智能家居规则交互漏洞检测研究综述

王靖尧[1], 陈泽茂[2], 王婷婷[3]

[1]武汉大学 国家网络安全学院 武汉 中国 430000
[2]武汉大学 国家网络安全学院 武汉 中国 430000
[3]中国人民解放军北部战区海军 青岛 中国 266000

摘要 许多智能家居平台支持用户定义自动化规则以管理智能家居设备, 这些规则在同一场景下可能发生交互。如果这些规则间的交互存在逻辑漏洞, 则可能导致智能家居设备自动执行违背用户本意的意外动作, 从而给用户带来安全危害(例如, 人不在家时打开窗户)。因此, 进行规则交互漏洞检测技术的研究是紧迫且必要的。本文对近年来智能家居规则交互漏洞检测技术相关研究进行了广泛的调研和细致的汇总。首先本文对智能家居规则交互漏洞检测技术研究中涉及的四个重要概念进行了解释。随后本文对迄今研究中提出的规则交互漏洞进行总结, 将其分为了 5 类, 分别是条件绕过, 条件阻塞, 动作冲突, 动作重复以及意外规则链, 并分析了每类漏洞的形成机理和潜在危害。在此基础上, 文章对现有的交互漏洞检测方案进行了分类研究, 根据检测方案技术原理的不同, 本文将交互漏洞检测中具有代表性的 20 项工作分为三个类别, 分别是基于静态规则分析的 TAP 交互漏洞检测, 基于动态执行监控的 TAP 交互漏洞检测以及基于用户视角的 TAP 交互规则检测。接着, 本文从技术原理和实现效果两个维度对相关工作进行梳理和对比。最后结合研究现状和发展历程本文提出了当前研究中面临的 3 个研究热点, 分别是考虑物理交互的 TAP 交互漏洞检测, 考虑时间要素的 TAP 交互漏洞检测以及 TAP 规则交互检测中的安全属性表达。

关键词 智能家居; 规则交互; 漏洞检测
中图法分类号 TP309.1 DOI 号 10.19363/J.cnki.cn10-1380/tn.2025.07.02

A Review of Research on Smart Home Rule Interaction Vulnerability Detection

WANG Jingyao[1], CHEN Zemao[2], WANG Tingting[3]

[1] School of Cyber Science and Engineering, Wuhan University, Wuhan 430000, China
[2] School of Cyber Science and Engineering, Wuhan University, Wuhan 430000, China
[3] Northern Theater Command Navy, Qingdao 266000, China

Abstract Many smart home platforms allow users to define automation rules to manage smart home devices that may interact in the same scenario. If there are logical loopholes in the interaction between these rules, it may cause the smart home device to automatically perform unexpected actions that are contrary to the user's intention, thus causing safety hazards to the user (for example, opening a window when the person is not at home). Therefore, it is urgent and necessary to conduct research on rule interaction vulnerability detection technology. This paper conducts an extensive survey and detailed summary of the research on smart home rule interaction vulnerability detection technology in recent years. First, this paper explains the four key concepts involved in the research on smart home rule interaction vulnerability detection technology. This paper then summarizes the rule interaction vulnerabilities proposed in the research so far and divides them into five categories, namely condition bypass, condition blocking, action conflict, action reverting and unexpected rule chain. The formation mechanism and potential harm of each type of vulnerability are analyzed. On this basis, the paper conducts a classification study on the existing interaction vulnerability detection solutions. According to the different technical principles of the detection solutions, this paper divides 20 representative work in interaction vulnerability detection into three categories, which are TAP interaction vulnerability detection based on static rule analysis, TAP interaction vulnerability detection based on dynamic execution monitoring, and TAP interaction rule detection based on user perspective. Next, this paper reviews and compares the relevant work from two dimensions: technical principles and implementation effects. Finally, based on the current research status and development history, this paper proposes three research hotspots faced in current research, namely TAP interaction vulnerability detection considering physical interaction, TAP interaction vulnerability detection considering time factors, and security/safety properties expression in TAP rule interaction detection.

Key words smart home; rule interaction; vulnerability detection

通讯作者: 陈泽茂, 博士, 武汉大学教授, Email: chenzemao@whu.edu.cn。
本课题得到国家自然科学基金项目(No. 61872430)资助。
收稿日期: 2023-09-28; 修改日期: 2023-12-13; 定稿日期: 2025-06-26

1 引言

物联网(IoT)的概念自 1999 年提出以来, 已逐渐发展成为一个跨学科领域, 其产品已经进入智能家居、智能交通和智慧城市市场[1]。在过去十年中, 许多智能家居平台都试图鼓励和激发用户对智能设备、传感器和网络平台之间的交互进行定制化[2]。触发动作编程(TAP)是最常用的解决方案之一, 它可以帮助计算机编程几乎没有经验的用户指定其物联网设备的交互行为[3-5]。TAP 规则可以用“If-Then“规则的形式表示[6]。简而言之, 如果触发器的条件满足, 那么就会执行一个动作。触发器和动作都可以由用户定义, 并利用来自其他设备或网络服务的信息。目前支持 TAP 范式的平台主要有 If-This-Then-That (IFTTT)[7]、SmartThings[8]、Zapier[9]、OpenHAB[10]、Apple HomeKit[11]以及米家[12]。截至目前, 在智能家居设备[13-19]、通信协议[20-23]和智能家居应用平台[24-26]中发现了大量安全漏洞。与此同时随着这些智能家居平台上TAP应用数量的日益扩大, TAP规则间交互漏洞可能导致的潜在安全风险已不容忽视[27-46]。

例如, 在以下的场景中存在两条规则, 规则一: 如果前门被打开, 就打开门厅的灯。规则二: 如果是白天, 则关闭门厅的灯。这个场景展示了两个可能相互矛盾的规则, 当它们应用于一个环境时, 如果处于白天且前门被打开, 则门厅的灯无论是开启还是关闭都会违反其中一条规则。类似的规则矛盾还会导致严重的安全问题, 例如在一个场景中有以下两条规则, 如果烟雾传感器检测到烟雾, 则设置警报并打开洒水器。如果湿度传感器监测到湿度超出一定的阈值, 则关闭水阀。在这个场景下, 如果发生火灾, 水阀很可能只工作一段时间后就关闭, 从而给用户的财产和生命带来威胁。由此可见, 智能家居应用场景中存在的不安全的规则交互会带来安全性问题。

截至目前, 研究人员已经在现实世界的 TAP 规则中发现了多种类型的逻辑漏洞。这种逻辑漏洞会导致 TAP 规则和物联网设备之间的冲突。Wang 等人[27]、Chi 等人[28]和 Alhanahnah 等人[29]指出, 在他们检查的现实世界的 TAP 规则集中, 超过 66%的规则中发现了逻辑漏洞。更严重的是, 存在这些逻辑漏洞的 TAP 规则有可能被利用于破坏用户的居家安全。

为了检测智能家居平台中存在的 TAP 规则交互漏洞, 需要考虑的挑战有三点:

(1) 物理交互, 规则与规则之间的交互可能会通过光照, 温度等物理信道进行间接交互[31], 例如在以下场景中存在两条规则, 规则一: 如果前门被打开, 就打开加热器。规则二: 如果温度高于 27 摄氏度, 则关闭门厅的灯。规则一的动作执行后会导致温度的上升从而激发规则二的触发器, 间接导致门厅的灯会被关闭。类似地, 规则之间大量隐式交互会通过温度这样的物理信道达成, 会产生让用户意想不到的规则触发链, 从而给用户带来安全威胁。因此检测规则交互漏洞时, 对物理信道这一指标的处理能力直接决定了最终结果的好坏。

(2) 时间要素, 分析规则之间的交互时, 时间因素是必不可少的。一方面, 有些规则只会在特定的时间点触发。例如在以下场景一中存在两条规则, 规则一: 晚上 10 点后如果窗户是开启状态就关闭窗户。规则二: 上午的时候, 如果温度高于 27 摄氏度就打开窗户。如果忽略了时间因素, 那么由于规则一和规则二对窗户执行了相反的动作, 那么这两条规则是构成了交互的。将时间因素忽略就形成了场景二, 其中存在两条规则, 规则一: 如果窗户是开启状态就关闭窗户。规则二: 如果温度高于 27 摄氏度就打开窗户。在这个场景中, 一旦温度超过 27 摄氏度, 窗户会陷入开启和关闭的循环。但是事实是, 由于场景一中的两条规则有时间的限制, 它们是不会互相干扰的。另一方面, 有些规则存在规则延迟[35]。由于应用程序中定义的延迟(例如, 在 SmartThings 中的 runIn)或事件处理[47]的平台延迟, TAP 规则的动作可能不会立即执行。此外, 除了立即完成的瞬时和持续动作, 还有一种动作需要在一段时间内完成, 如上传视频。以上两个因素都可能造成规则延迟, 而规则延迟也会导致规则交互漏洞的产生。例如在以下场景中存在两条规则, 规则一: 如果温度高于 27 摄氏度就打开空调制冷。规则二: 如果温度高于 27 摄氏度, 则在 15 分钟后打开风扇。在这个场景中, 由于空调制冷, 在几分钟之内, 室内温度会低于 27 摄氏度, 而站在用户的角度思考, 低于 27 摄氏度后是没有必要开风扇的, 然而由于规则延迟触发, 会造成风扇意外开启。从上述两个方面考虑, 在检测规则交互时, 将时间纳入考量是很有必要的。

(3) 安全属性的表达与制定, 模型检测方法会检测模型是否满足定义的安全属性, 因此安全属性的效果会直接影响模型检测的结果。现有的研究一般将安全属性分为通用安全属性和特定安全属性[11-12,14,17-18,20,23]。通用安全属性的制定依据的是对 TAP 规则交互漏洞的分析, 一般每一类规则交互漏洞都会有为其量身定制的安全属性, 这要求对规则交互漏洞的分类有充足的分析。通用安全属性如

果制定不规范会导致假阴性和假阳性的出现。特定安全属性的制定依据的是经验, 很多研究会将一些不合理的行为设为安全属性, 让模型检测时能给出警报。例如 IoTCOM[29]的一条特定安全属性是当卧室门关闭时, 不许打开卧室的灯。特定安全属性需要确保坏事不可以发生以及好事需要发生。然而好事和坏事是主观的判断, 需要看用户的需求, 仅仅依靠经验是不够的。例如晚上 10 点后要关灯这一安全属性, 对于很多人来说, 这条安全属性并不合适。综上所述, 安全属性的制定也是一项很重要的工作。

上述的挑战给智能家居平台规则交互漏洞的方案设计带来了很多麻烦。但是经过多年的研究, 这些挑战都或多或少被关注, 许多优秀的研究方案都被提出和验证[27-46]。此外, 还有其他致力于提升智能家居平台安全的研究工作, 例如权限一致性检查[48]、引入身份验证和访问控制机制从而在运行时阻止有害行为[49-52]、入侵及异常检测[53-56]和攻击后的溯源分析[57-59]。本文则主要聚焦于 TAP 规则交互漏洞检测研究领域。

本文主要贡献如下:

(1) 对智能家居中存在的规则交互漏洞进行归纳和总结。目前学术界对规则交互漏洞的定义以及分类都有所差别, 本文梳理了学术界对规则交互漏洞的认知发展历程, 并给出五类典型的规则交互漏洞定义。

(2) 对规则交互漏洞检测相关的学术研究进行归纳和总结。本文将近五年所搜集到的规则交互漏洞检测方案分为三类并分别给出简要的介绍。同时针对检测方案中关注的挑战点, 本文对其中的难点和技术路线进行了梳理。

本文的组织结构如下: 第 2 节介绍 TAP 规则交互漏洞及其检测研究中所涉及的物联网系统模型、TAP 规则、智能家居平台架构和模型检测等概念。第 3 节为本文讨论的安全问题提供了一个威胁模型并重点讨论规则交互漏洞的定义和分类。第 4-6 节梳理了规则交互漏洞检测研究方案。第 7 节是我们关于规则交互漏洞检测研究热点和挑战的三点思考。第 8 节总结全文。

2 相关概念

2.1 智能家居物联网系统模型

为了便于介绍这项工作, 给出智能家居系统中涉及的物联网术语[46]: 因素(factor)是与物联网系统相关的任何元素, 包括智能设备、用户和环境。例如, 温度传感器、天气、一天中的时间等。属性(attribute)是因素的一个方面。用户属性的一个示例是用户的居家状态。因素和属性的组合称为变量(variable)。例如, 前门是否上锁和小李是否在家。状态(state)是在一段时间内对变量正确的陈述, 例如, 小李一直处于离家状态。而事件(event)是变量状态的瞬时变化, 例如, 小李出门了。物联网系统状态(smart home system state)是环境中该时间点所有变量状态的集合。

物联网系统模型可以被抽象为数据层和控制层[28]。

数据层由传感器、执行器和环境组成。1) 传感器可以是测量物理特性(例如温度)的传感组件, 也可以是报告物联网中某个因素状态(例如门窗的开与关状态)的设备, 还可以是智能家居平台中定义的一个系统变量(例如房间的有人与无人状态)。2) 一个执行器可以是一个可控的设备, 也可以是一个系统变量。物联网设备可以是传感器、执行器或两者的组合装置。3) 环境具有时间、温度、光照度等一系列物理特征。传感器、执行器与环境之间的相互作用如图 1 所示。传感器可以感知环境, 执行器可以通过改变设备和系统变量的状态(例如, 打开开关产生打开事件)或通过物理信道(例如, 加热器可以提高温度)来影响传感器。

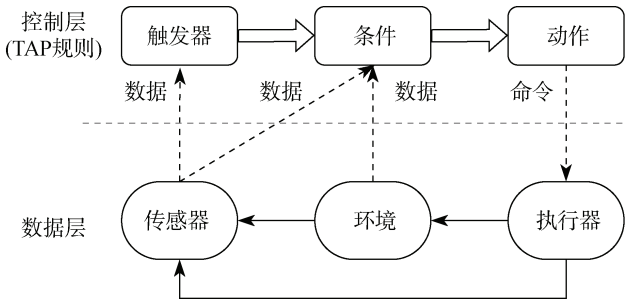

图 1 物联网系统模型

Figure 1 IoT system model

控制层由应用程序定义的自动化规则组成; 一个应用程序通常定义一个或多个规则。在智能家居系统中, 规则模型遵循触发器–条件–动作范式(即 TAP 编程范式), 如图 1 所示。在一个智能家居系统中, 触发器订阅激活规则执行的事件(例如, 打开电视)。条件包括家庭相关数据(传感器读数、设备和系统变量状态、时间等)。必须满足这个条件, 这个规则才能继续进行。触发器(trigger)和条件(condition)之间的区别在于触发器可以被激活执行, 而条件本身无法执行。此外, 条件是可选的, 并且可以是空的。动作(action)通常是向执行器发出一个或多个命令或通知用户。

数据层和控制层将在两个方向上进行交互。一方面, 规则从传感器和环境(例如, 时间)获取数据。另一方面, 规则会向执行器发送命令, 从而进一步影响传感器和环境。

2.2 TAP 规则

目前主流的智能家居平台均采用的规则模型是 TAP 编程范式。TAP 规则最简单的形式是 IF[触发器]THEN[动作]。动作是一个可以自动执行的事件, 比如让智能门锁自行上锁。触发器被激活后就会触发动作的执行。对于单触发器和单动作而言, TAP 规则十分简单易懂。例如, 如果温度高于 25 摄氏度则打开窗户。然而, 在日常生活中, 有许多居家行为, 即使非常常见, 也需要更强的 TAP 规则表达能力。例如, 对于复杂设备(如空调)的设置、对于场景本身安全相关设备(如门锁和窗户)的可靠设置、对于夜灯和咖啡机等设备的工作持续时间的设定。为了支持用户的日常需求, 不同的智能家居平台提供了不同的支持方式。例如, Stringify[60]引入 and 连接多个触发器[6]或者连接多个触发器和多个动作, HomeAssistant[61]使用更一般的 and/or 组合多个触发器或动作, Zapier[9]和 HomeAssistant[61]增加对触发器条件的过滤, SmartRules[62]增加“状态保持时长”的表达能力, HomeAssistant[61]支持自定义事件/服务。这些 TAP 拓展, 极大增强了用户的操作空间。

Huang 等人[63]提出用事件和状态区分触发器: 事件是瞬时信号(如“温度降到 10 摄氏度以下”), 而状态是可随时评价的布尔条件(如“在下雨”)。Brackenbury 等人[64]进一步将动作分成事件(在特定时刻发生, 如“关灯”)和状态(一段时间内设备的期望状态, 如“灯应亮着”)两类。Brackenbury 等人[64]提出了三种多触发器组合的时序范式: Event-Event, Event-State 和 State-State, 并将它们与动作组合成表 1 所示的 4 类 TAP 规则。

目前事件状态(event-state triggers)触发器的使用度是最广泛的, 之前的工作[33-35]发现这种 TAP 范式类型对用户来说是最直观的。事件状态触发器由一个事件和零个或多个必须全部为真才能触发动作的状态组成。一般只将事件状态触发器的事件部分称为触发器, 而将状态(如果有的话)称为条件集, 因此规则具有“if[触发器]WHILE[条件]THEN[动作]”的形式。充分理解规则的语义对规则交互漏洞的检测至关重要, TAP 规则复杂度的增加对提取规则语义提出了更高的要求。

2.3 TAP 智能家居平台

图 2 显示了 TAP 智能家居平台的架构和工作流的一般表示。一个 TAP 智能家居平台通常涉及三方, 即触发器-动作平台、触发器系统和动作系统。

表 1 TAP 规则分类

Table 1 TAP Rule Classification

范式类别	范式定义	规则举例
Event-Event→Event	IF event(AND event|AND AFTERWARDS event)*(WITHIN timewindow)THEN event	IF 主人离家 AND AFTERWARDS 外卖送达 WITTHIN 5 分钟 THEN 通知主人
Event-State→Event	IF event(WHILE state (AND state)*)* THEN event	IF 主人进入卧室 WHILE 晚上 THEN 打开卧室灯
Event-Event→State	IF state(AND state)* THEN state PRIORITY priority	IF 小明不在家 AND 小黄不在家 THEN 空调应处于关闭状态 PRIORITY 0 IF 室内温度高于 28 摄氏度 THEN 空调应处于制冷模式 PRIORITY 1
State-State→Event	IF state(AND state)* THEN event	IF 室内温度高于 30 摄氏度 THEN 记录温度到日志文件

触发器-动作平台。触发器-动作平台为自动化规则提供服务和设备之间的交互接口。它由两个主要的组件组成, 即触发器-动作服务端(如云服务)和触发器-动作客户端(如移动或 web 客户端)。该触发器-动作服务成为在线触发服务和动作服务相互通信的通道。任何两个服务之间的接口都受其提供者之间预先建立的协议的约束。触发器-动作平台可以通过提供 app(如 SmartThings)、applet(如 IFTTT)或者规则模板(如米家)让用户定制自己的自动化服务。

触发器系统和动作系统。第三方系统可以与触发器-动作平台集成从而启动触发器(即触发器系统)或者完成动作(即动作系统)。与触发器-动作平台类似, 每个系统由云服务和客户端(例如, 设备、移动或 web 客户端)组成。触发器服务启动自动化任务, 动作服务完成它们。

TAP 智能家居平台工作的一个示例: 给定一个自动化任务“如果外面开始下雨, 那么就关闭窗户”, TAP 智能家居平台将通过以下三个步骤实现它。

步骤 1:**服务连接。**用户授权触发器-动作服务通过触发器-动作客户端与所选的触发器服务(如第三方天气服务 Weather Underground)和动作服务(如智能窗户)进行连接和认证(图 1 中的步骤①)。

步骤 2:**TAP 规则创建。**TAP 规则创建时要指定“如果这个触发器发生, 那么就触发那个动作”。它会

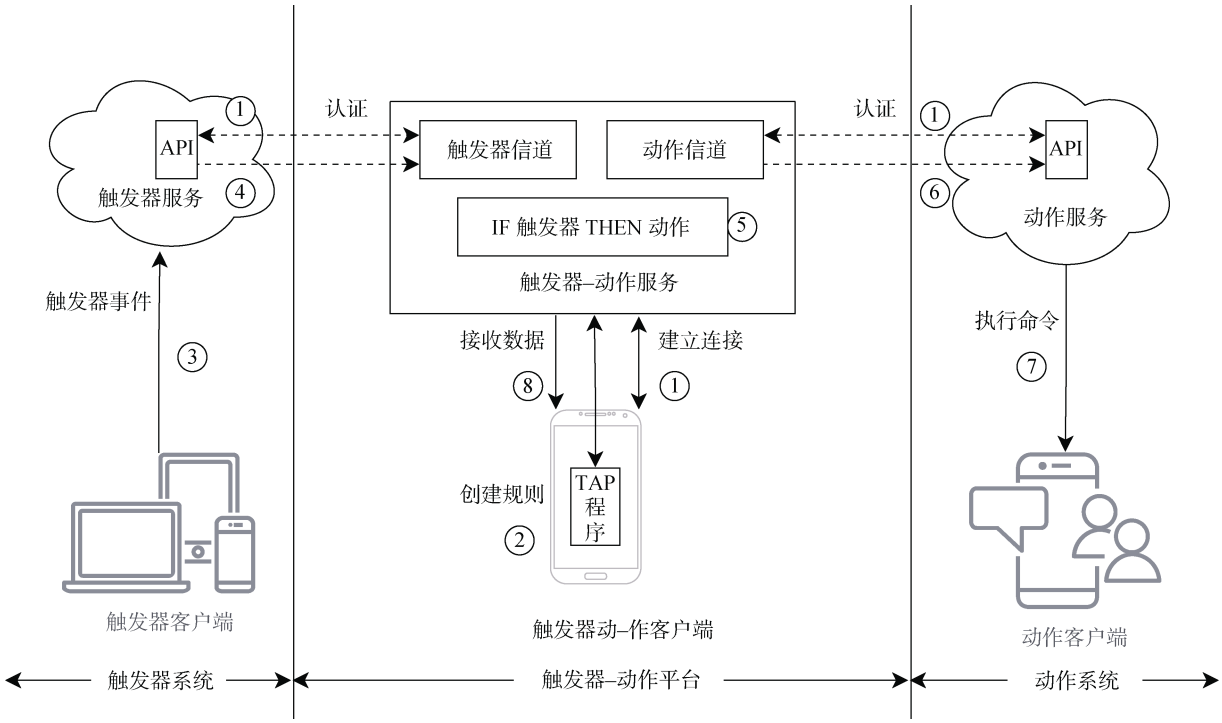

图 2　TAP 智能家居平台架构及工作流

Figure 2　TAP smart home platform architecture and workflow

从连接的服务中选择符合需求的事件作为触发器和动作(步骤②)。这个示例中, 用户从天气服务中选择下雨作为触发器, 再将动作服务设置为关闭智能窗户即可。

步骤 3:**TAP 规则执行**。创建的规则可以通过在触发器客户端启动一个触发事件来执行(步骤③)。在示例中, 当外界天气变为下雨时, 第三方天气服务接收到一个触发器。然后通知触发器-动作服务, 此时触发器事件已经发生(步骤④)。与触发器事件相关联的规则将由触发器-动作服务(步骤⑤)执行, 然后它将定义好的动作驱动发送给动作服务(步骤⑥)。作为响应, 动作服务完成动作(即关闭窗户), 并将智能门窗的状态更新推送到动作客户端(即智能窗户的客户端)(步骤⑦)。此外触发器-动作客户端还接收执行日志, 其中包含规则标识、执行状态和从触发器服务(步骤⑧)接收到的数据(例如文本信息)。

2.4　模型检测

形式化验证(formal verification)是一种使用数学方法和逻辑来证明系统是否满足预定义规范的技术。在计算机科学和软件工程领域, 形式化验证主要用于确保系统或软件的正确性、安全性和可靠性。模型检测(model checking)是形式化验证的一种重要技术, 它可以自动验证有限状态系统(如硬件、协议或程序)是否满足给定的时序逻辑规范[66](Temporal Logic Specification), 文献中也将这些规范称为安全属性(Safety/Security Properties)。

模型检测过程一般包括以下三个步骤:

(1) 构建系统模型(System Model): 模型检测首先需要对要验证的系统进行建模。这个模型通常表示为一个有限状态机(Finite State Machine, FSM), 其中包括一组状态、一组转换关系和一个初始状态。状态表示系统在某个时间点的配置, 转换关系表示系统从一个状态到另一个状态的变化。

(2) 制定安全规范(Specification): 规范是用来描述系统期望行为的一组条件。这些条件通常使用时序逻辑(Temporal Logic)来表示, 这是一种专门用于表示和推理时间相关属性的逻辑形式。常见的时序逻辑包括线性时态逻辑[67](Linear Temporal Logic, LTL)和分支时序逻辑[68](Computation Tree Logic, CTL)。

(3) 设计模型检测算法(Model Checking Algorithm): 模型检测算法是一种自动化的验证过程, 通过系统模型和规范(安全属性)作为输入, 来确定系统是否满足给定的规范(安全属性)。这个过程的主要任务是在系统的状态空间中寻找一个违反规范的执行轨迹(也称为反例)。如果算法找到了这样的执行轨迹, 那么就证明系统不满足规范; 否则, 系统被认为是满足规范的。

模型检测器(Model Checker)是用于执行模型检测过程的一种工具或软件。它根据输入的系统模型

和规范, 判断系统是否符合预期的行为。有许多不同类型的模型检测器, 它们支持不同的系统建模方法和规范逻辑。常用的模型检测器有: 1) SPIN[69]: SPIN是一种流行的模型检测工具,主要用于验证分布式软硬件系统的正确性。它支持建模语言 Promela,可以进行状态空间搜索来验证模型的性质。SPIN 在通信协议,多线程软件,嵌入式系统等领域有很多应用。2) NuSMV[70]: NuSMV 是一个基于符号模型检测技术的模型检测器, 支持有限状态机、硬件描述语言和高级软件语言等多种建模方法, 支持分支时序逻辑(CTL)和线性时序逻辑(LTL), 在硬件验证、协议分析、规划等领域有广泛应用。3) Alloy Analyzer[71]: Alloy Analyzer 是一种基于关系代数的轻量化形式化验证工具。它使用 Alloy 语言构建抽象模型,并使用 SAT 求解器自动分析模型。Alloy 可用于软件设计和规格分析、安全协议分析、多态分析等。它适合概念建模与模型的初步验证。

3 规则交互漏洞

在目前的研究中定义了各种规则交互漏洞[27-29,32,35,43], 在这些研究中会根据漏洞的特征针对性设计安全属性, 从而使得借助模型检测技术将这些漏洞识别出来。在很多研究中将其认为是安全属性的违反[12,14,18]。Wang 等人[27]将其称为规则间漏洞(inter-rule vulnerabilities), Chi 等人[28]将其称为跨应用程序交互威胁(Cross-App Interference (CAI) threats), Alhanahnah 等人[29]则将其称为交互威胁(Interaction Threats)。本文采用了 Yu 等人[35]的说法, 将其称为规则交互漏洞(Rule Interaction Vulnerabilities), 原因是此类漏洞本质上就是由于 TAP 规则通过其动作的执行造成交互, 从而给用户和场景带来了安全威胁。

3.1 威胁模型

规则交互威胁可能由两种群体造成: 攻击者和用户。

对攻击者的假设: 攻击者试图通过针对物联网平台应用程序逻辑层的规则级攻击, 暗中破坏物联网部署。规则级攻击通过利用物联网自动化规则之间的交互来最终破坏用户的意图。这种交互可能使攻击者能够执行特权操作, 导致设备上的服务被拒绝, 或者访问属于用户的敏感信息。这些攻击只能通过调用由用户合法安装的自动化规则来启用。攻击者可以通过许多场景创建或检测规则级攻击的机会: 1) 利用: 攻击者发现了两个或多个良性应用之间存在规则的可利用交互, 或通过操纵第三方服务触发事件[72]; 2) 目标规则:攻击者诱使用户安装恶意的规则, 例如通过网络钓鱼或社交工程[27]; 3) 恶意应用程序: 攻击者开发并发布包含隐藏规则的恶意应用程序[32]。

对用户的假设: 1) 用户对应用程序或规则的全部功能存在认知错误, 原因是应用程序提供的描述文档比较模糊[32]; 2) 用户缺乏相关的专业知识, 从而无法意识到存在微妙的规则交互; 3) 存在多个业主, 且不同的业主根据自己的需求安装了不同的应用程序和规则, 因此每个业主都很难获得全局视图; 4) 用户错误配置应用程序或规则[28]。在用户导致的规则交互漏洞中, 一部分可以归为严重的安全威胁, 一部分可能是烦人但无害的安全威胁, 另一些可能是为了满足用户在特定场景下的需求。

3.2 规则交互

规则在这里特指 TAP 规则。在一个通用的用户场景中可能存在一个或多个智能家居平台, 每个智能家居平台都可能安装不止一个应用程序, 每个应用程序的行为由一组规则 R 组成, 最通用的假设这些规则遵循触发器-条件-动作范式。因此每条规则 r 都可以表示成一个触发器、条件和动作的元组 $<T_r,C_r,A_r>$。T_r 表示规则 r 中触发器的集合, C_r 表示规则 r 中条件的集合。A_r 表示规则 r 中动作的集合。每个组件(触发器、条件或动作)被表示成一个三元组[29], 分别是一个设备、设备的一个属性和属性中可取的一组值, 即每个 T, C, A 都可以表示成<device, attribute,value>。例如解锁作为动作可以表示成<TheLock, locked,unlocked>。

规则之间通过以下三种方式之一进行交互: 1) 访问相同的设备。规则 r_1 和 r_2 可以访问共享设备, 例如 r_1 基于光照度传感器的输入打开灯, 而 r_2 可以基于运动传感器关闭灯。这两个规则都在同一设备(灯)上执行动作, 这被称为设备交互[73]。2) 物理信道。如果 r_1 的动作会导致环境中物理信道的变化, 从而影响 r_2 的执行, 那么两个规则就具有物理信道交互[31]。例如 r_1 在一天中的某个时间激活吸尘器机器人, 机器人的运动可以触发运动传感器从而让 r_2 执行[41]。3) 访问相同的全局变量。规则 r_1 和 r_2 可以通过相同的全局变量进行交互, 其值存储在云端, 例如 r_1 更新变量, r_2 访问它。这被称为全局变量交互[19,43-44]。SmartThings 平台中唯一允许写和读访问的全局变量 location.mode。其有三个预先配置的值: Home, Away 和 Night。一个示例场景是, 一条规则根据人体传感器的输入更新 location.mode, 而第二个应用程序读取它, 以确定一个门是否应该上锁。

3.3 漏洞类别

根据现有的研究[27-46,74-75], 给定一对自动化规则 i 和 j, 其产生的规则交互漏洞可分为五个类别:

(1) 条件绕过[27-29]。条件绕过分为两种情况, 第一种情况如图 3 所示, 如果规则 i 和规则 j 具有相同的动作和触发器, 但是需要的条件却不同。往往规则 j 中需要满足的条件要比 T 少(甚至没有), 那么利用规则 j 就可以绕过规则 i。例如, 如果规则 i 为温度高于 30 摄氏度, 且有人在家, 且在下午 6-8 点之间, 打开窗户。规则 j 为温度高于 30 摄氏度, 打开窗户。利用规则 j 可绕过规则 i 的设定, 使得实际中设定更安全合理的规则 1 形同虚设。

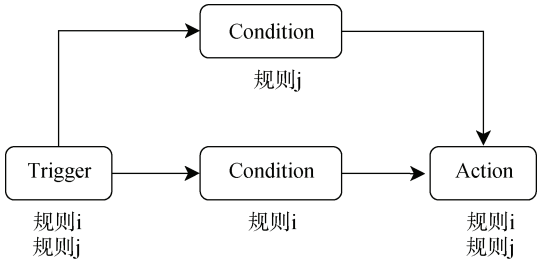

图 3 条件绕过(a)
Figure 3 Condition bypass (a)

条件绕过的另一种情况如图 4 所示, 规则 i 的动作可以达成另一条规则 j 中的条件, 从而使得规则 j 更容易被触发。例如, 规则 i 为如果有人在家就打开卧室灯。规则 j 为如果温度高于 30 摄氏度且卧室灯开着, 就打开窗户。

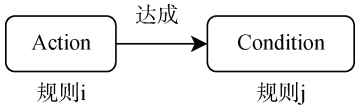

图 4 条件绕过(b)
Figure 4 Condition bypass (b)

(2) 条件阻塞[27-30,64], 如图 5 所示, 当一个规则 i 的执行动作无意中阻止了对另一个规则 j 的触发时(阻止规则 j 的条件达成), 就会发生条件阻塞。例如, 规则 j 是如果门前的运动传感器被触发, 且居家模式为 away, 就发送警告短信给房主;而规则 i 的动作会将居家模型改为 home, 那么规则 i 就会阻止规则 j 的执行。有的时候条件阻塞是很隐蔽的, 例如规则 i 是如果没人在家, 关掉智能插座, 以节约能源; 规则 2 是如果是上午 10 点, 打开智能宠物喂食器。如果宠物喂食器是由智能插座供电的, 并且上午 10 点没有人在家, 宠物将不会被喂食。

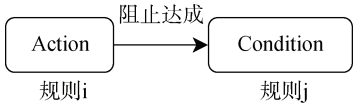

图 5 条件阻塞
Figure 5 Condition blocking

(3) 动作冲突[27-29,36,63], 如图 6 所示, 规则 i 的动作和规则 j 的动作是互相冲突的动作(如一个开窗户一个关窗户), 即对相同的设备中的同一属性将其修改为不同的值。这时存在的一种情况是如果规则 i 和规则 j 的触发器会被同一事件所触发(Event1 和 Event2 是同一个事件), 且条件也可能被同时满足, 那么就会发生动作冲突。最简单的情况是, 两条规则的触发器都设置为检测到有人, 而动作分别执行开门和关门。还存在一种隐蔽的情况是两个规则不是由同一事件触发的, 因此不是同时执行的, 而是执行相互冲突的命令。这两个规则是分开工作的, 但后一个规则会立即或在一段时间后覆盖前一个规则发出的命令。例如, 在一个场景中同时存在两条规则, 规则 i 为如果厨房传感器检测到烟雾, 就打开窗户; 规则 j 为如果天黑了, 请关闭窗户。那么当烤箱中有东西在燃烧, 且是晚上的时候, 可能会因为规则 i 和 j 的动作冲突无法打开窗户。

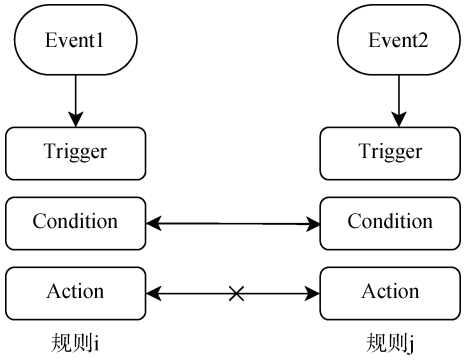

图 6 动作冲突
Figure 6 Action conflict

(4) 动作重复[27-39], 如图 7 所示, 规则 i 和规则 j 会执行相同的动作(例如不断的开锁)。对于某些只有两个状态值的设备(例如锁的状态属性只有开锁和关锁两种属性值), 这样的动作也可以改变设备的状态, 例如执行拨动开关这个动作既可以是开启开关也可以是关闭开关。

(5) 意外规则链[27-29,36,63,66], 如图 8 所示, 一个规则可能意外触发另一个规则, 即一个规则 i 的动作可能会触发规则 j 设置的触发器(通过物理信道)。例如, 如果规则 i 是如果我的房间温度低于 20 摄氏度, 打开我

房间的加热器; 规则 j 是如果我的房间温度高于 25 摄氏度, 打开窗户。执行第一条规则后, 温度可能上升到 25 摄氏度以上, 并激活规则 j, 这会导致窗户意外打开。

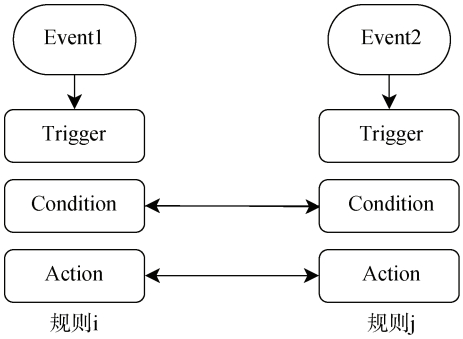

图 7 动作重复

Figure 7 Action reverting

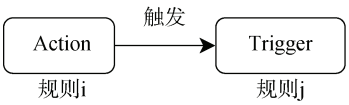

图 8 意外规则链

Figure 8 Unexpected rule chain

意外规则链的危害是相当大的, 通过层层嵌套让规则链形成循环, 可以实现动作的反复触发, 很多研究中将其称为动作循环。最经典的例子是在同一个场景中(有台灯和顶灯)有两条规则, 规则 i 设定是太亮的时候(借助光照度传感器), 关闭所有的灯, 因此会将台灯和顶灯关掉; 但是规则 j 设定让屋内的台灯开着。因此, 台灯会不断的被开启和关闭。

4 基于静态规则分析的 TAP 交互漏洞检测

为了检测出智能家居平台中存在的 TAP 规则交互漏洞, 目前的研究提出了很多有效的方案, 这些方案的技术路线和解决问题的侧重点都有一定的差别。本文将其中的一类方案称为基于静态规则分析的 TAP 交互漏洞检测方案, 此类方案的通用框架由下图 9 所表示:

以 TAP 智能家居平台作为输入, 经过规则信息提取和形式化验证两个步骤输出漏洞的检测结果。在形式化验证阶段, 静态分析方案只需要对当前场景中的模型进行静态验证即可, 不需要相关程序运行。

(1) 规则提取阶段: 本阶段主要构建 TAP 智能家居系统中的 TAP 规则模型, 以此作为形式化验证阶段的输入。最开始输入一个实际的 TAP 智能家居平台(目前的研究都以 SmartThings 或 IFTTT 作为研究平台), 这个平台上面有许多的 TAP 应用程序(SmartThings 上是 SmartApp, IFTTT 上是 Applet)。每个程序都可能会操作一个或多个物联网设备, 每个程序也都设立了一系列 TAP 规则。规则信息提取阶段又分为三个步骤: 规则提取, 配置提取以及构建规则模型。

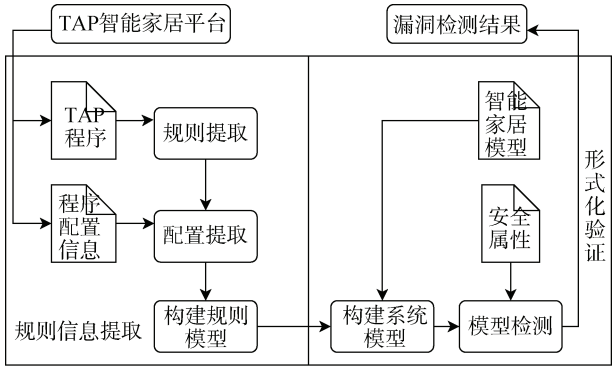

图 9 规则交互漏洞检测方案通用架构

Figure 9 General architecture for rule interaction vulnerability detection scheme

TAP 程序可以看成是一组 TAP 规则的集合, 规则提取的目的是将 TAP 程序中包含的每条规则 R 提取出来。这一步需要正确的识别并提取出规则 R 中的触发器 T, 动作 A 和约束条件 C 以及每个组件中涉及的设备、属性和属性值。这一步需要用到程序静态分析技术, 这些技术在整合规则信息时会产生不同的中间表示, 但无论哪种中间表示一定会包含触发器, 约束条件以及动作等信息。然而用户使用智能家居平台上的 TAP 程序时需要进行配置, 指定相应设备的 ID, 将虚拟设备与实际设备的配对, 以及设置一些环境的阈值, 例如设置一个温度阈值, 使得温度到达阈值后, 程序会触发执行相应的操作。为了检测特定场景中的规则交互漏洞, 配置提取这一步骤也是必不可少的。最后, 经过规则提取和配置提取两个步骤后, 规则的基本信息都搜集完毕, 检测方案根据后续形式化验证的需要, 会设计一套形式化建模方案。构建规则模型就是将规则的基本信息按照建模方案进行建模得到形式化的规则模型以用于形式化验证。

(2) 形式化验证阶段: 本阶段基于所提取的 TAP 规则进行交互漏洞检测, 目前的检测方案大多基于模型检测方法进行形式化验证。模型检测方法在检测规则交互漏洞时需要两个输入, 系统模型和安全属性。模型检测会依据安全属性对给定的系统模型进行形式化验证。此阶段主要涉及三个方面的工作: 系统模型构建, 安全属性设计以及模型检测。

规则提取阶段会提供当前物联网场景中存在的规则模型。智能家居模型会提供智能家居场景中设

备与物理信道之间的对应关系, 因此整合规则模型和智能家居模型就可以得到当前物联网场景的系统模型。这里涉及到模型构建算法的设计。安全属性的设计分为两方面, 一方面依据对漏洞特性的分析设计出通用的安全属性, 另一方面依据智能家居中安全性保障的常识性要求设计出特定的安全属性。模型检测方法可以选择成熟的模型检测器也可以选择针对性设计模型检测算法。需要注意, 部分模型检测算法并不需要额外输入安全属性。

接下来从规则信息提取和形式化验证两个角度出发梳理研究中采用的技术。

4.1 规则提取

Soteria[30]利用算法对 SmartTings 平台中的 SmartApp 进行预处理将源代码转换成中间表示(IR)。IR 包含三种关键信息: 1) 权限, 它会授予应用使用设备的权限。2) 事件订阅信息, 反映了事件和动作之间的关联信息。3) 调用图, 代表了一个事件处理程序的方法。通过 IR 可以精确地建模应用程序的生命周期还可以省略与形式化分析无关的部分代码。

IotMon[31]对 SmartApp 进行静态程序分析, 其分为三个阶段: 1) 首先, 为应用程序构建一个抽象语法树(AST)。对给定的物联网应用程序执行轻量级的静态分析。它通过创建和解析应用程序的 AST 来分析应用程序的规则语法, 以查找触发器和动作。2) 分析代码中的首选项部分, 首选项中会声明应用程序的所有功能和输入。首选项部分旨在让用户设置适当的设备和阈值。因此这一步会在 AST 上遍历这一节并构建一个功能和输入的列表, 从而成功提取出配置信息。3) 提取应用程序的触发器和动作。通过解析订阅(subscribe)函数来识别触发条件, 这些函数定义为在平台上注册事件以触发动作。可以通过分析已安装和已更新(installed 和 updated)函数识别动作。

IotSan[32]则是通过构建依赖关系图来捕获不同应用程序的事件处理程序之间的交互。依赖图的构建需要提取出应用程序的输入和输出事件, 输入事件有三个来源: 1) 订阅命令中显式声明。2) 通过读取智能设备状态的 api 识别。3) 由调度方法调用特定时间的中断指示。输出事件通过改变智能设备状态的 api 调用。IotSan[32]使用静态分析枚举应用程序的输入和输出事件。IotSan[32]还构建了一个翻译器, 可以自动将 Groovy 程序转换为 Promela, 将上一步得到的存在交互的事件处理程序转换为 Promela 代码。最后借助爬虫对已经提供用户名和密码信息的账号获取配置信息。

针对 IFTTT, iRuler[27]首先提取应用程序中的所有规则, 然后将这些规则转换成统一的规则表示形式(RR), RR 中规定规则由一个触发器和一个或多个动作组成。触发器被定义为带有约束的事件, 而事件是根据主体(例如, 某个设备)和属性来定义的。动作由条件、主体、要执行的命令和命令的参数组成。条件或约束可以为空(即没有条件)或逻辑表达式。它们之间的区别是, 约束是事件数据上的谓词, 而条件可以是其他主体上的谓词。

IoTCOM[29]使用一种图形抽象技术自动推断应用程序的行为模型。它首先对每个应用程序执行路径敏感分析, 以生成过程间控制流图(ICFG)。对于 IFTTT 平台中的 applet, 它会将每个 applet 的 Trigger 作为控制流图的入口节点(entry node), 把 Action 作为控制流图的方法调用节点(method call node)。对于 SmartTings 平台中的 SmartApp, 它会先提取四类主要信息: 1) 应用程序中使用的设备和属性。2) 用户配置的应用程序。3) 任何全局变量中定义的文档或设置使用的状态对象。4) 应用程序的触发器的入口方法, 入口方法分为 subscribe, schedule, runIn, runOnce 四种。之后为每一个入口方法建立一个 ICFG。得到 ICFG 后将其中无关的信息剔除(人工处理)就得到了行为规则图。最后将行为规则图转换为规则的形式化模型, 每条规则都是一个触发器、条件和动作的元组。

Homeguard[28]对智能家居平台的规则信息提取分为三个步骤: 1) 语义信息提取, 利用符号执行技术从 SmartApp 中提取触发器, 条件和动作信息。由于物联网应用比其他平台(如桌面平台、安卓平台)的应用要小得多, 而且路径数量有限, 因此采用简单的深度优先搜索策略, 不存在状态爆炸的情况。2) 配置信息提取, 给 SmartApp 中植入代码, 在应用程序安装期间获取配置信息。3) 规则建模, 将语义信息和配置信息按照 Homeguard[28]给出的规则模型进行形式化建模。

TAPInspector[35]则是沿用了 IoTCOM[29]提取 ICFG 的方法。

TapChain[33]是一个利用自然语言处理(NLP)方法将 IFTTT 规则转换为语法树的工具。TapChain[33]利用 Stanford CoreNLP 工具对 IFTTT 规则的描述进行分析, 然后从语法树中提取出 12 个不同的特征, 作为 4 个不同分类器的输入进行模型训练。

PSA[34]的组件解析器(The parser)负责将 TAP 程序转换为形式化模型, 其支持对 SmartThings 和 IFTTT 同时进行处理。解析器采用程序的源代码作为输入, 然后生成一组模型, 模型的数量取决于应用

程序中使用的事件处理程序和异步 API 的数量。PSA[34]选择参数化时间自动机(PTA)作为建模抽象来建模智能家居组件, 如 TAP 程序、设备、环境和安全意图(即安全属性)。研究人员给出了从源代码自动生成 PTA 模型的算法。

4.2 漏洞检测

Soteria[30]为单个 app 建立一个包含状态和状态转换的系统模型。提取状态的方法是研究者开发了一个爬虫脚本, 它可以访问 GitHub 中找到的 SmartThings 设备处理程序中的属性和动作代码块, 它将设备的属性值作为状态。然后, 研究者创建了设备功能参考文件, 其中包括每个设备的完整状态集和动作集。最后使用这个文件来识别在应用程序中使用的设备的相关状态。之后 Soteria[30]利用符号执行方法提取状态转换信息, 它会遍历 IR 的路径, 并搜集每条路径上的约束, 以及终止时的状态。对于每个路径, 在状态模型中添加一个从初始状态到结束状态的状态转换, 并通过触发事件处理程序和路径约束的事件来标记该转换。Soteria[30]利用研究者改进的多图联合算法将上面得到的多个单应用状态模型聚合成一个系统模型, 并对这个系统模型进行模型检测。Soteria[30]使用 NuSMV 进行模型检测, 研究者认为它是一个可靠和成熟的模型检测器。Soteria[30]设计安全属性时主要考虑了三个指标: 1) 资产, 是给予重视的单位, 例如车库门。2) 功能要求, 定义系统在正常环境下如何运行, 例如, 当车库门按钮打开, 门打开。3) 功能约束, 资产的使用或动作, 例如只有使用授权的开启设备才能让门打开。最终 Soteria[30]给出了 5 条通用的安全属性和 30 条特定的安全属性, 可以对条件阻塞, 动作冲突, 动作重复和意外规则链进行检测。

IotMon[31]使用研究者提出的算法识别出交互链, 它只能对意外规则链进行检测, 算法需要三个输入信息: 1) 应用内的交互信息, 2) 场景中的物理信道, 3) 场景中的系统信道。应用内的交互信息就是使用一个包含两个元素的元组(触发器, 动作)来表示。物理信道则是通过三个步骤进行提取: 1) 首先, 利用自然语言处理技术从应用描述中提取物理信道实体关键字。2) 使用 Word2Vec 和广泛使用的语言模型来计算提取的关键词的相似度。3) 根据实体关键字的相似性, 对实体关键字进行聚类, 并基于实体关键字聚类物理信道进行识别。IotMon[31]提出有几个系统通道(在三星 SmartThings 平台)可以用来缝合不同的应用内部交互。这些系统通道可以由同一平台上的多个应用程序共享。系统信道包括时间(time)、位置模式(locationMode)、开关(switch)和锁定(lock)。

IotSan[32]将事件处理程序的 Promela 代码、物联网系统的配置信息和安全属性(包括预定义的和用户定义的)作为输入, 使用研究者提出的模型构建算法构建系统的 Promela 模型。接着 IotSan[32]使用 Bandera 工具集中的 Spin 模型检测工具对其进行处理, 当模型检测产生错误跟踪时 Spin 在源代码级别呈现错误跟踪, 并允许用户沿着错误路径浏览代码, 同时显示变量值和 Java 对象的内部状态。IotSan[32]使用基于启发式的算法将违反安全属性的行为归因为错误配置或恶意应用程序。IotSan[32]设计了 7 条通用的安全属性和 38 条特定安全属性, 可以完成对动作冲突, 动作重复和意外规则链的检测。同时值得注意的是, IotSan[32]设计通用安全属性的时候考虑了信息的机密性。

iRuler[27]将物联网建模为一个基于事件(例如, 设备事件和时间事件)的转换系统, 并使用重写逻辑对转换系统进行建模。iRuler[27]使用规则表示、描述用户物联网部署的配置信息以及设备和服务元数据生成转换系统。配置信息和设备元数据可以通过分析物联网平台的文档或由平台开发者或专家提供的文档来构建。对于 IFTTT 平台, iRuler[27]通过爬取每个服务的 web 页面来构造服务元数据, 以获得该服务支持的触发器和动作, 之后再使用 NLP 技术提取服务动作的状态转换。iRuler[27]最后会将转换系统利用重写逻辑表示成元组形式的中间表示 IR。iRuler[27]以 IR 为输入, 使用重写模 SMT 来发现规则间漏洞。重写模 SMT 是一种集重写模理论、SMT 求解和模型检测于一体的符号技术。iRuler[27]支持对条件绕过, 条件阻塞, 动作冲突, 动作重复和意外规则链五种漏洞的检测。

IoTCOM[29]使用有界模型检测器对得到的形式化模型的安全属性进行验证。有界模型检测器的工作依赖三组正式规范: 1) 一个描述组成智能家居环境的一般实体的基本智能家居模型, 整个智能家居系统是一套设备, 一组 IoT 应用程序以及物理信道组成。2) 规则的形式化模型。3) 安全属性。这三个规范均以 alloy 语言规范书写, alloy 是一种简洁且简单的规范语言, 适用于物联网应用程序和安全属性的声明性规范。alloy 包括对传递闭包建模的支持, 有利于分析复杂的链的交互。其次, 它还提供了一个全自动的分析器, 可自动分析出产生闭环威胁的链。IoTCOM[29]定义了 7 条通用的安全属性和 29 条特定的安全属性, 支持对条件绕过, 条件阻塞, 动作冲突, 动作重复和意外规则链五种漏洞的检测。

每当用户安装一个新的应用程序时, Homeguard[28]就会采用约束求解器检测来自新应用程序的每个规则和每个现有规则之间以及新应用程序中不同规则之间的交互漏洞。Homeguard[28]利用基于语义的模型检测方法可以将条件绕过, 条件阻塞, 动作冲突, 动作重复和意外规则链五种漏洞均检测出来。

TAPInspector[35]在得到 TAP 规则的 ICFG 后, 从中提取有效的执行路径, 然后将所得的 TAP 规则转化为有限状态机(FSM)。为了模拟物联网系统的并发性, TAPInspector[35]会将触发器的属性设置为不确定, 这会导致 FSM 状态空间爆炸, 因此研究人员提出了模型切片和状态压缩方法等优化方法。TAPInspector[35]使用 NuSMV 中的 LTL(线性时序逻辑)或 CTL(分支时序逻辑)定义了 9 条通用安全属性和 58 条特定安全属性, 其中特定安全属性又被其分为安全性(37 条)和活性(21 条)两类, 分别保证坏事不会发生和好事最终达成。TAPInspector[35]可以完成对条件阻塞, 动作冲突, 动作重复和意外规则链的检测工作。

TapChain[33]利用训练出的模型直接对 IFTTT 规则进行分类, 从而检测出意外规则链。TapChain[33]评估的数据集包含 5528 对 IFTTT 中手动标记的触发器和动作规则链。评估结果显示, F1 平均为 89.45%。

PSA[34]中的模型生成器(The model generator)会查看应用程序模型, 安全意图库以及设备和环境属性的元数据, 为设备和环境创建模型。它还创建交互模型将设备的动作映射到环境上。最后 PSA[34]将模型送入模型检测器中, 如果与安全意图发生冲突, 模型检测器将输出一个反例。PSA[34]定义了 5 种安全意图, 能够检测出条件阻塞, 动作冲突, 意外规则链三类规则交互漏洞。PSA[34]还输出一组不会导致安全违规的输入值配置。此设置可以帮助用户安全配置应用程序。

表 2 梳理了当前工作中涉及的测试平台和数据集以及检测方案中涉及的关键步骤, 预处理方法总结了提取规则信息的方法, 初级中间表示和中间表示总结了各项工作对 TAP 规则的形式化表示方案, 模型检测列举了各项工作采用的模型检测器和模型检测方案。

表 3 对当前工作的实现效果进行总结, 从三个维度评估当前的工作, 方案是否考虑了物理信道带来的隐蔽交互, 是否考虑了时间对规则交互造成的影响, 以及对典型的规则交互漏洞检测是否适用。

5 基于动态执行监控的 TAP 交互漏洞检测

静态分析方案在形式化验证阶段对智能家居场景使用静态分析技术, 因此可以在违规发生之前检测出规则交互漏洞。还有另一类方案, 它们对智能家居场景进行模拟或实时监控, 从而在 TAP 程序运行时检测规则交互漏洞, 本文将其称为基于动态执行监控的 TAP 交互漏洞检测方案。动态分析方案也需

表 2 基于静态规则分析的 TAP 交互漏洞检测方案

Table 2 TAP interaction vulnerability detection based on static rule analysis

文献	方案名	测试平台	检测的数据集	预处理方法	初级中间表示	中间表示	模型检测
文献[30]	Soteria	SmartThings	64 个 SmartApp	研究者提出的算法	中间代码	元组	模型检测器(NuSMV)
文献[31]	IotMoN	SmartThings	185 个 SmartApp	NLP	形式化通用模型	抽象语法树	K-means clustering
文献[32]	IoSan	SmartThings		依赖图 DG	抽象语法树	Promela	模型检测器(Spin)
文献[27]	iRuler	IFTTT	自定义的规则集	NLP	规则的细粒度表示	元组	模型检测器(Rewriting Modulo SMT)
文献[29]	IoTCOM	SmartThings/IFTTT	从 404 个 Smartapp 和 3328 个 IFTTT applets 中提取的 622 个规则集	规则行为提取器	过程间控制流图	行为规则图	模型检测器(Alloy)
文献[28]	Homeguard	SmartThings	146 个 SmartApp	静态代码分析		形式化的规则表示	基于语义的方法
文献[33]	TapChain	IFTTT	5528 对动作和触发器	NLP		语法树	分类器
文献[34]	PSA	SmartThings/IFTTT		静态代码分析技术(解析首选项, 污点分析)	抽象语法树	PTA	IMITATOR PTA
文献[35]	TAPInspector	SmartThings/IFTTT	1108 个 SmartApp	静态代码分析	过程间控制流图	形式化的延迟敏感 TAP 规则	模型检测器(NuSMV)

表 3 基于静态规则分析的 TAP 交互漏洞检测方案实现效果

Table 3 Implementation effects of tap interaction vulnerability detection based on static rule analysis

文献	方案名	时间敏感	物理信道	条件绕过	条件阻塞	动作冲突	动作重复	意外规则链
文献[30]	Soteria				√	√	√	√
文献[31]	IotMoN	√	√					√
文献[32]	IoSan					√	√	√
文献[27]	iRuler	√		√	√	√	√	√
文献[29]	IoTCOM		√	√	√	√	√	√
文献[28]	Homeguard	√		√	√	√	√	√
文献[33]	TapChain		√					√
文献[34]	PSA	√	√		√	√		√
文献[35]	TAPInspector	√	√		√	√	√	√

要对程序中的规则信息进行提取, 与静态分析方案不同的是动态分析方案后续的处理分为两步: 1) 动态分析, 对运行 TAP 程序的智能家居场景进行动态分析, 获取物联网设备的行为信息。2) 违规判定, 对智能家居场景中设备的行为进行违规判定, 如果 IoT 设备违反安全策略, 则说明当前场景存在规则交互漏洞。这个过程也被称为安全策略验证。策略验证可以借助模型检测器, 此时安全属性就是安全策略, 也可以借助一个安全策略服务器, 此时安全策略由研究人员制定。

接下来从动态分析和违规判定两个角度出发梳理研究中采用的技术。其中涉及到规则信息提取的内容会在动态分析部分阐述。

5.1 动态分析

IotGUARD[36]是一个动态分析系统, 用于在运行时检测和阻止 SmartApp 和 IFTTT 小程序中的不安全行为。IotGUARD[36]会给应用程序的源代码打补丁, 以便收集运行时信息(如设备信息、触发器、动作和状态转换)用于动态分析, 并在每个动作之前添加一个守卫, 以便在不安全的动作被调用之前阻止其执行。给应用程序源代码打补丁的过程分为三个步骤: 1) 它首先识别出应用程序的动作。2) 对于每个动作, 它执行一个基于路径的静态分析来收集触发动作的事件, 动作的路径条件, 以及动作调用中涉及到的数值信息。3) 它在动作之前插入代码, 功能是将动作的信息转发触发。

Iotcheck[37]扩展了 Java Pathfinder(JPF), 这是一种显式的基于状态的模型检测基础设施, 主要用于检测动作冲突。它对 SmartThings 中的应用程序源代码结合配置信息后进行预处理生成模型检查器钩子(model checker hooks), 使 JPF 能够生成设备事件, 将多个应用程序组合到同一个程序中, 并设置运行该程序所需的配置。然后, 它输出检测好的 Groovy 代码, 这些代码由 Groovy 编译器编译成字节码。Iotcheck[37]有一个智能家居仿真框架, 这个框架包含了基准测试应用程序所使用的所有设备的虚拟化设备(即设备处理程序)。当实际的智能家居设备处理程序控制实际的设备时, 虚拟化的设备处理程序会更改表示设备属性值的状态变量的值。

IotSafe[38]是一种用于多应用物联网环境下的动态安全策略执行系统, 以预防性的方式保护用户免受不安全的物联网设备交互, 可有效的检测出意外规则链。研究人员通过使用静态分析和动态测试技术捕获物联网设备之间的真实物理交互。静态分析由代码分析器和交互图构建器组成。IotSafe[38]首先执行代码分析, 提取已部署应用的依赖关系图, 之后给应用程序植入代码, 实现提取用户配置信息以及保证安全策略实施两个目的。然后, 交互图构建器使用特定的用户配置, 如设备 id、触发器条件和房间信息来构建应用程序间交互图。在生成静态交互图之后, IotSafe[38]为每组设备生成测试用例。在动态测试过程中, IotSafe[38]以并行的方式自动测试设备的动作, 并收集传感器的读数。

IoTBox[39]采用沙箱挖掘技术来预防不安全的行为。它拓展了 IoTCOM[29]的工作, 利用 IoTCOM[29]提供的 Alloy 模型, 在沙箱中模拟当前物联网场景中设备的执行情况, 并对相关设备的行为进行观察和记录。

SSRules[40]提出以“State-trigger State-action”(SS)为基础构建面向终端用户易写易改的、具有丰富表达能力的具体 SS 规则范式。SSRules[40]最开始会提供一个友好的用户前端界面让用户编写 TAP 规则, 经过内部处理后在 HA(Home Assistant)上模拟执行 TAP 规则, 它会通过 MQTT 代理连接智能家居真实设备或模拟器。SSRules[40] 需要 HA 中的实体、状态、服务的抽象信息用于 SS 规则的转译和异常检

测,这些信息由 SSRules[40]运行时子系统的抽象信息获取模块来提取和更新。该模块定期从 HA 更新实体信息并检测变化情况,获取设备的动态信息。

IoTSEER[41]用于检测由潜在的物理交互引起的不期望的安全违规行为, 对于隐式意外规则链的检测能力很强。IoTSEER[41]在分析物联网应用程序源代码后从中提取驱动命令和传感器事件, 接着将其转化为一个物理执行模型来定义物理交互, 之后对多个构成交互的物联网应用程序的联合行为构建一个复合物理执行模型, IoTSEER[41]对物联网中的设备进行信息收集从而设置复合物理执行模型的执行参数, 例如它扩展了基于 RSSI 的定位技术, 以获得执行器和传感器之间的物理距离。如果用户安装了一个新的应用程序或设备, IoTSEER[41]会运行相关组件为新的应用程序和设备构建复合物理执行模型。如果应用程序的配置或设备的位置发生变化, IoTSEER[41]也支持实时更改模型的执行参数。

IOTMEDIATOR[42]用于检测和处理多平台系统中的交互威胁, 它利用本地中介器对 IoT 设备和平台之间的原始连接进行隔离和调解, 从而实现对当前物联网场景的实时监控。它能够做到以下三点: 1) 拦截来自物联网设备的所有事件并将其转发到物联网平台; 2) 拦截来自自动化应用程序、配套应用程序和语音命令的所有网络空间命令, 并将其转发给物联网设备; 3) 生成命令并发送到物联网设备。

5.2 违规判定

IotGUARD[36]将收集到的信息保存在一个可变的有向图中, 在运行时接收到事件及其相应的动作时, 安全策略服务器将对有向图使用边缘检测算法并根据一组物联网安全策略对其进行评估。这些策略来自用户和环境的真实需求, 如果一个动作不能满足安全策略, 它将在运行时被守卫停止。IotGUARD[36]制定了 4 条通用安全策略和 32 条特定安全策略, 可以实现对条件阻塞, 动作冲突, 动作重复, 意外规则链四种漏洞的检测, 此外 IotGUARD[36]还支持对出现完整性违反和机密性违反的检测。

Iotcheck[37]内置了一个 JPF 监听器, 它在 Iotcheck[37]仿真框架模拟执行字节码时执行分析, 当检测到冲突时, 使用 JPF 模型检测器检查生成的字节码, 监听器将停止 JPF 并立即报告该冲突。

IotSafe[38]在动态测试后会根据测试数据计算物理模型参数, 以预测未来的设备状态, 当设备状态超出安全策略定义的阈值时发出警告。

IoTBox[39]利用形式化模型和模型检测来确保预期行为与实际执行的一致性。如果两者不一致, IoTBox[39]会向用户发出警告。

SSRules[40]的规则执行有效性检查模块会从 HA 获取所关心的状态变化事件,定期获取全部实体的状态, 检查实体状态及其状态变化是否符合 SS 规则的描述,并将因网络/设备故障等问题导致的设备状态与 SS 规则描述不相符等情况进行告警以及写入日志。SSRules[40]可以有效解决动作冲突, 动作重复和意外规则链等漏洞。

IoTSEER[41]会检查复合物理执行模型的执行结果是否符合指定的安全策略, 并提供违规原因报告指导用户做出修改。IoTSEER[41]利用度量时间逻辑(metric temporal logic)定义了三条基于意图的安全策略和十条特定于设备的安全策略。IoTSEER[41]提出一种网格测试的算法对复合物理执行模型在有限的一组应用程序激活时间(应用程序调用驱动命令的时间)下判定其是否满足安全策略。然而网格测试无法对大量应用程序分析且会因为输入离散化而给出误判, IoTSEER[41]扩展了引导优化的证伪分析技术, 它可以从连续输入集中寻找违反 MTL 策略的反例。

IOTMEDIATOR[42]利用候选筛选和动态验证两步进行违规判定。候选筛选阶段将自动化应用程序定义的所有触发器–条件–动作规则和设备支持的所有设备控制(即命令)作为输入, 之后借助可满足性模理论(Satisfiability Modulo Theories)对其中的规则对进行筛选。IOTMEDIATOR[42]考虑了十二种不同情况下的交互威胁模式, 涵盖了条件阻塞, 条件绕过, 动作冲突, 动作重复, 意外规则链五类规则交互漏洞。如果规则对满足某个漏洞的模式(Pattern), 就将其报告给动态验证组件。对于每种交互威胁模式, IOTMEDIATOR[42]都设计了一个断言序列, 动态验证组件在运行时访问维护的历史事件和命令日志从而对断言序列进行验证, 如果断言序列全部满足, 则违规判定成功。

表 4 梳理了当前工作中使用的测试平台和数据集以及检测方案中涉及的关键步骤, 预处理方法总结了提取规则信息的方法, 初级中间表示和中间表示总结了各项工作对 TAP 规则的形式化表示方案, 动态分析和违规判定总结了各项工作中采用的动态分析技术和判定安全性违规的方法。

表 5 对每项工作的实现效果进行总结, 方案是否考虑了物理信道带来的隐蔽交互, 是否考虑了时间对规则交互造成的影响, 以及对典型的规则交互漏洞检测是否适用。

6 基于用户视角的 TAP 交互规则检测

正如威胁模型中提到的, 规则交互漏洞有时是

表 4 基于动态执行监控的 TAP 交互漏洞检测方案

Table 4 TAP interaction vulnerability detection based on dynamic execution monitoring

文献	方案名	测试平台	检测的数据集	预处理方法	初级中间表示	中间表示	动态分析	违规判定
文献[36]	IotGUARD	SmartThings/ IFTTT	35 个 SmartApp 和 30 个 IFTTT Applet	代码插桩		可变有向图	搜集应用程序运行时信息	边缘检测算法
文献[37]	IotCheck	SmartThings	198 官方+69 第三方 SmartApp	静态代码分析	Groovy 代码	字节码	Iotcheck 仿真框架	模型检测器(JPF)
文献[38]	IotSafe	SmartThings	45 个 SmartApp	静态代码分析与代码插桩	静态交互图	通用物理模型	搜集应用程序运行时信息	预测模型
文献[39]	IotBox	SmartThings/ IFTTT	沿用 IoTCOM 的数据集			沿用 IoTCOM 的 Alloy 模型	利用沙箱技术观察智能家居的行为	模型检测器(Alloy)
文献[40]	SSRules			由用户按模板输入规则	State-State 时序范式	Event-State 时序范式	利用 Home Assistant 模拟执行 TAP 规则	有效性检查
文献[41]	IoTSEER	IFTTT	37 个 IFTTT 小程序	静态代码分析	物理执行模型	复合物理执行模型	收集设备信息给模型设置参数后执行	网格测试, 引导优化的证伪分析技术
文献[42]	IOTMEDIAT -OR	SmartThings, Alexa, IFTTT, Philips Hue, openHAB	两个具体的智能家居场景			规则三元模型	利用本地中介器搜集应用程序运行时的事件和命令	断言序列验证

表 5 基于动态执行监控的 TAP 交互漏洞检测方案实现效果

Table 5 Implementation effects of TAP interaction vulnerability detection based on dynamic execution monitoring

文献	方案名	时间敏感	物理信道	条件绕过	条件阻塞	动作冲突	动作重复	意外规则链
文献[36]	IotGUARD				√	√	√	√
文献[37]	IotCheck					√		
文献[38]	IotSafe	√	√					√
文献[39]	IotBox		√	√	√	√	√	
文献[40]	SSRules	√				√	√	√
文献[41]	IoTSEER		√					√
文献[42]	IOTMEDIATOR	√	√	√	√	√	√	√

用户对 TAP 规则存在认知错误导致的。因此, 如果能够让用户充分理解 TAP 规则对当前智能家居场景带来的影响, 从而制定出安全的 TAP 规则集, 那么也可以有效制止规则交互漏洞的产生。本文把这种站在用户角度解析 TAP 规则, 以发现规则交互漏洞, 称为基于用户视角的 TAP 交互规则检测。文本梳理了其中较具代表性的四项工作。

AutoTap[43]是一个开源系统, 旨在允许用户指定所需的安全属性, 并生成满足这些属性的规则, 从而自动修复不完善或不安全的 TAP 规则。用户可以使用 AutoTap[43]中的“属性规范接口(property-specification interface)”来指定属性。AutoTap[43]提供了 7 个属性模板, 这些模板基于在线调查的结果, 其中包括 71 个物联网用户所需的属性(每个用户 10 个属性)。这些模板定义了形如‘if [state1, ..., staten] should [always] or [never] occur simultaneously’的属性规范。

AutoTap[43]利用用户指定的属性来检查规则逻辑漏洞, 并通过以下三个步骤实现该功能: 1) 将这些属性转化为线性时序逻辑表达式, 然后将由 LTL 规范的属性、TAP 规则和转换系统(由一组状态和与物联网设备相关的事件组成)组合为自动机(图形表示)。该自动机包括了 TAP 规则中涉及的所有物联网设备状态/事件的组合, 例如, “如果下雨, 则打开窗户”规则对应的自动机包含了“RAIN:ON”、“RAIN:OFF”、“Win:ON”和“Win:OFF”四个节点, 其中“RAIN:ON”和“Win:ON”节点之间的边表示可能的状态转换。2) AutoTap[43]采用一种算法来检测违反自动机属性的边缘, 例如, 如果存在阻止窗户打开的属性, 则会发现“RAIN:ON”节点到“Win:ON”节点之间的边。3) AutoTap[43]将适当的 TAP 规则添加

到 Bchi 自动机中, 以使前一步中发现的边无效化。对于上述示例, AutoTap[43]可以向自动机添加一个新的 TAP 规则: “如果下雨时窗户是打开的, 则关闭窗户”, 以替换可能导致属性违反的边。AutoTap[43]允许用户创建自己的安全属性并修复 TAP 规则中的逻辑漏洞。该方法仅支持检测和修复动作冲突。

Safetap[44]是一个基于 web 的工具, 它可以帮助用户检查 TAP 规则的不安全行为, 找出其中的意外规则链。Safetap[44]接受以下输入:TAP 规则、安全属性和系统的当前状态。安全属性使用分支时序逻辑进行形式化表示, Safetap[44]对 TAP 规则用符号模型检测算法进行安全属性违规检测, 当规则不满足某个属性时, Safetap[44]将生成所有违规案例(由违规规则序列组成)作为输出。此外研究人员提出了增量符号模型检查算法, 并将其与 Safetap[44]集成(称之为 SafetapΔ)。SafetapΔ不重新检查已经分析的未更改的规则, 只检查新添加或修改的规则, 这样做有效降低了检测的成本, 且不会反复标记用户已经忽略的违规行为。

EUDebug[45]提出了一种基于 web 界面的方法, 可以帮助用户识别 IFTTT 平台中小程序中可能存在的动作冲突和意外规则链漏洞, 还能进一步帮助用户在调试过程中理解这些错误。EUDebug[45]以触发器和动作的高级语义表示形式将它们添加到 SCPN (Semantic Colored Petri Net)图中, 通过这种形式对触发器和动作进行语义分类。例如将恒温器设置为居家模式和将恒温器设置为 25 摄氏度这两个动作都被归为升温动作, 因为它们都是为了增加室温而执行的动作。通过遍历图, 如果在同一物联网设备或 web 服务上执行的动作或者触发器被分为不同的类别, 就可以识别出冲突的存在。

Zhao 等人[46]从文本语义, 场景结果和安全属性三个维度出发将 TAP 规则之间的不同在 web 界面上向用户呈现, 这些界面的内容由 TAP 程序的形式化分析提供支持, 不仅从语法上比较 TAP 程序, 而且从语义上比较 TAP 程序。通过修改已有的文本对比工具(split diff), 将 TAP 规则中的触发器, 条件, 动作分隔开就可以有效的给出 TAP 规则之间文本语义的差异。之后, 研究人员将每个 TAP 程序和智能家居建模为一个转换系统, 使用宽度优先搜索来识别系统中的可达状态从而给出 TAP 规则场景结果的对比界面, 通过研究人员提供的算法分析转换系统中的不可达节点则可以得到转换系统中变量值的预测结果, 再将其映射到用户便于理解的安全属性形式。经过上述步骤就可以有效的帮助用户理解 TAP 程序, 解决潜在的动作冲突。

7 研究热点与挑战

在对 TAP 交互漏洞检测的研究过程中, Celik 等人[28]提出了系统性的解决方案, 能够识别出规则之间的交互漏洞。Ding 等人首次关注到物理交互带来的安全威胁提出了 IotMon[31], 并对常见的物理交互进行了初步的总结。Wang 等人[27]注意到规则中动作的时长也会给规则交互带来影响, 并对时间进行建模。Ding 等人提出的 IotSafe [38]可以将物理交互和规则中的时间同时进行考量, 从而解决它们带来的交互威胁。随着研究的深入, 后续的一些工作(如 IoTSEER[41])将处理物理交互和时间要素作为研究的一项重点。与此同时, 随着时间要素和物理交互给交互漏洞检测带来的复杂性, 安全属性的制定和表达也引发了研究者的关注。本节对目前 TAP 交互漏洞检测研究中的三个热点与挑战展开阐述。

7.1 考虑物理交互的 TAP 交互漏洞检测

物理信道会带来规则之间隐蔽的交互(在一些文献中称为物理交互), 一条 TAP 规则执行动作命令以后, 另一条规则中的传感器会因检测到受该命令影响的物理信道, 从而被触发。IotMon[31]是最早的对物理信道进行针对性处理的方案, 它总结了在当前智能家居环境中存在的物理信道分别是温度(temperature), 湿度(humidity), 光照强度(illumination), 位置信息(location), 运动(motion), 烟雾(smoke), 水阀泄露(leakage)。分别对应温度传感器, 湿度传感器, 光照度传感器, 门窗传感器, 运动传感器, 烟雾传感器以及水浸传感器。

检测物理交互成为很多安全团队研究的一个重点, 因为它使攻击者能够间接获得对敏感设备的控制, 并将用户和环境置于危险之中。例如在一个场景中有以下两条规则, 一条规则会打开加热器, 另一条规则会在温度超过阈值时打开窗户, 这两条规则通过温度信道进行交互, 攻击者可以通过控制打开加热器的规则实现打开窗户, 并在用户不在家时进入房间。

有许多工作主要集中在通过分析应用程序源代码识别 TAP 规则之间的交互, 仅靠这样的手段是无法检测物理交互的, 因为源代码无法提供相应的物理信道信息。发现物理交互的能力一直很有限。目前物理交互的检测面临三个难点:

(1) 识别出正确的物理交互。为了识别物理交互, 静态分析方案中针对物理信道的处理采用的工作主要分为两种, 一种使用了 NLP 技术, 能够识别出加热器在语义上与温度相关。另一种方案在执行器命

令和传感器事件之间使用预定义的物理信道映射, 例如将开启加热器映射到温度信道[29], 并构建了简单的设备模型, 例如加热器开启后会在 8 小时内使温度升高 1 摄氏度[27]。然而, 这两种方式限制了物理信道的表达能力, 它们会导致物理信道的过拟合从而发出错误的警报(例如, 系统建立映射打开烤箱会提高温度, 但烤箱产生的热量可能不足以触发温度传感器), 以及物理信道的欠拟合, 一些物理交互无法被识别(例如, 系统忽略了扫地机器人的移动会触发运动传感器)。然而静态分析方案中使用的方法不能应用于识别真实的物理相互作用。例如, IotMon[31]使用应用描述的文本挖掘来识别物联网设备之间的公共物理通道, 然后发现设备之间所有潜在的物理交互。因此, 在动态物理交互控制中, 它不能用于检测实时物理交互。采用动态分析方案的系统通过收集运行时设备状态来标识物理交互, 并在运行时强制执行安全策略[38], 然而这会带来接下来的挑战。

(2) 运行时的困境。在运行时检查设备状态的动态系统[18,20]不能推断出一个确切的命令对物理信道的影响。例如, 当运动传感器检测到运动时, 它不能确定这个运动是来自扫地机器人还是人类。当多个设备影响相同的物理信道时, 这个挑战变得更加难以解决。例如, 如果温度传感器能够同时检测到空调和加热器带来的影响, 动态系统不能确定是单个设备还是它们的聚合影响改变了温度。这使得它可能会发现错误的物理交互。当识别到物理交互时, 动态系统要么阻止设备动作, 要么通知用户。然而, 这些动作可能是危险的。例如, 如果在用户不在家的时候, 房子里发生了火灾, 那么开门动作可能会被阻止。为了解决这个挑战, IoTSEER[41]首先通过静态分析从应用程序的源代码中提取执行器命令和传感器事件。由此, 它为命令影响和传感器测量的每个物理信道构建物理执行模型。之后, IoTSEER[41]将多个物理执行模型利用其提出的合成算法统一到一个复合物理执行模型中, 以表示交互应用程序的联合物理行为, 这可以有效区分每个命令的影响。

(3) 缺少对设备放置灵敏度的考量。很多研究没有考虑执行器和传感器之间的距离对物理相互作用的影响[41]。直观地说, 如果执行器和传感器之间的距离增加, 那么执行器对传感器读数的物理影响会减小。因此, 当设备的位置发生变化时, 已识别的交互作用可能不再发生, 可能会出现之前未识别的新的交互。这种观察会导致错误的预测, 包括假阳性(错误的策略违反和不必要的策略执行)和假阴性(遗漏了违反因此未能制止它们)。但是这在智能家居中至关重要, 因为轻便便携的物联网设备可能会发生频繁的设备放置变化。IotSafe[38]和 IoTSEER[41]对此类挑战进行了考虑。IotSafe[38]给出了与物理信道相关的物理模型, 可以预测对应传感器的读数。其中距离会对预测模型的值产生影响。IotSafe[38]使用动态测试过程中收集的数据为每个设备生成离线物理模型, 并估计设备和传感器之间的距离因子。IoTSEER[41]中的物理执行模型在执行命令时需要接收距离参数, 其量化了命令在不同位置的影响程度(例如, 在距离风扇 1 米和 2 米处的声音强度) , 使得其复合物理执行模型能精确模拟应用程序的物理行为。IoTSEER[41]扩展了基于 RSSI 的定位手段, 以获得执行器和传感器之间的物理距离。

7.2 考虑时间要素的 TAP 交互漏洞检测

智能家居平台往往支持用户设置定时规则和延时规则, 定时规则支持在固定的时间点执行 TAP 规则, 例如当前场景中存在规则, 上午 10 点如果温度大于 27 摄氏度就打开窗户。那么只有在 10 点, TAP 程序才会判断当前场景中的温度是否能激活触发器, 从而执行开窗的动作。延时规则支持在用户设置的时间后执行 TAP 规则, 例如当前场景中存在规则, 10 分钟没有检测到移动就关灯。用户可以按照自己的习惯设置延时的时间。由此可见, 一部分时间因素是由于智能家居平台本身的功能所产生的。同时, 动作本身也会具有时间效应, 因为有些动作通常需要一段时间才能改变到期望的值, 而时间取决于特定的物理环境和设备能力, 例如加热器加热或者上传视频, 这两个动作执行时间的长短会对规则交互产生影响, 这被称为动作延时。检测方案是否支持对以上三类情况的处理会导致方案实用性的差异。

然而, 现有的工作有些会忽略时间要素, 有些只是粗略建模, 导致结果并不准确。目前来看, 理想的方案应该支持处理定时规则, 延时规则以及动作延时三种情况。目前将时间因素纳入考量的技术手段如下:

(1) 静态代码分析, 由于定时规则和延时规则都是智能家居平台提供的功能, 因此分析 TAP 程序的源代码可以得到定时和延时的信息。以 SmartThings 平台为例, SmartApp 为定时和延时行为提供了相应的 API。应用程序可以使用 runIn(e, f)调度动作, 其含义是在表达式 e 确定的时间点调用函数 f。或者使用 runEvery(e, f)调度动作, 其含义是在 e 定义的某个持续时间之后重复运行 f。schedule(e, f)则表示每天在 e 决定的时间点运行 f, 周期性地执行动作。因此分析上述 API 可以得到定时和延时信息。在为 TAP 规则

形式化建模时, 将时间信息作为模型的一部分进行建模。

(2) 使用预测模型, 这种方法能解决与物理信道相关的动作延时。对于传感器设备(例如与温度、湿度、烟雾和水位相关的传感器), IotSafe[38]会为其建立一个预测模型, 用来预测其未来某个时刻的值, 从而实现对动作延时的处理。预测模型的参数与环境, 相关器件功率相关。

表 6 考虑时间要素的 TAP 规则漏洞检测

Table 6 TAP rule vulnerability detection considering time factors

方案名称	定时规则	延时规则	动作延时
IotMoN	√		
iRuler	√		
AutoTap	√	√	
IoTIE	√		
Homeguard	√	√	
IotSafe	√	√	√
TAPInspector	√	√	√
PSA	√	√	
SSRules	√		√

7.3 TAP 规则交互检测中的安全属性表达

安全属性本身的质量直接决定对规则交互漏洞检测的效果, 目前的研究将安全属性分为通用安全属性和特定安全属性。通用安全属性主要用于检测 TAP 规则交互漏洞, 因此通用安全属性的数量直接决定检测方案覆盖的规则交互漏洞的种类, 对于同一类规则交互漏洞不同的方案仅存在形式化表述上的差异。值得注意的是, 物理信道和时间因素的介入会使得通用安全属性的制定进一步复杂化。通用安全属性一般都是研究人员设计并给出的。

特定安全属性用于保障智能家居中的设备处于安全状态, 一条特定安全属性的例子是用户不在家时, 窗户应该处于关闭状态。TAPInspector[35]首次提出特定安全属性应该分为安全性和活性, 安全性用于保证坏事不会发生。而通过制定活性可以保证好事总会发生。例如, 活性可以保证正在上传的视频最终被上传到云端。特定安全属性的特点是比较主观, 可以依据经验制定, 因此部分研究会延用之前研究中提出的特定安全属性。且绝大部分的研究特定安全属性也都是研究人员定好的。Safetap[44]让用户可以根据自身需求定制特定的安全属性。这是更加合理的解决方案, 例如一条特定安全属性是晚上十点后, 灯处于关闭状态。对于有些用户十点关灯并不是一个好的选择。为了提高用户的体验, Safetap[44]提供五类特定安全属性的模板。

表 7 安全属性数量制定现状

Table 7 Current status of safety property quantity development

	通用安全属性	特定安全属性
Soteria	5 条	30 条
IoSan	7 条	38 条
IoTCOM	7 条	29 条
IotSafe	14 条	22 条
Safetap	提供五种安全属性模板	
TAPInspector	9 条	37 条安全性 21 条活性
IoTSEER	3 条	10 条
IotGUARD	4 条	32 条

8 结论

对规则交互漏洞的检测是促进智能家居正常发展的关键。本文系统整理了近年来智能家居规则交互漏洞检测研究中的代表性工作, 对漏洞类型和漏洞检测技术进行了总结, 并以此为基础分析了当前面临的挑战和部分应对方案。可以预见到的是, 智能家居中设备的数量还会进一步增加, 为了实现更多元化更实用的功能自动化规则的复杂度和数量也会随之增大, 多个智能家居平台共同管理的场景也会变多, 这都会导致智能家居中的规则交互漏洞的数量和检测难度进一步增大。因此针对智能家居中规则交互漏洞的检测仍然需要更深入的研究。

参考文献

[1] Wang F, Hu L, Zhou J, et al. A Survey from the Perspective of Evolutionary Process in the Internet of Things[J]. *International Journal of Distributed Sensor Networks*, 2015, 2015: 462752.

[2] Liang Y, Cai Z P, Yu J G, et al. Deep Learning Based Inference of Private Information Using Embedded Sensors in Smart Devices[J]. *IEEE Network*, 2018, 32(4): 8-14.

[3] Ury B, McManus E, Ho M P Y, et al. Practical Trigger-Action Programming in the Smart Home[J]. *Conference on Human Factors in Computing Systems - Proceedings*, 2014: 803-812.

[4] Rahmati A, Fernandes E, Jung J, et al. IFTTT vs. Zapier: A comparative study of trigger-action programming frameworks[J]. arXiv preprint arXiv:1709.02788, 2017.

[5] Cai Z P, Zheng X. A Private and Efficient Mechanism for Data Uploading in Smart Cyber-Physical Systems[J]. *IEEE Transactions on Network Science and Engineering*, 2020, 7(2): 766-775.

[6] Ur B, Pak Yong Ho M, Brawner S, et al. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes[C]. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 2016: 3227-3231.

[7] IFTTT. Accessed: Jun. 1, 2023. [Online]. Available: https://ifttt.com.

[8] SmartThings. Accessed: Jun. 1, 2023. [Online]. Available: https://www.smartthings.com.

[9] Zapier. Accessed: Jun. 1, 2023. [Online]. Available: https://zapier.com/.

[10] openHab. Accessed: Jun. 1, 2023. [Online]. Available: https://www.openhab.org.

[11] Apple HomeKit. Accessed: Jun. 1, 2023. [Online]. Available: https://www.apple.com/ios/home.

[12] Mi Home. Accessed: Jun. 1, 2023. [Online]. Available: https://home.mi.com/.

[13] Ronen E, Leung D, Mishra P, et al. Smart locks: Lessons for securing commodity internet of things devices[C]. *Proceedings of the 11th ACM on Asia conference on computer and communications security*, 2016: 461-472.

[14] Ronen E, Shamir A. Extended Functionality Attacks on IoT Devices: The Case of Smart Lights[C]. *2016 IEEE European Symposium on Security and Privacy*, 2016: 3-12.

[15] Kolias C, Kambourakis G, Stavrou A, et al. DDoS in the IoT: Mirai and Other Botnets[J]. *Computer*, 2017, 50(7): 80-84.

[16] Zhang G, Yan C, Ji X, et al. Dolphinattack: Inaudible voice commands[C]. *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017: 103-117.

[17] Wang X, Sun Y, Nanda S, et al. Looking from the Mirror: Evaluating {IoT} Device Security through Mobile Companion Apps[C]. *28th USENIX security symposium,* 2019: 1151-1167.

[18] Meneghello F, Calore M, Zucchetto D, et al. IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices[J]. *IEEE Internet of Things Journal*, 2019, 6(5): 8182-8201.

[19] Mao J, Liu Z W, Lin Q X, et al. Semantic-Fuzzing-Based Empirical Analysis of Voice Assistant Systems of Asian Symbol Languages[J]. *IEEE Internet of Things Journal*, 2022, 9(12): 9151-9166.

[20] Fouladi B, Ghanoun S. Honey, I'm home!!, hacking zwave home automation systems[J]. *Black Hat USA*, 2013.

[21] Ronen E, Shamir A, Weingarten A O, et al. IoT goes nuclear: Creating a ZigBee chain reaction[C]. *2017 IEEE Symposium on Security and Privacy, IEEE*, 2017: 195-212.

[22] Jia Y, Xing L Y, Mao Y H, et al. Burglars' IoT Paradise: Understanding and Mitigating Security Risks of General Messaging Protocols on IoT Clouds[C]. *2020 IEEE Symposium on Security and Privacy*, 2020: 465-481.

[23] Wang Q, Ji S, Tian Y, et al. {MPInspector}: A systematic and automatic approach for evaluating the security of {IoT} messaging protocols[C]. *30th USENIX Security Symposium,* 2021: 4205-4222.

[24] Fernandes E, Jung J, Prakash A. Security Analysis of Emerging Smart Home Applications[C]. *2016 IEEE Symposium on Security and Privacy*, 2016: 636-654.

[25] Yuan B, Jia Y, Xing L, et al. Shattered Chain of Trust: Understanding Security Risks in {Cross-Cloud}{IoT} Access Delegation[C]. *29th USENIX security symposium,* 2020: 1183-1200.

[26] Zheng X, Cai Z P, Li Y S. Data Linkage in Smart Internet of Things Systems: A Consideration from a Privacy Perspective[J]. *IEEE Communications Magazine*, 2018, 56(9): 55-61.

[27] Wang Q, Datta P, Yang W, et al. Charting the Attack Surface of Trigger-Action IoT Platforms[C]. *The 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019: 1439-1453.

[28] Chi H T, Zeng Q, Du X J, et al. Cross-App Interference Threats in Smart Homes: Categorization, Detection and Handling[C]. *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2020: 411-423.

[29] Alhanahnah M, Stevens C, Bagheri H. Scalable analysis of interaction threats in iot systems[C]. *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, 2020: 272-285.

[30] Celik Z B, McDaniel P, Tan G. Soteria: Automated {IoT} Safety and Security Analysis[C]. *2018 USENIX Annual Technical Conference,* 2018: 147-158.

[31] Ding W, Hu H. On the safety of iot device physical interaction control[C]. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018: 832-846.

[32] Nguyen D T, Song C, Qian Z, et al. IotSan: Fortifying the safety of IoT systems[C]. *Proceedings of the 14th International Conference on emerging Networking Experiments and Technologies*, 2018: 191-203.

[33] Jiang K Y, Zhang H Y, Zhang W T, et al. TapChain: A Rule Chain Recognition Model Based on Multiple Features[J]. *Security and Communication Networks*, 2021, 2021(1): 6568602.

[34] Kashaf A, Sekar V, Agarwal Y. Protecting Smart Homes from Unintended Application Actions[C]. *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems*, 2022: 270-281.

[35] Yu Y B, Liu J J. TAPInspector: Safety and Liveness Verification of Concurrent Trigger-Action IoT Systems[J]. *IEEE Transactions on Information Forensics and Security*, 2022, 17: 3773-3788.

[36] Celik Z B, Tan G, McDaniel P. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT[C]. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.

[37] Trimananda R, Aqajari S A H, Chuang J, et al. Understanding and Automatically Detecting Conflicting Interactions between Smart Home IoT Applications[C]. *The 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020: 1215-1227.

[38] Ding W, Hu H, Cheng L. IOTSAFE: Enforcing safety and security policy with real IoT physical interaction discovery[C]. *the 28th Network and Distributed System Security Symposium,* 2021.

[39] Kang H J, Sim S Q, Lo D. IoTBox: Sandbox Mining to Prevent Interaction Threats in IoT Systems[C]. *2021 14th IEEE Conference on Software Testing, Verification and Validation*, 2021: 182-193.

[40] Wang B, Zhang Y, Geng J N, et al. SSRules: Make It Easier to Write and Check Automation Rules for Smart Home Systems[J]. *Journal of Software*, 2021, 32(12): 3728-3750.
(王博, 张昱, 耿佳宁, 等. SSRules: 让智能家居自动化规则更易于编写和检查[J]. *软件学报*, 2021, 32(12): 3728-3750.)

[41] Ozmen M O, Li X, Chu A, et al. Discovering IoT physical channel vulnerabilities[C]. *Proceedings of the 2022 ACM SIGSAC Confer-*

ence on Computer and Communications Security, 2022: 2415-2428.

[42] Chi H, Zeng Q, Du X. Detecting and Handling {IoT} Interaction Threats in {Multi-Platform}{Multi-Control-Channel} Smart Homes[C]. *32nd USENIX Security Symposium,* 2023: 1559-1576.

[43] Zhang L F, He W J, Martinez J, et al. AutoTap: Synthesizing and Repairing Trigger-Action Programs Using LTL Properties[C]. *2019 IEEE/ACM 41st International Conference on Software Engineering*, 2019: 281-291.

[44] McCall M K, Shezan F H, Bichhawat A, et al. SAFETAP: An Efficient Incremental Analyzer for Trigger-Action Programs[J]. 2021.

[45] Corno F, De Russis L, Monge Roffarello A. Empowering end users in debugging trigger-action rules[C]. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems,* 2019: 1-13.

[46] Zhao V, Zhang L F, Wang B, et al. Understanding Trigger-Action Programs through Novel Visualizations of Program Differences[C]. *The 2021 CHI Conference on Human Factors in Computing Systems*, 2021: 1-17.

[47] Mi X H, Qian F, Zhang Y, et al. An Empirical Characterization of IFTTT: Ecosystem, Usage, and Performance[C]. *The 2017 Internet Measurement Conference*, 2017: 398-404.

[48] Tian Y, Zhang N, Lin Y H, et al. SmartAuth: User-Centered Authorization for the Internet of Things[C]. *USENIX Security Symposium*, 2017, 5(1): 8.2.

[49] Jia Y J, Chen Q A, Wang S, et al. ContexIoT: Towards providing contextual integrity to appified IoT platforms[C]. NDSS, 2017, 2(2): 2.2.

[50] He W, Golla M, Padhi R, et al. Rethinking Access Control and Authentication for the Home Internet of Things[C]. *27th USENIX Security Symposium,* 2018: 255-272.

[51] Chi H, Zeng Q, Du X, et al. PFirewall: Semantics-aware customizable data flow control for smart home privacy protection[J]. arXiv preprint arXiv:2101.10522, 2021.

[52] Zheng X, Cai Z P. Privacy-Preserved Data Sharing towards Multiple Parties in Industrial IoTs[J]. *IEEE Journal on Selected Areas in Communications*, 2020, 38(5): 968-979.

[53] Moustafa N, Turnbull B, Choo K R. An Ensemble Intrusion Detection Technique Based on Proposed Statistical Flow Features for Protecting Network Traffic of Internet of Things[J]. *IEEE Internet of Things Journal*, 2019, 6(3): 4815-4830.

[54] Cook A A, Mısırlı G, Fan Z. Anomaly Detection for IoT Time-Series Data: A Survey[J]. *IEEE Internet of Things Journal*, 2020, 7(7): 6481-6494.

[55] Han G J, Tu J T, Liu L, et al. Anomaly Detection Based on Multidimensional Data Processing for Protecting Vital Devices in 6G-Enabled Massive IIoT[J]. *IEEE Internet of Things Journal*, 2021, 8(7): 5219-5229.

[56] Fu C, Zeng Q, Du X. {HAWatcher}:{Semantics-Aware} Anomaly Detection for Appified Smart Homes[C]. *30th USENIX Security Symposium,* 2021: 4223-4240.

[57] Li S C, Choo K R, Sun Q D, et al. IoT Forensics: Amazon Echo as a Use Case[J]. *IEEE Internet of Things Journal*, 2019, 6(4): 6487-6497.

[58] Babun L, Sikder A K, Acar A, et al. The Truth shall Set Thee Free: Enabling Practical Forensic Capabilities in Smart Environments[C]. *Proceedings 2022 Network and Distributed System Security Symposium*, 2022.

[59] Wang Q, Hassan W U, Bates A, et al. Fear and Logging in the Internet of Things[C]. *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.

[60] Oswald E. IFTTT competitor Stringify gets a major update[J]. *TechHive*, 2016, 22.

[61] Home assistant: Awaken your home. 2019. https://www.home- assistant.io/.

[62] Brice.Announcing SmartRules 2.0. 2016. http://smartrulesapp.com/2016/04/26/announcing-smartrules-2-0/.

[63] Huang J, Cakmak M. Supporting Mental Model Accuracy in Trigger-Action Programming[C]. *The 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015: 215-225.

[64] Brackenbury W, Deora A, Ritchey J, et al. How Users Interpret Bugs in Trigger-Action Programming[C]. *The 2019 CHI Conference on Human Factors in Computing Systems*, 2019: 1-12.

[65] Palekar M, Fernandes E, Roesner F. Analysis of the Susceptibility of Smart Home Programming Interfaces to End User Error[C]. *2019 IEEE Security and Privacy Workshops* (*SPW*), 2019: 138-143.

[66] Baier C, Katoen J P. Principles of model checking[M]. MIT press, 2008.

[67] Pnueli A. The Temporal Logic of Programs[C]. *18th Annual Symposium on Foundations of Computer Science*, 1977: 46-57.

[68] Clarke Jr E M, Grumberg O, Kroening D, et al. Model checking[M]. MIT press, 2018.

[69] Holzmann G J. The SPIN model checker: Primer and reference manual[M]. Reading: Addison-wesley, 2004.

[70] Cimatti A, Clarke E, Giunchiglia F, et al. NuSMV: A New Symbolic Model Verifier[C]. *Computer Aided Verification*, 1999: 495-499.

[71] Jackson D. Alloy[J]. *ACM Transactions on Software Engineering and Methodology*, 2002, 11(2): 256-290.

[72] Fernandes E, Rahmati A, Jung J, et al. Decentralized Action Integrity for Trigger-Action IoT Platforms[C]. *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.

[73] Celik Z B, Babun L, Sikder A K, et al. Sensitive information tracking in commodity IoT[C]. *27th {USENIX} Security Symposium,* 2018: 1687-1704.

[74] Zhang W, Meng Y, Liu Y G, et al. HoMonit: Monitoring Smart Home Apps from Encrypted Traffic[C]. *The 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018: 1074-1088.

[75] Chen X Y, Zhang X L, Elliot M, et al. Fix the Leaking Tap: A Survey of Trigger-Action Programming (TAP) Security Issues, Detection Techniques and Solutions[J]. *Computers & Security*, 2022, 120: 102812.

王靖尧 于2021年在武汉大学信息安全专业获得学士学位。现在武汉大学网络空间安全专业攻读硕士学位。CCF 会员: R7440G。研究领域为物联网安全。Email: 2017301500285@whu.edu.cn

王婷婷 于 2003 年在解放军信息工程大学计算机科学技术与应用专业获得学士学位, 现任解放军北部战区海军高工, 研究领域为信息安全。研究兴趣包括: 系统安全防护、密码理论等。Email: wtt19790214@163.com

陈泽茂 于 2005 年在海军工程大学电力系统及其自动化专业获得博士学位。现任武汉大学国家网络安全学院教授。研究领域为系统安全与可信计算。研究兴趣包括: 信息物理系统安全、软件安全。Email: chenzemao@whu.edu.cn