非对称设备操作和基于风险的能力 38](#_Toc37941032)

[教训2:任意事件和身份机制 39](#_Toc37941033)

[合作，审查应用程序商店 39](#_Toc37941034)

[八、总结 40](#_Toc37941035)

[信息披露和响应 40](#_Toc37941036)

[附录 41](#_Toc37941037)

# 摘要

最近，几个相互竞争的支持第三方应用程序开发的智能家居编程框架出现了。这些框架为用户提供了切实的好处，但也可能使用户面临重大的安全风险。本文首次对此类新兴的智能家居编程平台进行了深入的实证安全分析（目标：智能家居编程平台）。我们分析了三星拥有的*SmartThings*，它拥有目前最多的应用软件，支持多种设备，包括运动传感器、火灾警报和门锁。*SmartThings*将应用程序运行时托管在一个专有的、封闭源代码的云后端上，这让审查变得很有挑战性。（问题面临的挑战）我们通过对499个*SmartThings*应用程序(称为*SmartApps*)和132个设备处理程序的静态源代码分析（静态分析技术：是指不执行被测软件，可对需求分析说明书、软件设计说明书、源程序做结构检查、流程分析、符号执行来找出软件错误），克服了这一挑战，并精心设计了测试用例，揭示了该平台的许多未文档化的特性。我们的主要发现有两个方面。首先，虽然*SmartThings*实现了一个特权分离模型，但我们发现了两个固有的设计缺陷，它们导致了*SmartApps*中明显的过度特权。我们的分析显示，由于功能过于粗粒度，商店中超过55%的*SmartApps*被过度授权。此外，一旦安装了智能应用程序，它就可以完全访问设备，即使是它只需要访问有限的设备。其次，*SmartThings*事件子系统(设备使用它通过事件与*SmartApps*进行异步通信)没有足够的保护携带敏感信息(如锁码)的事件。利用框架设计缺陷，构建了四种概念验证攻击:(1)秘密植入门锁代码;(2)盗窃现有门锁代码;(3)居家残疾人度假模式;(4)伪造火警信号。最后，我们总结了新兴智能家居编程框架设计的安全经验。

# 一、介绍

智能家居技术已经超越了基本的便利功能，比如自动控制的电灯和开门器，为人们提供了一些切实的好处。例如，水流传感器和智能电表用于节能。支持*ip*的摄像头、运动传感器和连接的门锁为家庭安全提供了更好的控制。然而，攻击者可以操纵智能设备，造成身体、财务和心理上的伤害。例如，窃贼可以锁定一个连接的门锁，植入隐藏的门禁码;纵火者可以锁定一个智能烤箱，在受害者家中引起火灾.

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，并且仅限于自己动手的爱好者1最近，几家公司推出了更新的系统，这些系统更易于用户设置，受云支持并提供 一个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。 三星的SmartThings [27]，苹果的HomeKit [7]，Vera Control的Vera3 [1]，谷歌的Weave / Brillo [18]和AllSeen Alliance的2AllJoyn [3]是几个例子。

1.存在许多论坛供人们交流专门知识，例如http：// forum。 Universal-devices.com/。

我们提出的问题如下：易受攻击的新兴，可编程，智能家居是以什么方式为什么出现？这些攻击带来了什么？ 解决这个问题至关重要，因为在可编程智能家居普及之前，这个问题的答案将启动并引导对防御系统的研究。 在单个高性能智能家居设备[17]，[19]和在这些设备之间运行的协议（例如Z-Wave和ZigBee [9]，[21]）中发现了漏洞。 但是，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）

据我们所知，我们对智能家居编程框架进行了第一次安全分析。 具体来说，我们根据经验评估一种流行的智能家居可编程框架-Samsung SmartThings的安全性设计。 我们专注于编程框架，因为它是统一应用程序，协议和设备以实现智能家居效益的基础。 攻击者可以远程和秘密地瞄准框架中的设计漏洞，以实现前面概述的紧急威胁。

我们选择SmartThings有几个原因。首先，在撰写本文时，SmartThings有越来越多的应用程序，其中有521个叫做SmartApps，排名第二的是Vera，它在iOS store[1]上有204个基于lua的应用程序。其他与之竞争的框架，如HomeKit、Weave/Brillo和AllJoyn都处于形成阶段，各自的应用程序还不到50个。（应用程序广泛）其次，SmartThings支持132种主要制造商的设备类型。（支持丰富设备）第三，SmartThings与其他框架共享关键的安全设计原则（与其他框架共享关键的安全设计原则）。对设备访问的授权和认证是确保智能家居应用程序平台安全的关键，SmartThings有一个内置的机制，通过所谓的功能来保护设备操作不受第三方应用程序的影响。事件驱动处理在智能家居应用程序[30]中很常见，SmartThings为应用程序提供了注册回调的方法为设备生成的给定事件流。其他平台也支持事件驱动处理。例如，AllJoyn支持总线信号[2]，而HomeKit提供了特征通知API[6]。因此，我们相信通过对SmartThings框架的分析所获得的经验教训，将为许多可编程智能家居框架在早期设计阶段的安全关键组件的设计提供参考。

*SmartThings*框架认识到了潜在的安全漏洞，并整合了一些安全措施。*SmartThings*有一种名为“*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。SmartApps提供了安全存储，仅该应用程序本身可以访问。开发人员在Groovy（Groovy 一种java 框架？）的面向安全的子集中编写SmartApp。。基于Groovy的应用程序在沙箱中运行，该沙箱拒绝诸如反射，外部JAR和系统API之类的操作。（反射：运行时期获得class信息）OAuth协议(OAUTH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。与以往的授权方式不同之处是OAUTH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OAUTH是安全的. (1). 简单：不管是OAUTH服务提供者还是应用开发者，都很易于理解与使用；(2). 安全：没有涉及到用户密钥等信息，更安全更灵活；(3). 开放：任何服务提供商都可以实现OAUTH，任何软件开发商都可以使用OAUTH；)保护与*Smart Apps*的第三方集成。SmartThings为SmartApp和设备处理程序提供了*功能* 受保护的事件子系统，以进行异步通信。

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。执行安全分析很有挑战性，因为*SmartThings*平台是一个闭源系统。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。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。（面临的挑战）

为了克服这些挑战，我们将构建的静态分析工具、运行时测试和手工分析结合起来，对499个Smart Apps和132个设备处理程序的数据集进行了分析，这些处理程序都是以源代码的形式下载的。我们的分析工具可以在https://iotsecurity.eecs.umich.edu找到。（解决方案）

## 我们的贡献

我们在两个方面发现了安全关键的设计缺陷:SmartThings功能（特权分离机制）模型和事件子系统。（缺陷）(设备使用它通过事件与*SmartApps*进行异步通信)

我们发现SmartApps明显被过度授权了:(a) 55%的SmartApps没有使用它们的功能所隐式要求的的所有设备操作权限;(b) 42%的SmartApps被授予了没有明确要求或使用的功能。在许多情况下，由于能力模型的设备级授权设计，不可避免地出现了特权过高的情况，而特权并非由开发人员的过错造成的（§IV-A，§V-B）。 令人担忧的是，我们已经观察到68个现有的SmartApp已在不要求相关功能的情况下利用超权限来提供额外的功能。

我们研究了SmartThings事件子系统，发现：

（a）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；

（b）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；

（c）可以伪造事件（第IV-B节）。

我们利用设计缺陷和框架引发的开发人员错误的组合来展示各种安全问题如何共同削弱家庭安全。 我们构造了四种概念验证攻击：

* 我们远程利用了app store上现有的SmartApp，将后门pin码编程到一个相连的门锁上(VI-A节)。我们的攻击利用了SmartApp从未要求过的lockCodes功能-由于SmartThings功能模型的设计，SmartApp自动享有特权。（Fuzzing 测试）
* 当用户正在对一个Schlage智能锁的pin码进行编程时，我们窃听事件子系统来窥探该锁的pin码，并使用不受限制的smart things提供的SMS API来泄漏它们。我们的攻击智能应用程序宣传自己是一个电池监视器，只要求电池监测功能。（假冒）
* 我们使用一个欺骗事件来停止休假模式模拟(VI-C)，从而禁用了app store上现有的休假模式Smart App。这种攻击不需要任何能力。（欺骗事件）
* 我们使用一个模拟的物理设备事件(VI-D节)发出了一个假的火灾警报。一个没有特权的智能应用程序可以升级它的特权来控制它没有授权访问的设备，从而滥用那些无害的智能应用程序。

以上所有的攻击都会使一个家庭遭受重大的伤害:入室盗窃、盗窃、提供错误信息和破坏公物。这些攻击向量不是特定于特定的设备，而是广泛适用的。

最后，在我们的前瞻性分析中，我们总结了构建安全可编程智能家居框架的关键经验。我们将这些教训与在构建这样的框架中权衡利弊的探索结合起来。我们的分析表明，尽管一些问题很容易解决，但其他问题需要多种技术的良好平衡，包括设计基于风险的能力和身份机制(VII)。

# 第二章、相关工作

## 智能家居安全.

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。例如，有窃听的威胁和各种智能家居设备的直接危害。Denning等人还讨论了攻击的结构，包括数据销毁、非法物理入侵和侵犯隐私等。我们的工作使其中一些风险具体化，并演示了远程攻击者如何在实践中削弱家庭安全。虽然我们不是第一个认识到现代家居的安全风险，但我们首次研究了新兴智能家居应用程序及其相关编程接口的安全特性。

当前的智能家居安全分析围绕两个主题：**设备和协议**。在设备正面，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；霍尼韦尔（中国）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[17]，[19]。 Oluwafemi等。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[23]。 Ur等。研究了飞利浦Hue照明系统和Kwikset门锁等的访问控制，发现每个系统都提供了孤立的访问控制系统，该系统无法实现基本的用例，例如与其他用户（如儿童和临时工）共享智能设备[ 29]。相比之下，我们研究了智能家居编程平台的新兴应用程序和相关的攻击媒介，它们在很大程度上与家庭中使用的特定设备无关。

在协议方面，研究人员展示了ZigBee和Z-Wave协议在智能家居设备[9]、[21]上的缺陷。利用这些漏洞需要接近目标主机。我们展示了程序框架中的设计缺陷，这些缺陷可以被用于不需要物理访问的内部攻击。此外，我们的远程攻击与所使用的特定协议无关。

Veracode对包括SmartThings [32]在内的几个智能家居集线器进行了安全性分析。 安全分析的重点是基础结构保护，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。Veracode的研究发现，SmartThings hub已正确部署了所有研究过的基础设施安全机制，但集线器上开放的telnet调试接口除外，该接口已被修复。相反，我们对SmartThings平台及其应用程序进行了实证分析，以发现框架设计缺陷。

## 超过特权和最低特权

最小特权原则是众所周知的，应设计编程框架以使其更容易实现。 然而，在实践中，很难做到这一点，Felt等人最近在智能手机上的研究证明了这一点。 对Android应用程序进行了市场规模的超权限分析，并确定940个应用程序中的三分之一处于超权限状态[13]，并指出开发人员的困惑是导致超权限Android应用程序的主要因素。 我们的工作类似，只是分析了一个相对封闭的系统，在该系统中，应用程序仅通过专有协议与带有受SSL保护的会话的hubs一起运行在专有云后端上，并在家庭中控制设备。 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

Au等设计了PScout，这是一个用于Android源代码的静态分析框架，可以针对不同的Android版本生成完整的权限说明[8]。我们对SmartApp源代码进行了静态分析，以计算特权。 但是，与PScout不同，我们不能使用静态分析来完成功能文档，因为SmartThings运行时是开源的。 相反，我们依赖于分析SmartThings后端和客户端Web IDE之间的协议。

## 许可/功能模型设计

Roesner等。 引入了使用户保持在循环中的用户驱动的访问控制，此刻应用程序正在使用敏感资源[24]，[25]。 例如，遥控门锁应用程序应仅能够响应用户行为来控制门锁。 但是，某些设备类型和应用程序更适合安装时权限。 Felt等。 引入了一套有关何时使用不同类型的权限的准则[14]。 我们的工作评估了SmartThings功能模型在保护敏感设备操作免受恶意或良性但笨拙的SmartApp攻击方面的有效性。 我们需要确定对未来工作的能力授予方式。

# 第三章SmartThings的背景和威胁模型

我们首先描述了SmartThings平台架构，然后讨论了我们的威胁模型。除了开发人员文档外，人们对SmartThings的架构细节知之甚少。因此，我们还将讨论在适当的时候用于揭示SmartThings体系结构方面的分析技术。

## A.SmartThings 背景

SmartThings生态系统由三个主要组件组成: hubs、SmartThings云后端和智能手机配套应用(见图1)。每个hubs由用户购买，支持多种无线协议，包括Z-Wave、ZigBee和WiFi，可以与用户家中的物理设备进行交互（类似小米网关）。用户管理他们的集线器，将设备与集线器关联起来，并使用智能手机伴侣应用程序(称为SmartThings Mobile)从应用商店安装SmartApps。云后端运行SmartApps。云后端还运行*SmartDevices*，后者是用户家中物理设备的软件包装器。配套的应用程序、集线器和后端通过一个受ssl保护的专有协议进行通信。虽然目前还没有关于SmartThings用户群规模的公开统计数据，但作为采用规模的粗略衡量，我们注意到，截至2016年3月，在谷歌Play Store上安装了10万到50万的Android版本的配套应用。

智能应用和智能设备有两种通信方式。首先，SmartApps可以通过方法调用来调用*SmartDevices*上的操作(例如，锁上门锁)。其次，智能应用程序可以订阅智能设备或其他智能应用程序可以生成的事件(MQTT的订阅发布机制)。智能应用程序可以使用SmartThings api发送短信和进行网络通话。智能设备通过专用协议与中心通信。

**1) *SmartApps and SmartDevices:*** 编程框架支持创建SmartApps和SmartDevices，它们是用Groovy3的一个受限子集编写的，而Groovy3是一种编译成Java字节码的语言。由于Smart Apps和Smart Devices是在封闭源代码的云后端执行的，所以Smart Things提供了一个基于web的环境，托管在云后端，用于软件开发。

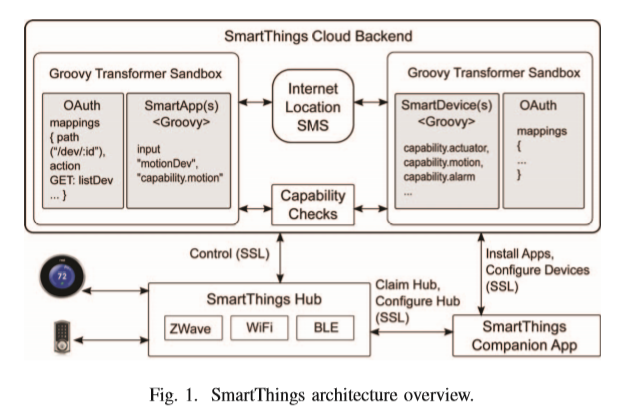
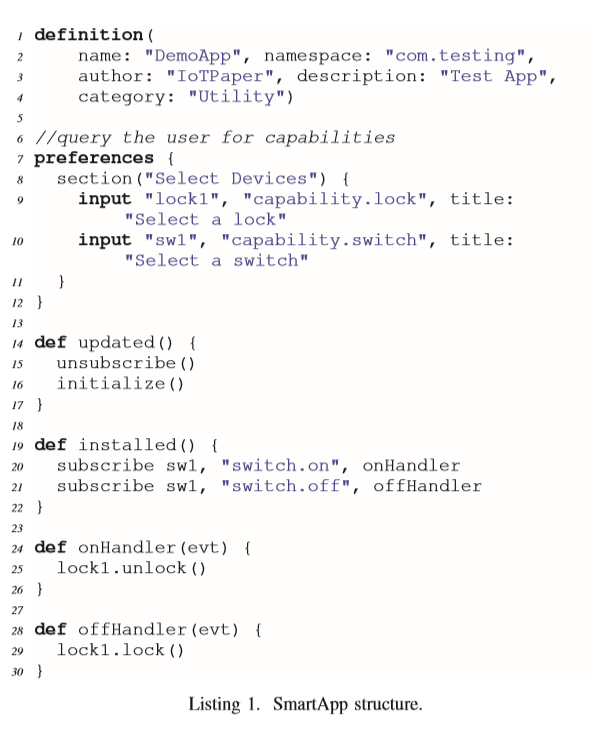


图 1.SmartThings 结构概览

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(图1)。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。（如图1所示）

接下来，我们将解释编程框架的关键概念。清单1显示了一个示例SmartApp，它根据开关的开/关状态来锁定和解锁物理门锁。SmartApp以definition部分开始，定义部分指定元数据，如SmartApp名称、命名空间、作者详细信息和类别



清单1.SmartApp 结构

**2) *Capabilities & Authorization：***SmartThings有一个安全架构，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。可以控制或驱动设备的指令。属性表示设备的状态信息。表I列出了示例功能

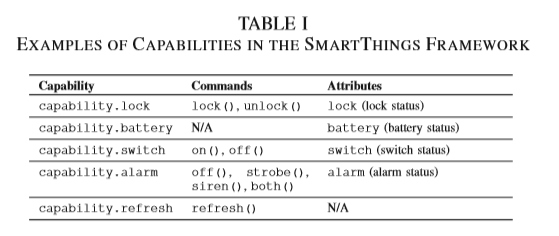
****

表1：SmartThings框架中的功能示例

考虑清单1中的SmartApp。preferences部分有两个输入语句，它们指定了两个capability: capability.lock和capability.switch。当用户安装这个SmartApp时，这些功能会触发一个*device enumeration*过程，该过程会扫描当前与用户hub配对的所有物理设备，对于每个输入语句，用户会看到所有支持指定功能的设备。

对于给定的示例，用户将为每个输入语句选择一个设备，授权SmartApp使用该设备。图2显示了清单1中的示例SmartApp的安装用户界面。



一旦用户为每个输入语句选择一个设备，SmartThings编译器就会将变量lock1和sw1(它们在输入语句中作为字符串列出)分别绑定到所选的锁设备和开关设备上。SmartApp现在被授权通过这两个设备的SmartDevice实例访问它们。

图2所示。安装用户界面和设备枚举:此示例显示应用程序请求设备支持的capability.lock和capability.switch。当用户点击左边屏幕的第一个输入字段时，右边的屏幕就会显示结果。SmartThings列举了所有的锁设备(在这个例子中只有一个)。用户必须选择一个或多个应用程序可以访问的设备。

一个给定的capability可以由多个设备类型支持。图3给出了一个示例。SmartDevice1控制一个Z-Wave锁，SmartDevice2控制一个动作传感器。SmartDevice1支持以下capabilities: capability.lock, capability.battery, and capability.refresh。SmartDevice2支持一个稍微不同的capabilities: capability.motion, capability.battery, and capability.refresh。安装一个电池监测智能应用程序，要求capability.battery将导致用户被要求从一个由Z-Wave锁和运动传感器组成的设备列表中进行选择。输入语句中提供了一个选项，允许将指定的变量绑定到设备列表。如果进行了这样的绑定，一个电池监测智能应用程序就可以监测任意数量设备的电池状态。

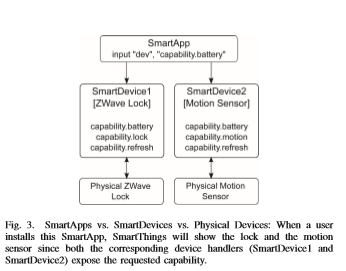


图3所示。SmartApps vs. SmartDevices vs. Physical Devices:当用户安装这个SmartApp时，SmartThings会显示锁和运动传感器，因为相应的设备处理程序(SmartDevice1和SmartDevice2)都会显示请求的功能。

**3)** **Events and Subscriptions：**

首次安装SmartApp时，将调用预定义的已安装方法。在清单1的SmartApp中，installed创建两个事件订阅来切换sw1的状态更新事件(第20、21行)。当开关被打开时，开关SmartDevice引发一个事件，导致函数onHandler执行。该函数解锁与lock1(第25行)对应的物理锁。类似地，当关闭开关时，将调用offHandler函数来锁定与lock1对应的物理锁(第29行)。

**4)** **WebService SmartApps:**

Smart Apps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。HTTP请求触发端点处理程序(由SmartApp指定)，执行开发人员编写的Groovy代码块。

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。SmartApp开发人员通常还会编写外部应用程序来访问SmartApp的Web服务端点。外部应用程序访问智能应用程序需要以下条件:(a)拥有或获取智能应用程序的客户端ID和客户端秘密;(b)将用户重定向至SmartThings网站上受https保护的网页，以使用特定用户的用户ID和密码进行身份验证。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，为其颁发了客户端ID和客户端机密。有关访问Web服务的整个SmartThings身份验证协议的详细信息，请访问http://docs.smartthings.com/en/latest/smartapp - Web -services-developers-guide/overview.html。

**5)****Sandboxing:**

SmartThings云后端使用Kohsuke sandbox技术[20]隔离了SmartApps和SmartDevices。我们使用手动fuzzing来确定这一点，我们建立了测试智能应用程序，尝试未经授权的操作，我们观察到异常的踪迹。Kohsuke sandboxing是一个更大的Groovy源代码转换器类的实现，它只允许在Groovy程序中成功地执行白名单方法调用。例如，如果应用程序发出线程调用，安全监视器会拒绝调用(抛出安全异常)，因为线程不在SmartThings白名单上。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。

总之，从编程的角度来看，智能应用程序、智能设备和功能是关键的构建块。功能定义了一组命令和属性，设备可以支持这些命令和属性，而智能应用程序则声明了它们需要的功能。基于此，用户将智能设备绑定到智能应用程序。

## B.威胁模型

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。任何涉及框架设计缺陷的攻击都在范围之内。我们没有研究试图绕过Groovy运行时环境、on-hub操作系统或云后端基础设施的攻击。这些地方的漏洞可以修补。相比之下，针对设计缺陷的攻击影响更深远，因为一旦有大量应用程序使用该框架，如果不进行重大破坏，就很难更改编程框架。

# 四、SMARTTHINGS框架的安全分析

我们研究了SmartThings框架在五个方面的安全性。我们的方法包括根据对SmartThings架构的研究创建一个潜在安全问题列表，并使用原型SmartApps对每个潜在安全问题进行广泛测试。我们在下面调查每一项调查，并在本节后面详细阐述每一点。

## 1). 最小特权原则坚持

能力模型是否保护设备的敏感操作不受不可靠的、但又有漏洞的应用程序的影响?确保SmartApps只请求他们需要的权限，并且只被授予他们需要的权限，这一点很重要。然而，我们发现许多现有的智能应用程序被过度使用了。

## 2).敏感事件数据保护

提供了哪些访问控制方法来保护设备生成的敏感事件数据不受不可信或有缺陷的智能应用程序的影响?我们发现未经授权的智能应用程序可以窃听敏感事件。

## 3).外部、第三方集成安全

智能应用程序和第三方应用程序是否以安全的方式进行交互?不安全的交互增加了智能家居的攻击面，为远程攻击者打开了通道。像SmartThings这样的智能家居框架应该能够限制第三方安全漏洞造成的损害。我们发现外部平台的开发者bug削弱了SmartThings的系统安全性。

## 4).外部输入处理:

一个WebService智能应用如何保护自己不受不可信的外部输入的影响?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。然而，我们发现SmartApp的端点很容易受到命令注入攻击。

## 5).外部通信api的访问控制:

SmartThings云后端如何限制不可信或善意但漏洞多多的智能应用的外部通信能力?我们发现，互联网接入和短信接入对任何智能应用都是开放的，没有任何手段来控制它们的使用。

## 智能应用程序中出现的过度特权

我们在SmartThings框架中发现了两个关于越权的重要问题，它们都是其功能设计和执行方式的产物。首先，SmartThings框架中的功能是粗粒度的，为设备提供对多个命令和属性的访问。因此，智能应用程序可以获得在设备上调用命令的权限，即使它不使用这些命令。其次，由于SmartThings框架将SmartApp绑定到设备上的方式，SmartApp最终获得的功能可能比它要求的更多。我们将在下面详细讨论这两个问题。

## 粗粒度的功能（特权分离机制）：

在SmartThings框架中，功能定义了一组命令和属性。下面是一个关于capability.lock的小例子：

* 相关命令: lock and unlock
* 相关属性：lock. lock属性具有与命令相同的名称，但该属性指的是已锁定或未锁定的设备状态。

我们对SmartThings体系结构中定义的现有功能的研究表明，许多功能都过于粗粒度化。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，如果智能应用程序被利用。如果使用了lock的命令，SmartApp会把授权的家庭成员锁在门外，造成不便;而如果使用了解锁的命令，SmartApp会让房子很容易被入侵。设备指令的风险通常是不对称的。例如，打开烤箱可能是危险的，但是把它关掉相对来说是安全的。因此，当智能应用程序只需要访问安全命令时，自动授予它对不安全命令的访问权限是不合适的。

为了提供由于功能是粗粒度的而导致的过度特权的简单度量，基于静态分析和手动检查，我们对每个评估的SmartApp计算如下:{需求命令和属性} - {使用命令和属性}。理想情况下，这个集合对于大多数应用程序都是空的。正如在V-B中进一步解释的，超过55%的现有智能应用程序被发现由于功能是粗粒度的而被过度特权化

## 粗SmartApp-SmartDevice绑定

正如在III-A中讨论的，当用户安装SmartApp时，SmartThings平台会列举所有支持在app的 preferences部分声明的功能的物理设备，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设备处理程序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。

更具体地说，SmartDevices提供了对相应物理设备的访问。除了管理物理设备和理解底层协议外，每个SmartDevice还公开一组适合于其管理的设备的功能。例如，默认的Z-Wave锁智能设备支持以下功能：

* capability.actuator,
* capability.lock,
* capability.polling,
* capability.refresh,
* capability.sensor,
* capability.lockCodes,
* capability.battery.

这些功能反映了锁设备操作的各个方面。考虑一个智能应用程序请求该功能的情况。比如电池，用来监控锁电池的状态。SmartThings框架会要求用户授权访问Z-Wave锁设备(因为它符合请求的功能)。不幸的是，如果用户授予了授权请求，SmartApp还可以访问请求的功能以及为Z-Wave锁定义的所有其他功能。（请求其中一个功能，可以访问相当的其他功能）特别是，智能应用程序将能够锁定/解锁ZWave锁，读取它的状态，并设置锁码。

为了提供由于授予了不必要的功能而导致的过度特权的简单度量，基于静态分析和手动检查，我们对每个评估的SmartApp计算如下:{授予的功能}-{使用的功能}。理想情况下，这个集合应该是空的。正如在V-B中进一步解释的那样，超过42%的现有SmartApps被发现由于被授予了额外的功能而被剥夺了特权。在那一节中，我们还将讨论如何保守地计算这个度量。

## 敏感事件数据保护不足

SmartThings支持回调模式，在这种模式下，SmartDevice可以触发充满任意数据的事件，而SmartApps可以注册这些事件。在用户家中，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。之后，设备标识符是稳定的，直到它从hubs中移除或再次配对。因此，128位的设备标识符对于用户的home是惟一的，因为一个home拥有128位的设备标识符在另一个home中是没有用的。尽管如此，我们还是发现了访问事件控制方式中的重大漏洞：

* 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。SmartThings框架没有专门的机制，让智能设备有选择地将事件数据发送给一小部分SmartApps，或者让用户将SmartApp的访问权限限制在一小部分事件上
* 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。特别是，我们发现任何SmartApp或SmartDevice都可以欺骗与位置相关的事件和特定于设备的事件。

## 通过基于能力的访问发生泄漏

如上所述，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。我们发现智能设备广泛使用事件来传递敏感数据。例如，我们发现smartthings提供的Z-Wave锁智能设备传输包含锁pin码的codeReport事件。任何可以访问Z-Wave锁智能设备的智能应用程序(比如，用于监控设备的电池状态)都会自动获得监控其所有事件的能力，并可以使用该访问将事件记录到远程服务器上，并窃取锁pin码。智能手机应用还可以跟踪门锁代码，因为它们被用来进出房屋，因此跟踪家庭成员的移动，可能会导致侵犯隐私。

## 通过事件泄漏智能设备标识符

如上所述，用户家中的每个智能设备都被分配了一个随机的128位标识符。然而，这个标识符并不是隐藏在智能应用程序中。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。智能应用程序通常使用订阅来注册事件:subscribe(deviceObj, attrString, handler)。在这个调用中，deviceObj是SmartThings Groovy编译器在输入语句执行时注入的设备的引用，attrString指定正在订阅其更改的属性或属性，handler是在属性更改事件发生时调用的方法。我们发现，如果一个智能应用程序学习了一个智能设备的设备标识符，它可以在上面的调用中用设备标识符替换deviceObj来注册与该智能设备相关的事件，即使它没有被授权与该智能设备对话。也就是说，拥有授权设备标识符值的它的承载方法可以读取设备处理程序产生的任何事件，而不考虑任何已授予的功能。

不幸的是，设备标识符很容易在智能应用程序之间交换。它不是一个不透明的句柄，也不是特定于一个智能应用程序。目前，SmartThings应用商店中有几个SmartApps，它们允许用户通过OAuth协议远程检索用户家中的设备标识符。我们将讨论一种在六中利用这一弱点的攻击。

## 事件欺骗

SmartThings框架既没有对引发事件进行访问控制，也没有为触发的SmartApps提供验证事件完整性或事件来源的方法。我们发现，一个没有特权的智能应用程序既可以欺骗物理设备事件，也可以欺骗与位置相关的事件。

智能设备检测设备中的物理变化并引发相应的事件。例如，烟雾探测器智能设备在探测到附近的烟雾时将引发烟雾事件。事件对象包含各种状态信息以及位置标识符、中心标识符和作为事件源的128位设备标识符。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。获取设备标识符也相对简单(VI-B)。我们将讨论一种攻击，在这种攻击中，没有特权的智能应用程序升级其特权以控制VI-D中的警报设备。

SmartThings框架提供了一个共享位置对象，该对象表示当前的地理位置，如家庭或办公室。SmartApps可以读写location对象[26]的属性，也可以订阅这些属性的变化。例如，一个家居占用检测器监视一组运动传感器，并相应地更新location对象的模式属性。度假模式应用程序使用mode属性来决定何时开始模拟入住。由于所有SmartApps和SmartDevices都可以访问location对象，所以SmartThings可以灵活地使用它。

然而，我们发现一个智能应用程序可以引发受欺骗的位置事件，并错误地触发所有依赖于location对象属性的智能应用程序。

总之，我们发现SmartThings事件子系统的设计是不安全的。SmartDevices广泛使用它来发布它们的状态和敏感数据，从我们的数据集引发事件的132个设备处理程序中选择111个(见表二)。

## 第三方集成的不安全性

SmartApps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。例如，If-This-Then-That4（动作规则定义网站）可以连接到SmartThings并帮助用户设置触发操作规则。Android、iOS和Windows Phone应用程序可以连接起来，提供简化的管理和规则设置界面。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。

之前的研究已经表明，许多移动应用程序不正确地实现OAuth协议是由于开发人员的误解、混淆OAuth文档以及移动操作系统的限制，这些限制使得OAuth进程不安全。此外，SmartThings OAuth协议的设计要求智能手机应用程序开发人员引入另一层身份验证，以安全地使用SmartThings客户端ID和客户端机密。在对与SmartApps接口的Android应用程序进行了简短的搜索之后，我们在谷歌Play store上发现了一个Android应用程序的实例，它没有遵循SmartThings的推荐，而是选择了一种更短但不安全的方法，即在字节码中嵌入客户端ID和secret（硬编码？）。我们发现其不正确的SmartThings OAuth协议实现可以用来窃取OAuth令牌，然后用来远程利用相关的SmartApp。VI给出了一个我们自己验证过的攻击示例。

## 不安全地使用Groovy动态方法调用

如前所述，WebService SmartApps公开了通过OAuth保护的HTTP端点。OAuth令牌的作用域是特定的SmartApp。但是，开发人员可以自由决定端点集、它们作为输入的数据类型以及端点处理程序的编写方式。

Groovy提供了动态方法调用，方法可以通过将其名称作为字符串参数来调用。考虑一个方法def foo()。如果有一个Groovy字符串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(动态调用，程序可以在运行期间获取class 类相关信息)。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。（类似JS语法 def foo() str=‘foo’命名函数）

使用这一功能的应用程序可能容易受到利用过度特权和欺骗应用程序执行意外操作的攻击。我们将讨论一个示例攻击，该攻击欺骗WebService SmartApp在VI中执行不受支持的操作。这种不安全的设计容易导致命令注入攻击，它类似于众所周知的sql注入攻击。

## API访问控制:不受限制的通信能力

虽然SmartThings框架使用OAuth认证来自外部的来自SmartApps的互联网请求，但该框架并没有对SmartApps的出站互联网通信设置任何限制。此外，SmartApps还可以通过smartthings提供的服务向任意号码发送短信。这样的设计选择允许恶意的智能应用程序滥用这种能力，从受害者家中泄露敏感信息。§6讨论了一个示例攻击。

# 五、智能应用的经验安全分析

为了了解四清单中讨论的安全问题在实践中是如何体现的，我们从SmartThings应用商店下载了499个SmartApps并进行了大规模的分析。我们首先介绍了潜在的易受攻击的应用程序的数量，然后深入研究，确定由于IV-A中讨论的设计缺陷，哪些应用程序被过度使用了。

## 我们数据集的总体统计信息

SmartApps在专有的云后端运行。SmartApp的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。这就排除了只进行二进制分析的可能性，就像过去对智能手机应用程序分析[13]所做的那样。

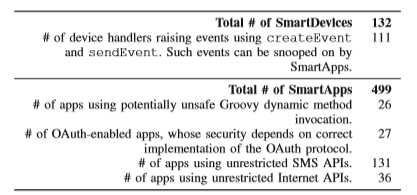
不过，SmartThings支持Web IDE，开发人员可以用Groovy编程语言构建应用程序。Web IDE允许程序员在源代码级别的市场上共享源代码，其他程序员可以浏览这个市场。如果SmartApp开发者选择在这个源代码级别的市场上分享他们的代码，那么这些代码就会被标记为开源的，并且是免费的。用户还可以访问源代码级市场下载和安装应用程序。这个源代码级别的市场可以通过Web IDE访问，但是没有自动下载所有应用程序的选项。

我们的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。我们下载所有可用499 SmartApps 2015年7月的市场使用的未公开REST URL,和另一组URL,我们截获通过SSL中间人代理伙伴应用程序(我们不能下载22个应用,总共521,因为这些应用程序只有在二进制形式,没有已知的REST URL)。同样，我们下载了全部132个唯一的SmartDevices(设备处理程序)。我们注意到，我们本可以访问所有SmartApps和SmartDevices的源代码页面，也可以手动下载源代码。为了方便起见，我们选择了上述的自动化方法。

表II显示了我们的数据集的分解。请注意，并非所有这些应用程序都是脆弱的。表中显示了上界。在VI中，我们选择这些应用程序的一个子集来显示实际的漏洞实例。接下来，我们研究了499个应用程序要求的功能，以衡量智能应用程序在现场部署时的过度特权程度。

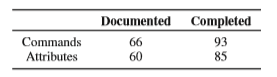
表2

我们的SMARTAPP和SMARTDEVICE数据集的分解



表三

64种SMARTTHINGS功能的命令/属性



## 越权测量

我们首先讨论如何获得包括组成命令和属性在内的完整功能集。然后，我们讨论了为499个基于groovy的智能应用程序计算overprivilege而构建的静态分析工具。

## 功能完整列表

截至2015年7月，智能应用程序定义了64种功能。然而，我们发现仅记录了这些功能的一些命令和属性。我们的越权分析需要一套完整的功能定义。之前的工作使用了二进制检测和自动化测试来观察应用程序的运行时行为，从而推断出与特定功能[13]相关的操作集。但是，这不是我们的选择，因为运行时是在专有后端内部。

为了克服这个挑战，我们分析了SmartThings编译系统，并确定它包含所有功能的信息。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个框架设备处理程序(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设备处理程序，我们查询SmartThings后端并接收完整的命令和属性列表。表III总结了我们的数据集。

## Groovy代码的静态分析

由于SmartApps编译为Java字节码，我们可以使用一个像Soot这样的分析框架来编写一个计算过特权[31]的静态分析。然而，我们发现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（反射：运行时可以获取class info ）。这种反射使现有的二元分析工具，如Soot在很大程度上对我们的目的无效。

相反，我们使用SmartApp的抽象语法树(AST)表示来计算过特权，因为我们有每个应用程序的源代码。Groovy支持用于修改编译过程的编译自定义器。就像LLVM在一个阶段执行程序员写的传递一样，编译定制器可以在编译过程的任何阶段执行。我们的方法使用一个在语义分析阶段之后执行的编译器定制器。我们编写了一个编译自定义程序，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。

为了检查我们的工具的正确性，我们随机挑选了15个SmartApps，并手工调查了源代码。我们发现有两个潜在的分析错误来源:动态方法调用和同名的方法/属性。我们以以下方式修改了我们的分析工具，以适应这些缺陷。

当检测到动态方法调用时，我们的工具标记一个SmartApp进行手动分析。26个智能应用被打上了flag。我们发现这些智能应用, 只有2是越权的。在研究这26个SmartApps时，我们发现其中有20个在WebService处理程序中使用了动态方法调用，其中远程方指定了一个字符串表示要在设备上调用的命令，因此可能会导致命令注入攻击。（方法的改进）

第二个错误来源是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。当我们的工具检测到这种情况时，它会再次发出手动分析标记。七个智能应用程序因此被标记。在检查中，我们发现这七个人都被正确地标记为过度特权。综上所述，由于上述两个误报源，24个app被标记为越权，误报率为4.8%。我们的软件可以通过https://iotsecurity.eecs.umich.edu访问。

## 粗粒度的功能

对于每个智能应用程序，我们会计算请求命令和属性set和使用命令和属性set的差异。一组差异表示智能应用程序可不可以访问命令和属性。表四总结了我们基于499个SmartApps得出的结果。我们发现，在499个智能应用程序中，至少有276个因为粗粒度的功能而被滥用。请注意，我们的分析是保守的，如果不能可靠地确定是否存在过度特权，则选择将SmartApps标记为非过度特权。

## 粗SmartApp-SmartDevice绑定

回想一下，粗糙的SmartApp-smartdevice绑定越权意味着SmartApp获得了完全未使用的功能。考虑一个智能应用程序，它只根据一天的时间来锁门和开门。此外，请考虑门锁是由暴露capability.lock或capability.unlock的设备处理程序操作的。因此，门锁/解锁智能应用程序也可以访问门锁的锁码功能，即使它没有使用锁码capability。我们的目标是计算出一组表现出这种过度特权的智能应用程序。

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，因为在实践中可能有任意数量的device handler设备句柄。我们只知道一个智能应用程序要求一个特定的功能。我们并不确切地知道它由于与特定的设备处理程序关联而获得的功能集。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。

例如，考虑上面相同的门锁/解锁SmartApp。假设它要求capability.imageCapture，这样它就可以拍摄人们进入家中的照片。现在，对于这两种功能，我们必须确定实现这些功能的所有可能的设备处理程序组合。对于每个特定的组合，我们将获得一个越权结果。

在实践中，我们注意到组合的数量非常大(大于数十万的数量级)。因此，我们限制了组合的数量(我们的分析是保守的，代表了越权的一个下界)。我们限制这些组合，使我们只选择在所有可能组合中实现最少功能的设备处理程序。

我们的结果表明，213个智能应用程序表现出了这种超权限(表IV)。这些智能应用程序可以访问智能应用程序明确要求之外的其他功能命令/属性。

## 越权使用患病率

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。这是不可取的，因为它可以将SmartThings锁定为支持越权的特性，而不是纠正overprivilege。随着智能应用程序数量的增长，解决过度特权问题将变得更加困难。理想情况下，SmartApps必须有另一种方式:(1)检查设备支持的额外操作，(2)明确要求这些操作，让用户处于循环中。

请注意，由于粗糙的SmartApp-SmartDevice绑定，这68个SmartApps中的成员仍然可能表现出过度特权。然而，这种情况是否会发生并不影响SmartApp是否会使用额外的功能。使用越权(不应该发生)的智能应用包括：

**Gentle Wake Up：**这款智能应用会慢慢增加灯光亮度，唤醒沉睡的人们。它动态地确定灯光是否支持不同的颜色，并在可能的情况下改变灯光的颜色。这款智能应用程序使用的命令来自它没有要求改变灯光颜色的功能。

**Welcome Home Notiﬁcation：**这款智能应用会打开Sonos播放器，并在门打开时播放音乐。智能应用还可以控制Sonos播放器的电源状态。Sonos智能设备支持capability.musicPlayer ,capability.switch。开发人员依赖SmartThings来访问交换机功能，尽管SmartApp从未明确请求过。如果开发人员也单独请求开关功能，那么在安装过程中就会出现两个相同的设备选择屏幕。

# 六、概念验证攻击PCA

我们展示了四种具体的方法，将四中讨论的各种安全设计缺陷和开发人员bug结合在一起来削弱家庭安全。我们首先提出一个攻击，利用现有的网络服务智能应用程序与偷来的OAuth令牌植入后门pin码到门锁。然后，我们展示了三种攻击:盗窃门锁密码，在假期模式下禁用安全设置，以及使用精心制作的智能应用程序制造假的一氧化碳(CO)警报。表V显示了高级攻击摘要。最后，我们讨论了一项调查研究，我们对22个SmartThings用户进行了关于我们的门锁pin码窥探攻击。我们的调查结果表明，我们的大多数参与者对SmartThings平台的安全和隐私风险了解有限，超过70%的参与者回应说，他们有兴趣安装一个电池监测应用程序，并会让它使用门锁。只有14%的参与者报告说，智能手机的电池监控器可以执行门锁密码窥探攻击。这些结果表明，我们伪装在电池监控器上的pin码窥探攻击并不是不现实的。

## 后门Pin码注入攻击

我们演示了对现有的WebService智能应用程序进行命令注入攻击的可能性，该攻击使用的是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。这种攻击利用了不安全的Groovy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。

为了进行概念验证攻击，我们从谷歌Play Store下载了一个流行的Android app7，用于简化远程设备交互和管理。我们将此应用程序作为第三方应用程序。第三方应用程序请求用户验证到SmartThings，然后授权WebService SmartApp访问各种家庭设备。WebService SmartApp是由第三方应用的开发者编写的。图4显示了第三方应用的截图——该应用允许用户远程锁定和解锁ZWave门锁，并打开和关闭智能电源插座。

7.The app has a rating of 4.7/5.

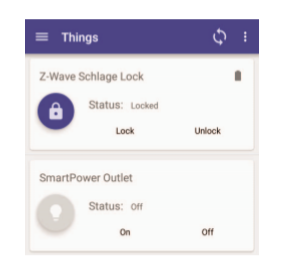


图4第三方Android应用程序，使用OAuth与SmartThings交互，使家庭成员能够远程管理连接的设备。我们故意不命名这个应用程序。

该攻击有两个步骤:(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。

## 窃取一个Oauth 令牌

与Chen等人[10]的研究类似，我们调查了第三方Android应用程序的一个被拆解的二进制文件，发现需要获得OAuth令牌的客户端ID和客户端机密被嵌入到应用程序的字节码中。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（XSS攻击，点击劫持？）。受害者不应该怀疑任何东西，因为URL确实把受害者带到真正的HTTPS登录页面SmartThings。一旦受害者登录到真正的SmartThings Web页面，SmartThings就会用6个字符的码字自动重定向到指定的重定向URI。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。SmartThings的OAuth协议流记录在[28]中。注意，SmartThings提供了OAuth承载令牌，这意味着任何拥有令牌的人都可以访问相应的SmartThings部署。我们强调，窃取OAuth令牌是我们进行攻击的惟一先决条件，我们执行此步骤是为了完整性(附录B提供了其他详细信息)。

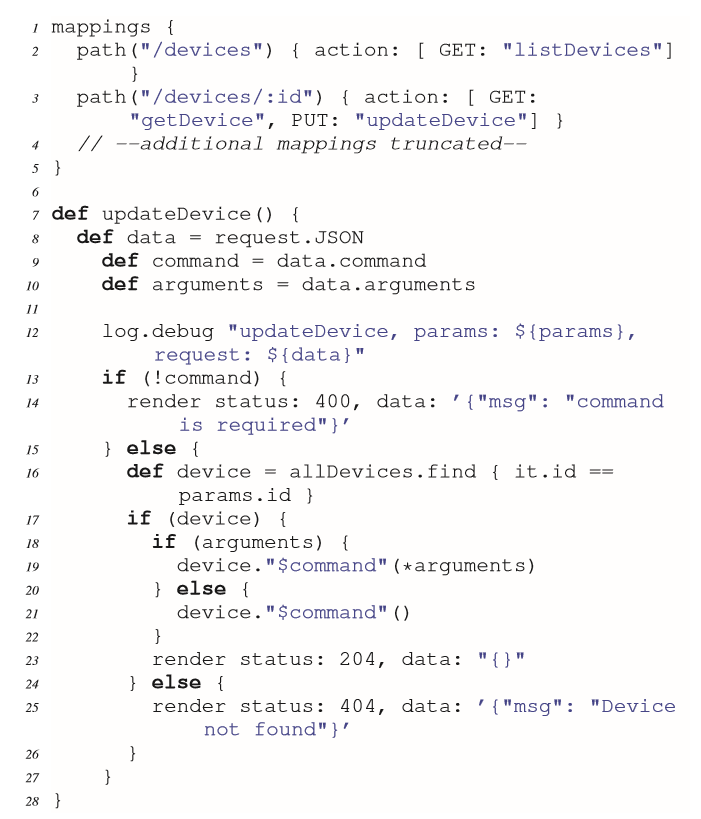
## 注入命令来利用过度特权

攻击的第二部分涉及(a)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，(b)确定激活SmartApp端点所需的命令字符串的格式。

被破解的第三方安卓系统包含了足够多的信息，可以重构WebService SmartApp所期望的命令字符串格式。判断SmartApp是否使用了不安全的Groovy比较困难，因为我们没有源代码。在手动测试setCode操作的命令字符串变体并检查HTTP返回代码以确定该命令是否成功之后，我们确认所有类型的命令(与锁相关)都被接受了。因此，我们通过OAuth向WebService SmartApp传输了一个有效载荷来设置一个新的锁代码。我们证实后门密码被安装在门锁上。我们注意到，我们注入的与利用越权—setCode是capability.lockCodes的成员，由于SmartThings的能力模型设计(参看§IV-A)，该问题中脆弱的SmartApp自动获得的能力。

虽然我们的示例攻击利用了一个二进制的SmartApp，但是出于演示目的，我们在清单2中展示了Logitech Harmony WebService SmartApp的一部分。第19行和第21行很容易受到命令注入的影响，因为“$command”是直接通过HTTP接收的字符串，没有经过消毒。

总而言之，这种攻击使用现有的易受攻击的智能应用程序创建任意的锁代码(实际上是创建通往受害者家的后门)，而实际上智能应用程序只能对门进行锁定和解锁。这种攻击利用越权由于SmartApp- smartdevice的粗绑定、用于Groovy动态方法调用的未消毒字符串，以及与易受攻击的SmartApp一起工作的智能手机应用程序中OAuth协议的不安全实现而导致的过度特权。请注意，攻击者也可以使用破解的Android应用程序直接解锁门锁;但是植入上述后门可以让攻击者持续进入家中，即使Android应用程序被打了补丁，或者用户中心离线。



清单2. Portion of the Logitech Harmony WebService SmartApp的源代码。mappings部分列出了所有端点。第19和21行不安全地使用了Groovy动态方法调用，使得应用程序很容易受到命令注入攻击。如果执行命令，第23行返回一个HTTP 204。我们的概念验证使用了一个类似的WebService SmartApp

## 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。智能应用程序可以读取与SmartThings hub配对的各种电池供电设备的电池电量。正如我们后面在§VI-E中所展示的，用户会考虑安装这样一个智能应用程序，因为它提供了有用的服务。智能应用程序只要求capability.battery。

我们在测试基础设施上测试了攻击应用程序，包括一个Schlage lock FE599(电池驱动)、一个智能电源插座和一个SmartThings集线器。测试基础设施包括一个从应用商店安装的智能应用程序，它可以执行锁码管理，这是一个为联网门锁用户提供的常见智能应用程序。在安装攻击智能应用程序时，用户被要求授权该智能应用程序访问包括门锁在内的电池操作设备。

图5显示了一般的攻击步骤。当受害者设置一个新的pin码时，锁管理器应用程序会在Z-Wave锁Device Handler上发出setCode命令。处理程序依次向集线器发出一系列set和get Z-Wave命令，然后集线器生成适当的Z-Wave无线电层信令。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。清单3显示了从我们的一次测试运行中提取的事件创建日志，其中包括明文pin代码值。此时，伪装的电池监控器SmartApp使用了SmartThings提供的不受限制的通信能力，通过短信向攻击者泄露pin码。

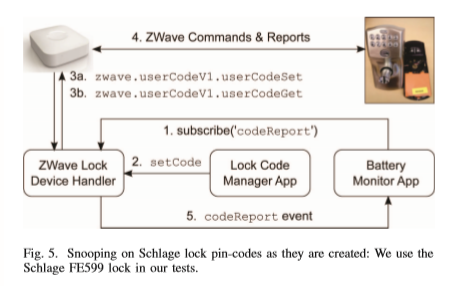
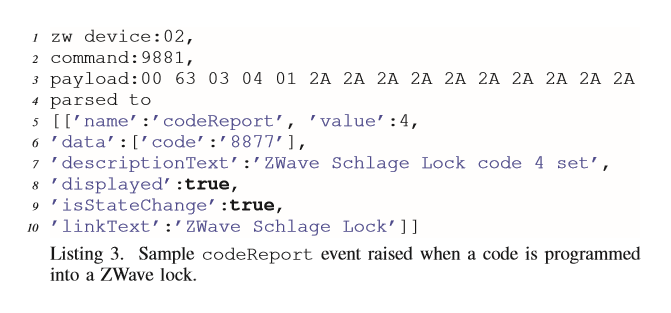


图5. 在创建Schlage锁pin码时窥探它们:我们在测试中使用Schlage FE599锁。



列表3.简单codeReport事件引发的当一个code被编程到一个Zwave 锁上

第一个基本问题是由于粗糙的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。第二个基本问题是，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。

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。这种攻击是偷偷摸摸的，可以让攻击者破门而入。详见附录A。

## 从任何设备泄漏事件

我们使用事件泄漏加强了我们的门锁pin-码窥探攻击。正如在IV中讨论的，如果一个无特权的应用程序学习了一个128位的设备标识符值，它可以从该设备侦听所有事件，而不需要获取相关的功能。我们修改了我们的伪装电池监测应用程序，使用一个128位的设备标识符的Z-Wave锁，并验证它可以监听codeReport事件，即使其是没有电池的能力。

一个很自然的问题是:攻击者如何检索设备标识符?设备标识符值在所有应用程序中都是常量，但如果设备从SmartThings中移除并再次添加，则会发生变化。设备标识符没有固定的模式(如递增值或已知项的可预测散列)。我们在下面讨论两个选项：

* 串通SmartApp:攻击者可以部署一个良性的串通SmartApp，读取各种设备的设备标识符，利用SmartApps不受限制的通信能力来泄露它们
* 远程开发另一个智能应用程序:如前所述，WebService智能应用程序可以远程开发。攻击者可以利用一个WebService SmartApp，让它输出一个所有设备的设备标识符列表，这个WebService SmartApp是经过授权的。

这两种技术都会泄漏目标物理设备的设备标识符。然后，攻击者可以将标识符传输到已安装的恶意应用程序。我们强调，我们的目的是展示智能应用程序如何使用设备标识符来升级其特权。

## 禁用休假模式攻击

度假模式是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，以此来阻止潜在的破坏和盗窃行为。我们从数据集中选择了一个SmartApp，它的依赖于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。当模式被重置时，SmartApp会停止占用模拟。

回想一下，SmartThings在send Location Event API周围没有任何安全控制。我们写了一个攻击智能应用程序，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。这种攻击只需要一行攻击代码，可以从任何智能应用程序启动，而不需要特定的capabilities。

## 假警报的攻击

我们展示了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控制它没有被授权访问的设备。我们从App Store下载了一个警报面板SmartApp。警报面板应用程序请求用户授权一氧化碳(CO)探测器，警报装置，运动传感器和水传感器。警报面板智能应用程序可以启动警笛警报，如果一氧化碳探测器被触发。我们编写了一个攻击智能应用程序，它为CO检测器引发了一个假的物理设备事件，导致警报面板应用程序发出警报声。因此，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，使用一个欺骗的物理设备事件来控制警报器警报。

## SmartThings用户调查研究

上面讨论的三种攻击要求用户能够安装一个攻击智能应用程序(Pin码窥探、禁用休假模式、假警报)。尽管一项调查显示，用户对安装Android应用程序(如[16])的安全和隐私风险的理解程度有所提高，但对智能家居应用程序的用户却没有类似的研究。为了评估我们的攻击场景是否真实，我们对SmartThings用户进行了调查，主要关注以下问题：

* SmartThings的用户会安装像电池监控器这样的应用程序来请求使用电池供电的设备吗?
* 用户希望电池监视器应用程序访问哪些安全关键的家庭设备(如门锁、安全警报)?
* 用户是否了解将安全关键的家用设备授权给电池监视器应用程序的风险?
* 如果用户知道电池监控器应用程序窥探了门锁的pin码，他们会有什么反应

从2015年10月到11月，我们通过(1)家庭自动化爱好者的工作场所邮件列表和(2)网络智能家居论坛招募了22名参与者。然而这并不影响我们的研究，因为如果我们的攻击欺骗有经验的参与者，那么它进一步支持我们的论点，攻击是现实的。所有参与者都表示拥有一个或多个智能家居中心。参与者报告连接到他们的集线器的设备数量从少于10个到接近100个不等。参与者平均报告安装了15个智能应用。在完成调查后，我们查看了回复，并奖励了参与者一张10美元的亚马逊礼品卡或一张10美元的工作间餐厅用餐卡。为了捕获参与者对应用程序安装请求的不带偏见的响应，我们完全没有提到安全性，并且宣传这项调查是关于SmartThings应用程序安装体验的研究。这项调查是由我们团队的研究人员设计和实施的，他们所在的机构不需要审查委员会的批准。其余的团队成员被限制访问调查结果。我们没有收集任何私人数据，除了那些想要收到礼品卡的人的电子邮件地址。发送礼品卡后，邮件地址被删除。

在调查的第一部分，我们介绍了电池监视器SmartApp。我们让参与者想象一下，他们已经安装了4个带有智能家居集线器的电池供电设备，并且可以选择安装电池监视器智能应用程序。然后，调查显示了SmartApp在所有安装阶段的屏幕截图。在设备选择界面中，调查显示了以下四种设备:SmartThings运动传感器、SmartThings状态传感器、Schlage门锁、FortrezZ siren频闪报警器。

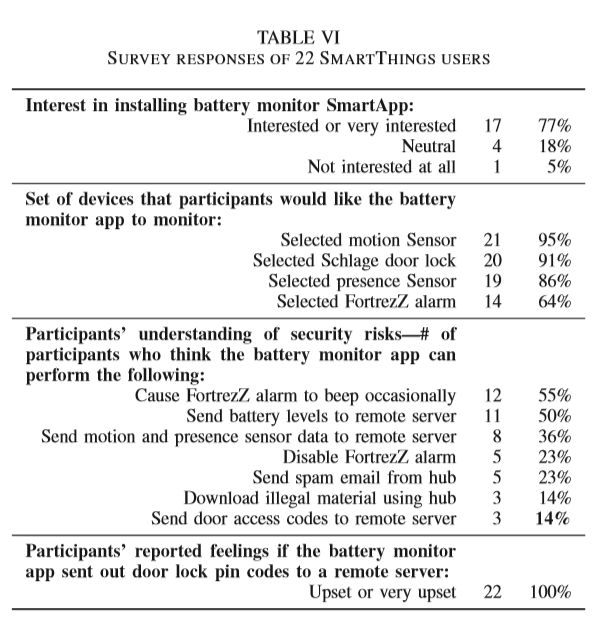
然后我们询问参与者，他们对安装智能电池监控器有多大兴趣。我们用Likert量表记录了人们的选择(1:不感兴趣，5:非常感兴趣)。接下来，参与者希望智能应用程序监控电池监控器。

我们设计了调查的下一部分，以衡量参与者对安装电池监视器SmartApp的安全和隐私风险的理解(或缺乏)。该调查首先展示了我们从SmartThings功能中得出的以下风险，并要求参与者选择他们认为电池监测应用程序可以采取的所有行动，而无需事先询问他们(除了监测电池电量之外)

* 导致FortrezZ报警哔哔声偶尔
* 禁用FortrezZ报警
* 发送垃圾邮件使用SmartThings hubs
* 下载非法材料使用SmartThings hubs
* 发送到远程服务器发送电池水平
* 发送SmartThings运动和存在传感器事件到一个远程服务器
* 收集门Schlage门锁接入码,发送到远程服务器
* 没有以上

请注意，如果允许访问相关敏感设备，电池监控器应用程序可以执行上述任何操作。然后，调查询问参与者，如果每种风险都发生了，他们会有多沮丧。我们用李克特量表记录了人们的选择(1:漠不关心，5:非常沮丧)。最后，调查询问了参与者的SmartThings部署情况。

表六总结了22名参与者的回答。结果表明，大多数参与者会对安装电池监控器应用程序感兴趣，并希望给它打开门锁的权限。这表明VI-B中讨论的攻击场景不是不现实的。附录C包含调查问题和所有回答。



只有14%的参与者似乎意识到，电池监控器应用程序可以监视门锁代码并向攻击者泄露pin码，而所有参与者都担心门锁窥探攻击。虽然这是一个小规模的在线调查，但结果表明，SmartThings框架中更好的保护措施是可取的。然而，我们注意到我们的研究有局限性，为了提高生态有效性，需要进行一项实地研究，测量人们是否真的会在他们的集线器上安装伪装的电池监控器应用程序，并让它进入他们的门锁。我们把它留给以后的工作去做。

# 七、挑战和机遇

我们讨论了从分析SmartThings平台(IV)中获得的一些经验教训，我们认为这些经验教训可以广泛应用于智能家居编程框架设计。我们也强调了一些国防研究方向。

## 教训1:非对称设备操作和基于风险的能力

烤箱控制capability的开启和关闭操作在功能上很有意义。 类似地，公开锁定和解锁的锁定capability在功能上有意义。 但是，随机时间打开烤箱会引起火灾，而关闭烤箱可能只会导致未煮熟的食物。 因此，我们观察到，功能相似的操作有时在相关的安全风险方面有所不同。 我们了解到，设备操作本质上是不对称的风险方式，而capability模型则需要将此类操作划分为等效类。

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。第一步可能是调整Felt等人的用户研究方法，该方法用于智能手机api[15]，包括来自多个利益相关者的输入:用户、设备制造商和框架提供商。

基于风险的功能划分会影响粒度。此外，用户很难理解和有效地使用细粒度系统。除了SmartThings，我们还调查了几个竞争对手的智能家居系统AllJoyn、HomeKit和Vera3的访问控制模型。我们观察了一系列的粒度，没有一个是基于风险的。在这个范围的一端，HomeKit将应用程序授权给家庭级别的设备。也就是说，一个应用程序要么可以访问所有与家庭相关的设备，要么什么也不能访问。Vera3也有类似的粒度模型。在这个范围的另一端，AllJoyn提供了为AllJoyn设备或AllJoyn应用程序的每个接口设置不同ACLs的方法。但是，目前还没有标准的接口集。用户必须在应用程序安装时配置ACLs，这对普通用户来说是一个可用性障碍。我们设想第二组用户研究，确定可用性和安全性之间的最佳折衷粒度级别。

## 教训2:任意事件和身份机制

我们观察到SmartThings事件子系统存在两个问题: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。任何可以访问设备ID的应用程序都可以监视该设备的所有事件。此外，应用程序容易受到欺骗事件的影响。如前所述，事件构成了基本的触发-动作编程范式的基础。因此，我们了解到安全事件子系统的设计通常对于智能家居平台是至关重要的。

提供强大的应用程序标识概念，并围绕引发事件进行访问控制，这可能是更安全的事件体系结构的基础。这样的机制可以让应用程序验证事件数据的来源，也可以让事件生产者选择性地传播敏感事件。然而，这些机制需要在基本层面上进行更改。AllJoyn[4]和HomeKit[5]都是从零开始建立起来的，有很强的身份认同概念。

Android intent是SmartThings事件的近亲。Android及其应用程序使用intent作为IPC机制和通知机制。例如，只要电池电量发生变化，Android操作系统就会触发一种特殊的广播意图。然而，与SmartThings不同的是，intent是建立在内核强制的uid上的。这种强身份的基础使意图接收者能够在对信息采取行动之前确定出处，并允许发送者选择性地传播意图。然而，意图使用中的bug可能导致绕过访问控制检查，并允许欺骗[11]。SmartThings的安全事件机制可以受益于现有的针对Android[22]的攻击防御研究。

### 合作，审查应用程序商店

与智能手机应用商店的情况一样，我们需要对智能家居应用程序的验证进行进一步的研究。像Groovy这样的语言提供了一些安全优势，但是也有一些可能被滥用的特性，比如正在执行的输入字符串。我们需要一些技术来验证智能家居应用程序是否能够抵御代码注入攻击、过度特权和其他更微妙的安全漏洞(例如，伪装的源代码)。

不幸的是，即使像SmartThings这样的编程框架提供商做了所有这些，其他的应用验证挑战仍然存在，因为并非所有我们发现的安全漏洞都是由于SmartThings应用本身的缺陷造成的。本文报道的其中一个漏洞是由用于控制智能应用程序的相关Android应用程序所包含的秘密引起的。这款安卓应用显然通过了谷歌的审查程序。谷歌不太可能发现这样的漏洞，并评估它对智能家居用户的风险，因为谷歌甚至无法使用Groovy应用程序。需要研究多个商店运营商(例如，SmartThings应用商店和谷歌Play商店)合作的方式，以验证与智能家居应用功能相关的整个生态系统。

智能家居设备及其相关的编程平台将继续激增，并将保持对消费者的吸引力，因为它们提供了强大的功能。然而，本文的发现表明，对于早期采用者和框架设计者来说，谨慎也是必要的。风险是巨大的，不太可能仅通过简单的安全补丁就能轻松解决。

# 八、总结

我们对流行的可编程智能家居SmartThings框架进行了安全评估。分析SmartThings是具有挑战性的,因为所有的应用程序运行在一个私有云平台和框架保护等主要组件间的通信中心,云的后端,应用智能手机伴侣。我们进行了市场规模越权分析现有的应用程序来确定如何SmartThings能力模型保护物理设备和相关的数据。我们发现(a)超过55%的现有SmartApps没有使用它们所要求的功能所隐式的所有设备操作权限，这主要是由于SmartThings提供的粗粒度功能;(b) SmartThings授予SmartApp对设备的完全访问权，即使它只指定需要对设备进行有限的访问;(c) SmartThings事件子系统没有足够的安全控制。我们结合这些设计缺陷与其他常见的漏洞,我们发现在SmartApps锁和能够窃取身份证号,禁用SmartApp度假模式,并导致假火警,都无需SmartApps功能来进行这些操作,不需要物理访问。我们的经验分析，加上我们总结的一组安全设计经验，是实现安全智能家居的第一个关键部分。

### 信息披露和响应

我们于2015年12月17日向SmartThings披露了本文中发现的漏洞。我们在2016年1月12日收到回复，他们的内部团队将在2016年4月之前基于后门pin码注入攻击加强他们的OAuth令牌，其他攻击载体将在未来的版本中考虑。我们还联系了Android应用程序的开发人员，他们的OAuth客户端ID和密码都存在于字节码中。开发人员告诉我们，他正在与SmartThings进行沟通，以帮助解决这个问题。正在考虑的一种可能的方法是，让开发人员为OAuth流提供一个重定向URI可能性的白名单，以防止任意重定向。2016年4月15日，SmartThings安全团队给我们发来了跟进回复。详见附录D。

# 附录

### 附录A:变相电池监控器

清单4显示了我们的电池监视器SmartApp的源代码。这款应用程序的设计初衷是监控电池电量(并且只能请求电量)，但它也可以窃取锁pin code。SmartApp的准确行为取决于从一个声称提供电池电量图表服务的Web服务接收到的命令。第60行用于攻击。它可以根据从远程服务接收到的配置执行http Post或sms发送。攻击者可以使用这个智能应用拦截并泄漏pin码。

### 附录B: OAUTH令牌窃取细节

我们在这里详细介绍OAuth令牌窃取过程。我们使用*apkstudio*和*smali*为WebService SmartApp拆解了一个对应的Android应用。我们发现Android应用程序开发人员在应用程序的字节码中硬编码了客户端ID和秘密值。使用客户端ID和secret，攻击者可以独立于Android应用程序完成OAuth流。我们的特定攻击涉及到创建一个攻击URL，其中*redirect\_uri*部分被攻击者控制的域替换。我们的攻击URL是*https://graph.api.smartthings.com/oauth/authorize?response\_type=code&client\_id=REDACTED&scope=app&redirect\_uri=http%3A%2F%2Fssmartthings.appspot.com* (we tested this URL in Dec 2015) (我们在2015年12月测试了这个URL)。请注意，我们已经修改了客户端ID值来保护Android对应的应用程序。

关于这个URL，有几点需要注意。

首先，它使用HTTPS。当单击URL时，用户将被带到真正的SmartThings登录表单，其中显示一个绿色的锁图标(图6)。

其次，重定向URI是一个受攻击者控制的域，但精心设计成包含“smartthings”这个词。

第三，URL相当长，重定向URI部分是URL编码的，降低了可读性。

SmartThings文档建议客户ID和秘密值存储在一个单独的服务器,在智能手机的应用程序,但需要一个单独的身份验证的用户的Android应用程序。没有什么可以防止攻击者妥协,单独的层验证如果不正确执行。



图6. OAuth窃取攻击:用户被带到正宗的SmartThings HTTPS登录页面。

### 附录C:调查结果

略

### 附录D:供应商跟进响应

2016年4月15日,SmartThings安全团队跟进他们的最初反应,要求我们添加以下信息:虽然SmartThings探索长期、自动化,防御能力解决这些漏洞,我们公司已经在下面提到的非常有效的措施来降低业务风险。SmartThings有一个专门的团队负责审核任何现有的和新的SmartApps。我们的即时缓解措施是让这个团队分析已经发布的和新的应用程序，以检测任何暴露HTTP端点的行为，并确保通过HTTP请求传递的每个方法名不会被动态调用。我们的团队成员现在还检查所有的web服务端点，以确保它们在其操作中是良性的。SmartThings继续努力加强最小权限原则，将有效访问的范围限制在执行任何给定的授权操作时明确需要的那些区域。此外，我们打算更新我们的内部和公开可用的文档，以便使用行政手段将这一做法正式化并加以实施。

