SnIPU zz：基于消息片段推理

冯晓涛†，孙若曦†，朱晓刚†‡，薛敏辉†，文胜†，刘东喜‡，苏里亚·尼帕尔‡，杨翔†
澳大利亚斯威本科技大学
澳大利亚阿德莱德大学
‡CSIRO Data61，澳大利亚

摘要

物联网（IoT）设备的激增使人们 由于难以获得和仿真物联网固件，在缺乏内部执行信息的情况下，对物联网设备进行黑盒模糊处理成为一种可行的选择。然而，现有的黑盒模糊器不能形成有效的变异优化机制来指导他们的测试过程，主要是由于缺乏反馈。此外，由于IoT设备中普遍使用各种非标准通信消息格式，因此很难甚至不可能应用现有的基于语法的模糊策略。因此，在IoT模糊化领域中需要具有语法推断的高效模糊化方法。

为了解决这些关键问题，我们提出了一种新的自动黑盒模糊物联网固件，称为Sn I p UZZ。Snipuzz作为与设备通信的客户端运行，并基于响应推断用于变化的消息片段。每个代码段引用一个连续的字节块，这些字节块反映模糊处理中的近似代码覆盖率。这种基于消息片段的变异策略大大缩小了改变探测消息的搜索空间。 我们将SNIPUZZ与四种最先进的物联网模糊方法进行了比较，*即*、IoTFU zzeR、BooFU zz、DoonA和Nemesys。Snipuzz不仅继承了基于应用的模糊的优点（*例如，*、IoTF（使用），而且还利用通信响应来执行有效的突变。此外，SNIPUZZ是轻量级的，因为其执行不依赖于任何先决条件操作，诸如应用的逆向工程。我们还在20个流行的现实世界物联网设备上评估了Sn i p uzz。我们的研究结果表明，Sn I p uzz可以识别5个零日漏洞，其中3个*只能*通过Sn I p UZZ暴露。 所有新发现的漏洞都已得到其供应商的确认。

ACM参考格式：
Xiaotao Feng， Ruoxi Sun， Xiaogang Zhu， Minhui Xue， Sheng Wen， Dongxi Liu， Surya Nepal， and Yang Xiang.2021 年 。 SNI PUZZ：通过消息片段推断对IoT固件进行黑盒模糊*2021*年

CCS 2021，2021年11月14 - 21日，韩国首尔
©2021计算机协会ACM ISBN 978-1-4503-XXXX-X/18/06。. . 15美元
https://doi.org/10.1145/1122445.1122456
虚拟事件，韩国。ACM，New York，NY，USA，15页。https：//doi.org/10.1145/1122445.1122456

1 引言

物联网（IoT）是指现在连接到互联网的全球数十亿个物理设备，所有这些设备都收集和共享数据。早在2017年，物联网设备的数量就超过了世界人口[39]，到2020年，这个星球上的每个人平均拥有四个物联网设备[23]。虽然这些设备丰富了我们的生活和行业，但不幸的是，它们也以漏洞的形式引入了盲点和安全风险我们以Mirai [25]为例。Mirai是物联网僵尸网络恶意软件中最突出的类型之一2016年，Mirai在分布式拒绝服务（DDoS）活动中关闭了广泛使用的网站，该活动由数千个受感染的家庭物联网设备组成。在Mirai的案例中，攻击者利用漏洞来攻击物联网设备本身，然后将设备武器化，用于更大的活动或将恶意软件传播到网络。事实上，攻击者还可以利用易受攻击的设备进行横向移动，从而使他们能够到达关键目标。例如，在COVID-19期间的在家工作场景中，趋势科技报告称，将易受攻击的物联网设备引入家庭将使员工暴露于恶意软件和攻击，这些恶意软件和攻击可能会进入公司的网络[26]。考虑到物联网设备的无处不在，我们认为这些已知的安全事件和风险场景只不过是冰山一角。

物联网漏洞通常是关于设备固件中 为了尽快推出新产品，开发人员总是倾向于在固件开发中使用开源组件，而没有良好的更新计划[1]。这牺牲了物联网设备的安全性，并使它们暴露在安全团队无法快速修复的漏洞 即使供应商计划修复其产品中的漏洞，无线补丁通常也是不可行的，因为物联网设备没有可靠的网络连接[16]。因此，据报道，市场上一半的物联网设备都存在漏洞[28]。

因此，在攻击者之前发现这些漏洞并修复它们至关重要。然而，大多数物联网软件安全测试严重依赖于设备固件可用性的假设。在许多情况下，制造商倾向于不发布其产品固件，这使得基于代码分析[7，13，15，18，32，46]（或仿真[8，10，20，50，51]）的各种动态分析方法变得困难。 在现有的防御技术中，模糊测试已经显示出克服这些问题的承诺，并已被广泛使用，作为一种有效的方法，在发现漏洞。而且，物联网设备与外界通信的能力

为我们提供了一种新的选择，那就是通过交换网络消息来测试设备固件因此，IoT模糊器可以被设计为向目标设备发送随机通信消息，以便检测它是否显示出任何故障症状。如果在执行过程中触发崩溃或设备被推送回异常消息，可能会暴露出潜在的漏洞。

然而，使用网络通信来模糊IoT设备的固件是非常具有挑战性的。由于不可能从设备获得内部执行信息，因此大多数现有的网络IoT模糊器[9，31，44]以黑盒方式工作这使得优化突变策略非常困难。由于突变种子的选择是完全随机的，因此现有的黑盒物联网模糊方法可能会变得非常难以处理，有时甚至更像是蛮力破解测试。此外，IoT设备对于通信中的输入具有严格的语法规范。由随机突变生成的大多数消息将破坏输入的语法规则，并且在执行之前在固件中的语法验证期间将被快速拒绝。基于语法的变异策略[2，40]可以有效地生成满足输入要求的消息。这可以通过经由记录的语法规范或从标记的训练集学习语法来完成。然而，如表1所示，许多非标准IoT设备通信格式正在实践中使用因此，为基于语法的变异策略准备足够的学习材料是一个巨大的工作量，这使得基于语法的物联网模糊的部署变得困难。

挑战。在本文中，我们专注于通过向物联网设备发送消息来检测物联网固件中的漏洞。为了设计有效且高效的模糊化方法，必须克服若干挑战。

. *挑战1：缺乏反馈机制*。在不访问固件的情况下，几乎不可能从IoT设备获得内部执行信息以指导模糊处理（如在大多数典型的模糊器中所做因此，我们需要一个轻量级的解决方案来获取设备的反馈，并优化生成过程.
. *挑战2：不同的消息格式*。表1显示了物联网通信中使用的一些消息格式，包括JSON、SOAP、键值对、字符串甚至是自定义格式。为了应用于各种设备，解决方案应该能够从原始消息中推断出格式。
. *挑战3：反应的随机性IoT设备*的响应消息可以包含随机元素，诸如时间戳或令牌。这样的随机性导致对相同消息的不同响应，并且降低了模糊化的有效性，因为SNIPUZZ的输入生成依赖于响应。

我们的方法。在本文中，我们提出了一种新的和自动的黑盒物联网模糊，命名为Sn I p UZZ，检测物联网固件中的漏洞。与其他现有的IoT模糊方法不同，SNIPUZZ实现了基于片段的变异策略，该策略利用来自IoT设备的反馈来指导模糊。具体地，Sn I p uzz使用新颖的启发式算法来检测消息中的每个字节的角色。它将首先对消息中的字节逐个进行变异

一个用于生成探测消息，并对相应的探测消息进行分类。
从设备收集响应具有

表1：IoT设备的格式要求

#	设备类型	卖主	机型	固件版本	格式设置
1	智能灯泡	Yeelight	YLDP05YL	1.4.2_0016	JSON
2	智能灯泡	Yeelight	YLDP13YL	1.4.2_0016	JSON
3	智能灯泡	飞利浦	公司简介	1.46.13_r26312	JSON
4	智能灯泡	LIFX	迷你C	v3.60	自定义字节
5	智能灯泡	泛光灯	公司简介	35.V7.63.7189-A	自定义字节
6	家桥	飞利浦	顺化	1935144040	JSON
7	家桥	阿尔罗	基站	1.12.2.8_9_fc4b603	JSON
8	智能插头	特普林克	公司简介	1.5.2	JSON
9	智能插头	特普林克	公司简介	1.5.2	JSON*
10	智能插头	贝尔金WeMo	F7C027au	2.00.1821	SOAP
11	智能插头	梅罗斯	MSS310	2.1.14	JSON*
12	智能插头	奥尔维博	B25AUS	v3.1.3	JSON
13	智能插头	孔凯	迷你美国	简体中文	字符串
14	智能插头	Broadlink	SP4L-AU	v57209	自定义字节
15	路由器	网件	R6400	1.0.1.46	SOAP*
16	助教	ZKteco	WL10	ZLM-FX1-3.0.23	自定义字节
17	摄像机	阿尔罗	Alro Pro 2	1.125.14.0_34_1189	JSON*
18	摄像机	福斯卡姆	F19821W	2.21.1.127	JSON*
19	NAS	QNAP	T-131P	4.3.6.0959	键值对
20	通用遥控器	BroadLink	RM mini 3	v44057	自定义字节

*：在响应中具有随机性。

消息中的相同角色形成初始消息片段，这是突变的基本单位此外，SNIPUZZ利用分层聚类策略来优化变异策略并减少由响应消息中的随机性和固件的内部机制引起的类别的误分类。因此，Sn I p UZZ作为黑盒模糊器，在没有语法规则和设备内部执行信息支持的情况下，仍然可以有效地测试物联网设备的固件。

SNIPUZZ通过使用响应作为引导来优化模糊处理来解决挑战1。基于响应，Sn IPUZZ设计了一种新颖的启发式算法来初步推断消息中每个字节的角色，这解决了**挑战2**。Snipuzz利用编辑距离[42]和凝聚层次聚类[43]来解决**挑战3**。我们总结我们的主要贡献如下：

. **消息片段推理机制。**来自IoT设备的响应与固件中的代码执行路径有关。基于响应，我们推断消息片段和固件中的代码执行路径之间的关系这种新颖的突变机制使得Snipuzz不需要任何语法规则来通过设备响应推断输入的隐藏语法结构Snipuzz提出的片段判定方法与实际的句法规则相比，相似度为87.1%.
. **更有效的物联网模糊。**测试IoT设备时，响应类别的数量与固件中代码执行路径的数量呈正相关。在实验中，无论分析持续时间多长（10分钟或24小时），Sn i puzz探索的响应类别的数量都远远超过大多数设备上的其他方法。
. **实施和漏洞发现。**我们实现了Sn IPUZZ的原型。[1]我们使用它来测试20个真实世界的消费级物联网设备，同时与最先进的模糊工具进行比较，*即*、IoTFU zzeR、DoonA、BooFU zz和Nemesys。在20台设备中的5台中，SnI p uzz成功发现了5个零日漏洞，包括空指针异常、拒绝

[1]可在www.example.com上公开获取https://github.com/XtEsco/Snipuzz。

的服务，和未知的崩溃，其中3个可能会暴露
只有我才知道。

2 二.背景

2.1 模糊测试

Fuzzing是一个强大的自动化测试工具，用于检测软件漏洞。经过几十年的发展，模糊技术已被广泛用作几个安全测试领域的基础，如操作系统内核[12，36]，服务器[33]和区块链[3]。

一般来说，模糊处理向目标程序提供大量变异的输入，并监视异常（*例如：*崩溃）。如果执行揭示了不期望的行为，则可以检测到漏洞为了更有效地发现漏洞，模糊算法基于执行的反馈（*例如，*覆盖知识），而不是使用纯随机突变策略。此外，模糊器可以从反馈机制中判断种子变异生成的每个测试用例是否"有趣"（即，是否是"有趣"的）。测试用例是否已经探索了看不见的执行状态）。如果一个测试用例是有趣的，它将被保留为一个新的种子，以参与未来的变异。通过反馈，许多模糊器[4，5，29，41，49]将计算资源转向感兴趣的测试用例，并实现更高的发现漏洞的可能性。

2.2 IoT设备的通用通信架构

为了对外部输入做出反应，大多数物联网设备都实现了类似的高级通信架构。根据图1中呈现的伪代码示例，通信架构的典型实现可以由四个部分组成:

2）功能开关，3）功能定义，和4）应答器。

当物联网设备接收到外部输入时，Sanitizer会开始解析输入并执行常规匹配。 如果-mat的输入违反语法要求，或在解析过程中发生异常，Sanitizer将通过发送描述输入错误的响应消息直接通知Replier，并终止输入的处理。 如果输入在语法上是正确的，则Function Switch根据从输入中提取的属性Key和相应的值val将控制转移到相应的Functions。如果Key无法匹配，则此输入的处理将终止，与Replier所做的类似当Functions使用参数val完成处理（如setFlow（））时，它通知Replier生成响应消息。注意，功能的实现特定于IoT设备。如上所述，Replier负责向客户端（例如用户根据调用位置（由示例中的参数代码指示），Replier确定要发送的响应消息的内容

3 动机

3.1 基于响应的反馈机制

IoT设备的交互能力使得可以通过网络测试设备固件的安全性然而，在使用基于网络的模糊器测试物联网设备时也存在一些挑战。由于大多数网络模糊方法不能直接获得设备的执行状态，因此难以建立有效的网络模糊模型。

反馈机制来指导模糊化过程。在没有反馈机制的情况下，模糊测试在突变靶标的选择中可能是盲目的，并且可能倾向于蛮力随机测试。

如前所述，由于缺乏开源固件，很难甚至不可能对IoT设备进行仪表化因此，由固件返回的响应消息可以被视为运行时设备状态信息的有价值的来源。图1中的Replier将使用变量code的值来确定响应消息的内容。代码的价值来自固件中许多不同的功能块。当Sanitizer无法解析输入或触发某些异常时;或当功能开关无法匹配输入中的关键命令字符时;或在功能中执行每个输入后，因此，通过响应消息的内容，可以推断出已经在固件中执行的代码块当固件源代码不可用时，不能直接提取固件执行与响应消息之间的对应关系。此外，即使执行不同的功能，固件也可以返回相同的响应消息。

虽然响应消息不能等同于设备的执行路径，但它仍然可以在IoT设备的黑盒模糊测试中发挥重要作用 虽然很难链接对应于每个响应消息的代码执行路径，但是如果两个输入得到不同的响应消息，则可以推断出两个输入去往不同的固件代码执行路径。**我们的方法。** Sn IPUZZ使用响应消息来建立新的反馈机制。SnIpuzz将收集每个响应，并且当发现新响应时，对应于该响应的输入将被排队作为用于后续突变测试的种子

3.2 消息段推断

IoT设备的固件可以被认为是对输入具有严格语法要求的软件程序 如果在模糊测试中使用基于字节的变异策略（例如逐个变异输入中的每个字节或随机选择字节进行变异测试），则生成的测试用例可能很少以满足输入语法要求。基于语法的模糊器利用详细的文档或大型训练数据集来学习语法规则并使用它来指导突变的生成[34，40]。在许多情况下，IoT设备中的输入语法是多样的或非标准的。 表1显示了来自不同供应商的20个物联网设备中使用的通信格式要求。它们中的一些使用众所周知的格式，如JSON和SOAP，但有些使用键值对甚至自定义字符串作为通信格式。因此，很难为基于语法的变异策略提供语法规范或建立大规模覆盖通信格式的训练数据集。

*最好的语法指导来自固件本身。*来自IoT设备的响应建议消息的执行结果。 如果我们一个字节一个字节地改变一个有效的消息（*即*，违反格式），我们会得到很多不同的回应。如果有效消息中的两个不同位置的突变接收到相同的响应，则这两个位置具有它们与固件中的相同功能相关的高可能性因此，具有相同响应的那些连续字节可以合并成一个片段。这

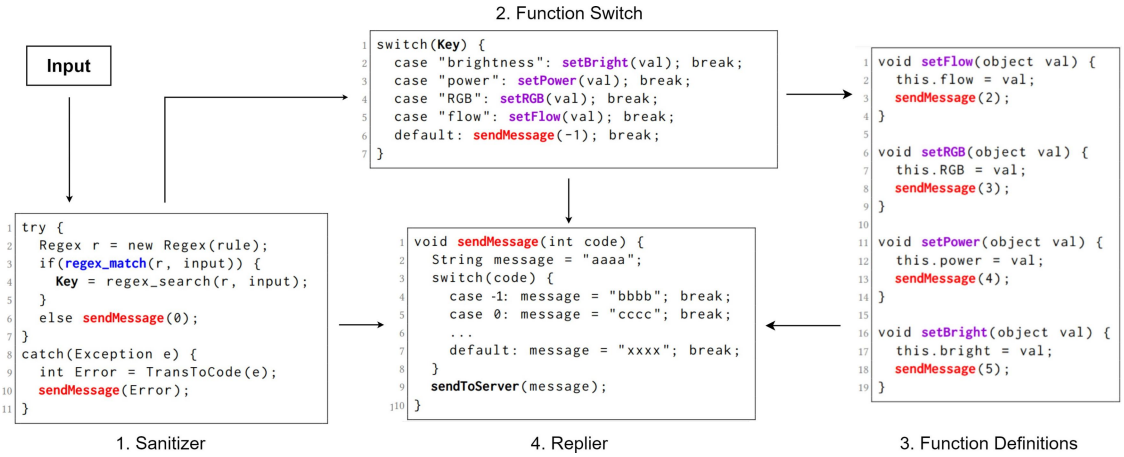

图1：与IoT固件的交互。物联网设备的大多数实现都具有类似的通信架构，包括消毒器、功能交换机、功能定义和应答器。如果消毒剂和功能开关运行正常，将执行相应的除了崩溃之外，Replier将始终向客户端发送响应。

推断消息片段的方法可以清楚地反映每个字节在进入固件之后的效用另外，基于消息片段的变异算法可以大大减少搜索空间，提高模糊搜索的效率。

我们的方法。 Sn I p uzz将具有相同响应的连续字节合并到一个片段中。我们还提出了不同的突变算子执行的片段。

4 方法论

为了清楚地呈现我们的方法，我们首先在解释Sn IPUZZ的模糊化过程的同时引入一些符号。在高级别上，SNIPUZZ作为发送消息序列的客户端***执行***

向IoT设备请求某些操作。任何消息 都请求IoT设备执行特定功能，并且所有消息 = 一起工作以请求动作（或响应）。

行动）。与典型的模糊器类似，我们用初始消息序列初始化***种子***，并用所有种子初始化种子语料库 （4.1节）。同时，***收集恢复消息序列***以用于将IoT设备重置到预定义状态。

为了建立有效的模糊化，如图2所示，SNIPUZZ首先进行片段确定过程。具体地，SNIPUZZ选择种子中的消息 ，从该种子中生成***探测消息。***

并且将生成相应的序列 。每个message-sage 将触发一个***响应消息*** （简称响应），其中包含有关执行输出的信息。SNIPUZZ为每个消息分配***响应池***，该响应池用于确定新的响应是否是唯一的。 响应的唯一性表示它不属于响应池中存在的任何类别的响应。如果 是唯一的，则Sn IPUZZ将添加 到池中，并保留相应的消息序列 作为新的种子。然后，Sn IPUZZ根据响应将消息 划分为不同的片段（第4.2节）。在获得片段后，SNIPUZZ根据各种策略进行突变，*例如：*、空、字节翻转、数据边界或破坏（详细信息请参见

第4.3节）。在整个模糊过程中，Sn I p uzz设置网络监视器以检测可能指示漏洞的崩溃（第4.4节）。

4.1 消息序列获取

初始种子的质量对起毛作业有因此，我们考虑获得符合物联网设备所需的高度结构化格式的高质量初始种子，因为这些输入可能会执行复杂的执行路径，并扩大暴露深层代码漏洞的机会基于配套应用程序逆向工程[9]或可访问的规范（如第3.2节所述）生成种子可能是直观的解决方案。然而，它们要么需要大量的工程工作，要么可能容易出错（*例如*，种子可能违反所需的格式或具有错误的消息顺序）。

初始种子获取。Snipuzz提出了一种轻量级的解决方案来获得初始有效种子。考虑到许多物联网设备都有第一方或第三方API文档以及测试套件，双方提供的测试程序可以有效地充当客户端，向物联网设备或远程服务器发送控制命令。大多数结构信息（*例如*报头、消息内容）和协议（*例如*，、HTTP、HNAP、MQTT）在API程序中被定义为消息有效载荷。因此，Snipuzz利用这些测试套件来与目标设备通信，同时提取消息序列作为初始种子。例如，当使用API程序打开灯泡时，程序首先向服务器或IoT设备发送登录信息，然后发送消息以定位特定灯泡设备，最后发送消息以控制设备打开灯。Snipuzz捕获触发IoT设备的功能的这样的消息序列作为初始种子。

恢复消息序列采集。为了重放崩溃分类的测试用例，SNIPUZZ确保被测设备在每轮测试中具有相同的初始状态。在向设备发送任何消息序列之后，SNIPUZZ将发送恢复消息序列以将设备重置到预定义状态。

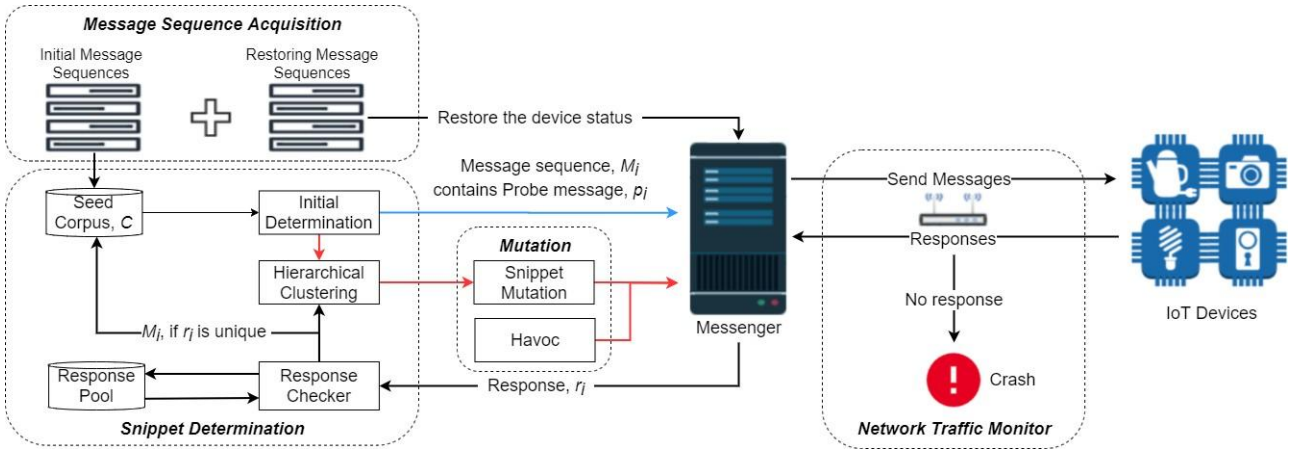

图2：Snip uzz的工作流程。利用有效的消息序列（种子），Snip uzz对每个单独的消息执行片段确定然后，Snip uzz对片段进行变异以生成新的消息序列。通过监视网络流量，Snip uzz在没有收到响应时确定

手动操作。尽管我们尽最大努力提供轻量级模糊器，但仍需要一些手动努力来获得有效且可用的初始种子。首先，我们手动配置测试套件中的程序，比如设置IP地址和登录信息。请注意，我们只需要为每个设备配置一次这些程序。其次，为了动态捕获消息序列，我们需要在网络流量监控器中手动定义特定的格式和协议。最后，我们过滤掉一些消息序列，将误导的模糊过程。例如，一些API程序提供可以自动更新或重启设备的操作。这些操作将停止设备，因此不会发回任何响应。这会导致误报崩溃，因为我们将无响应执行视为崩溃。手动工作花费每个设备大约5个工时，并且仅在Snipuzz的消息序列获取阶段期间需要。

4.2 片段测定

Snipuzz的核心思想是基于响应确定的片段来优化模糊过程。换句话说，Snipuzz利用片段变异来减少输入的搜索空间，而片段经由对来自IoT设备的响应进行分类而被自动聚类。主要的挑战是正确理解响应的语义。例如，由于时间戳的存在，如果利用简单的字符串比较，则两个语义相同的响应将被分类到不同的类别因此，Snipuzz利用启发式算法和分层聚类方法来确定每个消息中的片段

*4.2.1 初步确定。消息片段*的本质是消息中的连续字节，它使固件能够执行特定的代码段。对于有经验的专家来说，根据官方文档中的语义定义对消息片段进行分段并然而，对于缺乏这些知识的算法，应用一些自动方法来识别消息中每个字节的含义是必不可少的。

SnI puzz首先使用启发式算法将每条消息粗略地划分为初始片段。启发式算法的核心思想是 通过删除message（）中的某个字节 来生成探测消息 。通过对 每个探测消息的响应进行分类，SNIPUZZ初步确定消息中的片段 。

例如，如表2所示，为了确定消息中的片段，Sn_ puzz通过逐个地重新移动消息中的字节来生成*探测消息*当删除中的第一个字节*mp'{'时，对应的探测消息1为"on"：true}*。类似地，当删除第二个字节时，相应的探测消息
#21040;是"*真*"的。因此，具有11字节的消息可以生成-检查11条不同的探测消息（$_1$到$_{11}$）。SnI puzz将向设备发送包含探测消息的11个对应的消息序列（ $_1$至 $_{11}$）并收集响应。

然后 ，通过分类响应来区分消息中的片段。具体地，具有相同对应响应类型的连续字节被合并到相同片段中。根据表2所示的示例，响应
$_1$、 $_2$和 $_5$被合并到一个类别中，该类别指示JSON语法中的错误，而响应 $_3$和 $_4$被合并到另一个类别中，该类别指示无效输入参数的错误。因此，其对应响应属于相同类别的连续字节可以形成消息片段。通过这种启发式方法，SNIPUZZ可以确定消息中的所有初始片段 。

对响应进行分类的简单方法是利用字符串比较，*即*逐字节比较响应的内容然而，由于响应中存在随机性（*例如，*时间戳和令牌），简单的字符串比较可能不正确地将具有相同语义含义的响应区分为不同的类别。因此，引入了更高级的解决方案"编辑距离"[42]来确定响应类别。如等式（1）所示，计算两个响应 和 之间的相似性得分

表2：探测消息和对应响应消息的示例

留言内容		答复内容		分类
留言	关于我们	响应$_0$	{“success”：“/lights/1/state/on”：true}	0
探测消息$_1$	“on”：true}	响应$_1$	{“error”：{“type”：2，“address”：“/lights/1/state”，“description”：“body contains invalid json”}}	1
探测消息$_2$	{on“：true}	响应$_2$	{“error”：{“type”：2，“address”：“/lights/1/state”，“description”：“body contains invalid json”}}	1
探测消息$_3$	联系我们	响应$_3$	Copyright © 2018 - 2019深圳市创科光电科技有限公司All lights Reserved.粤ICP备16018888号-1	2
探测消息$_4$	联系我们	响应$_4$	Copyright © 2018 - 2019 www.lings.com.cn All lights Reserved.粤ICP备15018888号-1	3
探测消息$_5$	关于我们	响应$_5$	{“error”：{“type”：2，“address”：“/lights/1/state”，“description”：“body contains invalid json”}}	1
探测器消息$_{11}$	{“on”：true	响应$_{11}$	{“error”：{“type”：2，“address”：“/lights/1/state”，“description”：“body contains invalid json”}}	1

Message: {“on”:true}
Snippets: {“ o n “:true}
Response Category: 1 2 3 1

图3：片段确定的示例。

$$s_{kt}=1-\frac{\_\ (\ ,\ )}{\_\ (\ ,\ )}\quad (1)$$

其中，等式中的 _ 选择两个响应之间较长的字符串，_ 计算将一个字符串转换为另一个字符串所需的最小操作数，包括插入、删除和替换。因此，两个响应越相似，的 值越大。

SnI puzz首先为每个探针计算一个自相似性得分留言。请注意， 是通过更改消息中的第 -个字节生成的 。具体地，SnIPUZZ发送相同的探测消息
p在一秒的间隔内重复两次Two responses，$r'_i r$
将相应地从IoT设备收集然后根据两个响应计算自相似性分数

r_i，r_i'，根据等式（1）。请注意，由于中的随机性
两个响应之间可能存在差异
r_i、r_i'，即使它们来自相同的探测消息。因此，自相似性得分可以小于1。

为了确定两个响应是否属于同一类别，Snipuzz计算两个响应的相似性分数并将其与自相似性分数进行比较。例如，对于两个响应和，SnI puzz使用等式（1）来计算 相似性得分。 之后， 将其与自相似性进行比较。如果 >= 或 >= 满足，则响应 和 will be considered belonging to the same category;otherwise,responses and are then assigned to the different categories.

对于新接收到的响应 ，SNIPUZZ将基于相似性分数将其与对应响应池 中的所有响应进行比较。如果新响应 不属于任何现有类别，则响应 以及对应的探测消息 将被添加到*响应池中*。

利用响应池 ，Sn IPUZZ对消息中的每个字节进行分类 。具体来说，消息中第 个字节的类别
根据响应的类别进行分配 然后，具有相同类别的连续字节将合并为一个片段。图3示出了根据表2中的响应类别对消息 =*{“on”：true}*的初始片段确定的示例。

*4.2.2 分层聚类。*尽管SNIPUZZ利用相似性比较来减轻由响应中的随机性引起的错误分类，但是两个语义相同的响应仍然可能被错误分类到不同的类别中。当响应包含从探测消息提取或复制的内容时，可能会发生这种情况。例如，由于从探测消息引用特定的错误内容，启发式算法不会将它们分配到一个类别。具体地，=*{“on”：true}*的相似性得分$_{34}$

表2中的自相似性分数为0.979，小于表2中的自相似性分数。
$_{33}$= 1。000和 $_{44}$ = 1。000（因为在响应中没有随机性）。但是，这两个响应在语义上是相同的

并且应当被识别为一个类别，*即，*它们都是错误消息，指示参数语法错误位于探测消息中并且设备正在执行相同的代码块。

为了解决上述问题，SNIPUZZ使用凝聚层次聚类来细化消息片段。层次聚类的核心思想是不断合并两个最相似的聚类，直到只剩下一个聚类。

如算法1所示，Snipuzz将根据在第4.2.1节（第1行）中确定的初始片段来初始化片段之后，响应池中的每个响应类别将被初始化为一个集群（第2行）。Sn IPUZZ将响应转换成特征向量（第3行，在后面的段落中详细描述），其将用于计算每对聚类之间的距离（第5-7行）。然后，两个最接近的聚类将被合并，并且聚类中心将被相应地更新（第8-10行）。在执行聚类过程之后，SNIPUZZ将根据当前聚类结果生成新的片段，并将新的片段添加到片段分割结果中（第11行），片段分割结果将进一步用于变异。

具体地，SNIPUZZ首先从响应中提取特征，其将响应向量化为自相似性分数、响应的长度、字母段的数量、数字段的数量和符号段的数量的元组。每个段由具有相同类型的连续字节组成。例如，“123”是1个数字段，“1”中有2个字母段和1个数字段 。更具体地说，
表2中的$_1$将被矢量化为$_1$=1， 91， 10，（2，下 10 页 类似地，响应$_3$和$_4$将被转换为$_2$=1、 94、 11、 2、 13和14。 下一页
$_3$=1，（94，下 11页 2， 13.

图4显示了根据消息进行聚类的示例
=*{“on”：true}*。根据算法1，在聚类的准备轮（第0轮）中，响应池中的每个类别将被初始化为单个聚类。在第一轮中，由于聚类2和3是具有最小距离的两个聚类

算法1：片段的层次聚类**输入：**初始片段$_0$，响应
池结果：片段

```
1  F←
2←（ 0）;
3  V          ←
4当size（）> 1do
5  |  为←（）  至2做sizei
6  |  ror← （）-1to1dosize
7  |  |  distanceij,          vi    ),
   |  结束结束
8  |  |          ,
9  |  C     clusterCij←
10 |  V                  ij←
11 |  FF                 ←
   结束
```

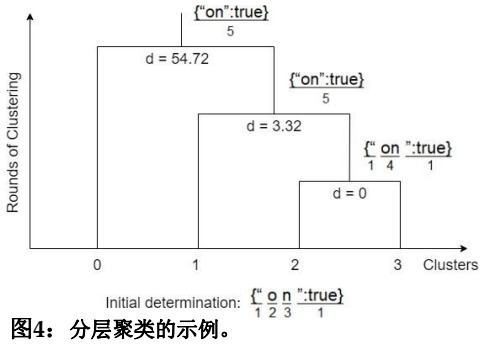

图4：分层聚类的示例。

（ 2_3=0），则两个集群被合并成一个新的集群。相应地，消息片段'O'和η被合并成用索引#4标记的新片段。类似地，在下一轮中，两个最接近的聚类（聚类1和新聚类）被合并，并且也将生成新的片段最后，消息中的所有片段被合并成一个新片段，*即*消息本身。所有新生成的代码片段以及初始代码片段将在下一阶段的消息变异中使用。

4.3 突变方案

片段突变。为了进行有效的模糊化，Snipuzz对在片段确定阶段中获得的片段进行突变。请注意，变异方案是对整个代码片段而不是消息中的单个字节执行的。

. **空的如果**未正确检查数据域，则数据域为空可能会使固件崩溃。因此，SNIPUZZ删除整个片段以清空数据域。

. **字节翻转。** 为了检测语法解析器和函数代码中的错误，Sn I p uzz翻转片段中的所有字节。这会改变字符串的语法意义，并在解析器没有正确检查语法时发现错误另一方面，字节翻转改变数据域的值以检查固件。**数据边界。** 为了检测在赋值期间发生的越界错误，Sn I p UZZ将数值数据的值修改为一些边界值

. （*例如，*，65536）。

. **字典。** 对于Dictionary的方案，SNIPUZZ用诸如"true"和"false"的预定义字符串替换片段，这可以直接探索更多的代码覆盖。

. **重复**为了检测语法解析器中的错误，Sn I p uzz多次重复片段。同时，数据域的重复可以检测到由边界外问题引起的缺陷。

大作战触发bug的条件可能很复杂。例如，它可能需要修改

同样的消息来触发一个bug前述片段

变异方案一次仅变异一个片段然而，在这方面，
大破坏突变随机选择a中的一些随机片段

消息，并且对每个所选择的片段执行上述突变方案Havoc突变不会停止，直到找到新的响应类别或目标物联网设备崩溃。

4.4 网络流量监视器

监视设备的网络通信，并设置超时以确定设备是否已崩溃。实际上，对设备网络通信的监控不是一个步骤，它发生在整个模糊过程中。在超时的情况下，SNIPUZZ将继续发送相同的消息序列三次，因为超时的原因可能是网络波动而不是设备崩溃。如果超时发生三次，则Sn IPUZZ将使用控制命令来物理地重启设备，并再次向设备发送相同的消息序列。如果设备仍然没有按时返回消息，则Sn IPUZZ将记录崩溃和相应的消息序列。

4.5 执行情况

Snipuzz的设计包括四个步骤：**消息序列获取**、**Snippet确定**、**变异**和**网络通信监控**。在**消息序列获取**步骤中，我们在程序中使用WireShark [45]来检测和记录API与IoT设备之间的通信数据包，并手动清理这些消息序列。其余的核心功能步骤被打包在一个原型中，该原型用4，000行C#代码实现。网络监视器将记录发送到设备的每一条消息，并在设备没有回复时再次向设备发送信息。智能插头用于实现目标设备的物理重启功能。当Sn IPUZZ需要物理地重新启动被测设备时，它会向智能插头发送控制消息，插头会关闭，然后打开。以这种方式，被测设备将短暂断电并重新启动。

5 实验评价

5.1 实验设置

环境设置。为了初始化IoT设备，我们使用制造商提供的应用程序来完成配对。为了更好地监控网络通信，所有被测设备都连接到本地路由器。我们的自动数据包提取器和SnI p uzz在配备IntelCore i7六核x 3.70 GHz CPU和16 GB RAM的Windows 10台式机上运行。PC也连接到路由器。

正在测试的IoT设备我们从全球线上和线下市场中挑选了20款流行的消费物联网设备，涵盖了飞利浦、小米、TP-Link、Netgear等知名品牌。所选物联网设备的类型包括智能插头、智能灯泡、路由器、家庭网桥、IP摄像头、指纹终端等。这些设备要么是亚马逊推荐的商品，要么是可以在超市买到的畅销产品。 表1详细说明了受测IoT设备的信息。

基准工具。 为了验证Sn IPUZZ在发现崩溃和消息分割方面的性能，我们使用了七种不同的模糊方案作为基准。

- **[9].** IotF U zze R的核心思想是通过对配套应用的静态分析，找到向物联网设备发送控制命令的函数，并对特定变量的值进行变异，从而在不破坏消息格式的情况下进行模糊测试。请注意，我们对IoTF U zze R的实现是最好的复制，因为他们的代码不是公开的，我们承认这可能会提供与原始版本略有不同的结果。
  我们通过将Sn I p uzz框架中的变异算法替换为IoTF uzze r中的变异策略来实现IoTF uzze R。考虑到IoTF u zze r中配套应用分析的目的是确保只有通信消息中的数据域发生变化，为了使基准测试尽可能公平，我们使用与Sn I p UZZ中使用的种子相同的种子

  并且在将每个种子消息馈送到IoTF用户之前手动地对每个种子消息的数据域进行分段。我们相信这种手动分割足以提供IoTF使用的上限性能。请注意，我们删除了与反馈机制和片段分割相关的方法，因为这些方法未在IoTF中使用。
- **Nemesys [22].** Nemesys是一款用于网络消息分析的协议逆向工程工具。它利用单个消息中值更改的分布来推断每个数据域的边界。考虑到Nemesys是一种协议推断方法而不是现成的模糊工具，我们基于Sn I p uzz框架实现了Nemesys的方法来推断片段边界，取代了相应的片段确定方法（第4.2节）。
- **BooF U zz [31].** 作为Sulley [19]的继承者，BooF uzz是一个出色的网络协议模糊器，已经参与了最近的几项模糊研究[9，37，48]。与其他自动模糊器不同，BooF UZZ需要人工引导的消息分割策略作为输入。在我们的研究中，我们利用这个属性，并手动定义更多的模糊策略，以丰富的基准评估。
  - **BooFuzz-Default**。在这个策略中，我们将输入中的每条消息都设置为一个完整的字符串，也就是说，BooFU zz会将消息作为字符串进行突变测试。
  - BooFuzz-Byte 输入中消息的每个字节将单独用于突变测试。
  - BooFuzz-Reversal 与IoTF U zze R的思想相反，在该策略中，我们专注于消息中非数据域的突变，同时保持数据域不变。
- **Doona[44].** Doon A 是 Bruterforce Exploit Detec-tor（BED）[6]的一个分支，其设计用于检测与网络协议中的缓冲区和格式相关的潜在漏洞。与其他工具不同的是，Doon A不以网络通信包为种子。DoonA的测试用例需要为每个受试器械或方案预先定义
- **剪你的手**。SNIPUZZ使用消息片段的分段来增强模糊的效率和发现崩溃的能力为了验证片段确定是否确实有益于模糊化，我们基于SnI puzz实现SnI puzz-NoSnI Snipuzz-NoSnippet不具有片段确定的组件，并且在不知道响应的情况下盲目地改变消息中的字节。

除了DoonA（其测试用例是预设的）之外，所有基准测试工具和SnI puzz都是在相同的输入集上测试的 这些输入集可以是不同的格式（*例如，*，BooFU zz要求手动设置输入，NU mesys要求输入为pcap文件格式），但内容相同。

还有许多其他流行的模糊工具能够通过网络通信测试物联网设备，例如Pe A c H[30]和AFLNET [33]。然而，由于它们是需要仪器固件的灰盒模糊，因此将这些工具视为黑盒方案的基线是不可行和不公平的。

5.2 漏洞识别

5.2.1 Sni PU zz. 在使用SnI puzz执行模糊测试之后20个物联网设备中的每一个24小时，我们在5个设备中检测到13个崩溃。 如表3所示，检测到的崩溃包括7个空指针取消引用，1个拒绝服务，以及我们进一步手动验证的5个未知崩溃。SnI p uzz发现的13个崩溃是通过提供格式错误的输入触发的这些格式错误的输入以不同的方式破坏消息格式例如，删除占位符、清空数据域或幸运地更改数据值的类型。

请注意，由SNIPUZZ识别的所有崩溃都在基于JSON的设备中，尽管我们成功地在具有各种通信格式（诸如JSON、SOAP和K-V对）的20个IoT设备上进行了实验。 实验还表明，与其他模糊器相比，Sn I p uzz观察到更高数量的响应类别（如第5.3节中详述的）。

空指针取消引用。 如表3所示，TP-Link HS 110和HS 100中由Sn I p uzz触发的7次崩溃均由空指针解引用引起。将测试用例发送到HS110和HS100后，设备崩溃，无法回复任何交互。然而，几分钟后，设备自动重启并恢复到初始状态。通过对测试用例的分析，我们发现这些漏洞都是由JSON语法突变的消息触发的换句话说，当一些重要的占位符（如花括号和冒号）或测试消息的一部分发生变化时，消息的语法结构和语义就被破坏了。如果设备不能正确处理变异的输入消息，它将使设备崩溃。我们于2020年6月13日通过电子邮件向设备供应商TP-Link报告了漏洞。 他们已经确认了这个漏洞，并承诺通过固件更新来修复它。

表3：实验结果。Snip uzz发现的类别数量最多，暴露的bug数量也最多

#	装置	SnipU		IoTFU		Doona		BooF U zz-Defa ULT		BooF U zz-By T e		BooF U zz-Reversa L				Nemesys	Snip U
		T C	10/24	T C	10/24	C类	10/24	C类	10/24	C类	10/24	C类	10/24	C类	10/24	C类	10/24
1	YLDP05YL	UC 3*	**46/71**	UC 1*	31/33	不适用	无/无	0	11/17	0	11/41	0	11/22	0	26/61	0	21/69
		UC 2*	**35/76**	UC 1*	20/24											0	22/70
2	YLDP13YL	DoS 1	**28/41**	/零	18/22	不适用	无/无	0	8/18	0	8/42	0	8/22	0	18/62	0	20/39
		/零	**46/72**	/零	18/31											0	18/70
3	公司简介	/零	**28/51**	/零	8/19	0	5/16	0	7/13	0	8/33	0	5/21	0	22/36	0	13/48
		/零	**65/110**	/零	29/36											0	22/99
4	迷你C	/零	**34/51**	/零	29/33	0	7/15	0	5/11	0	6/31	0	5/21	0	18/68	0	23/50
5	公司简介	NPD 3	**全天**24	/零	20/27	不适用	无/无	0	4/11	0	4/31	0	4/20	0	13/40	0	十九/**七十一**
		NPD 4	小时	/零	17/22											0	19/78
		/零	**24/79**	/零	7/10											0	12/21
6	顺化	/零	**十三/**	/零	15/17	0	4/11	0	7/11	0	9/31	0	7/25	0	34/110	0	20/61
7	基站	/零	**二十一**	/零	8/13	0	7/16	0	6/9	0	9/17	0	7/13	0	19/38	0	9/41
8	公司简介	/零	**42/61**	/零	8/41	不适用	无/无	0	6/13	0	6/31	0	6/22	0	20/64	0	8/49
		/零	**十九/**	/零	18/32											0	17/40
9	公司简介	/零	**四十二**	/零	**20**/24	不适用	无/无	0	6/14	0	9/33	0	6/22	0	20/62	0	6/**41**
		/零	**25/61**	/零	38/44											0	29/76
10	F7C027au	/零	**37/43**	/零	16/22	0	6/14	0	8/12	0	6/18	0	6/15	0	8/14	0	十三/**四十一**
11	MSS310	/零	11/37	/零	36/33	0	8/16	0	5/11	0	8/45	0	8/21	0	30/59	0	21/**76**
12	B25AUS	/零	**第**	/零	9/22	0	7/19	0	7/14	0	11/17	0	7/11	0	16/36	0	20/**91**
13	迷你美国	/零	**53/81**	/零	9/30	不适用	无/无	0	7/16	0	7/35	0	7/22	0	9/55	0	5/35
			号决议														
			25/36														
14	SP4L-AU		**39**/75			0	5/11	0	5/17	0	7/32	0	5/23	0	23/40		
15	R6400		**三十六/**			0	4/13	0	3/12	0	4/24	0	4/18	0	6/30		
16	公司简介		八十			不适用	无/无	0	8/16	0	8/46	0	8/27	0	41/70		
			14/36														
17	Alro Pro 2					0	10/14	0	8/13	0	14/22	0	10/17	0	18/22		
18	F19821W					0	7/13	0	5/11	0	7/23	0	7/14	0	27/65		
19	T-131P					0	7/16	0	7/20	0	9/42	0	7/35	0	21/65		
20	RM mini 3					不适用	无/无	0	10/17	0	14/31	0	10/23	0	6/30		

UC：未知崩溃。空指针解引用。**DoS**：拒绝服务。**T**：漏洞类型。**C**：崩溃次数**10/24**：响应类别的数量（10分钟/24小时）。
*：可远程**NA**：由于DoonA仅适用于某些网络协议，因此无法测试的设备由"NA"表示

表4：Snip uzz IoTF uzz的突变信息

变异消息的内容	
{"{"id"：0，"method"："start_cf"，"params"：["4，4，"1000， 2，2700，100，500，1，255，10，5000，7，0，0，500，2，5000， 1"]}"	原始消息
{"{"id"：0，"method"："start_cf"，"params"：["4，"，1000， 2，2700，100，500，1，255，10，5000，7，0， 0，500，2，5000，1"]}"	SnI pU
{"{"id"：0，"method"："start_cf"，"params"： ["，4，"1000， 2，270000，100，500，1，255，10，5000，7，0， 0，500，2，5000，1"]}"	zzIoTFU zzeR

拒绝服务。另一个有趣的发现是在飞利浦A60智能灯泡中检测到的拒绝服务漏洞。 在经过Sn I p uzz 24小时的测试后，飞利浦的官方配套应用无法正常管理设备。具体地，在应用中找不到设备，并且如果通过应用发送任何进一步的消息，则应用中的响应将继续要求将设备绑定到设备组，并且没有进一步的交互可用。 但我们观察到，如果消息包直接发送到设备，设备可以正常工作。这表示设备没有完全崩溃，但其通过配套应用程序的服务被拒绝。

未知的崩溃。 Sn IPUZZ在Yeelight灯泡、YLDP05YL和YLDP13YL上发现了5次崩溃。这些设备在大约一分钟内崩溃并自行重启 通过分析测试用例，我们发现崩溃是由于删除了某些数据域，例如表4中标记为红色的参数无效。由于2台设备的固件未公开，无法确定漏洞的根本原因;但我们仍然可以推断，该漏洞是由于设备在解析过程中读入空值，导致分配过程中崩溃。 我们还发现，使用本地网络的通信不需要任何身份验证，这意味着设备可以被本地网络中的任何攻击者崩溃。因此，我们认为这些漏洞是

5.2.2 使用最先进的工具进行基准测试 如表3所示，对于每个所有设备，这也限制了其容量。由于Boo fuzz直接用预设的字符串替换消息中的指定位置，因此只能触发有限类型的漏洞。Nemesys提供了一个确定消息片段的新想法。但是，由于它通过消息中值的分布来确定消息片段，因此

Nemesys很难准确地确定数据域和非数据域之间的边界。因此，Nemesys很难检测到只有通过改变数据域或非数据域才能触发的漏洞。SnI puzz-NoSnI pet，它不应用

设备上的24小时模糊测试，除了IoTF之外，没有一个基准测试工具发现崩溃。由于各种原因，他们没有发现坠机事件。DonnA更多地关注通信协议的突变此外，DonnA不能应用于

Snipuzz中使用的基于片段的突变方法类似于经典的模糊器AFL[24]。由于SnI puzz-NoSnI pet不推断消息的结构，而是直接使用单个或多个连续字节作为突变单元，因此由SnI puzz-NoSnI pet生成的大多数测试用例破坏了消息的结构。这种方法难以在需要高度结构化输入的设备上工作

IoTF U zze R在2个智能灯泡设备中检测到2个崩溃，*即*、YLDP05Y和YLDP013Y。由于IOT-Fuzze R的突变策略，通过清空数据域来获得由IoTF uzze R提供的畸形输入。根据表4中列出的突变消息，我们可以看到，由IOT-Fuzze突变的消息类似于由SNIPUZZ生成的消息。表4中的来自Snipuzz和IoTFUZZ的消息的突变域都在数据域中就突变测试的效果而言，SnI pUZZ和IoTFU zzeR在这两个消息上实现了相同的目标然而，Sn_PUZZ可以覆盖IoTF_uzz_R的突变空间，因为IoTF_uzz_R仅关注数据域突变，而Sn_PUZZ可以使数据域和非数据域两者突变。

为了进一步确定崩溃的根本原因，我们获得了TP-Link生产的两种典型市场消费级智能插头HS 100和HS 110的固件源代码，并进行了案例研究，反映了Sn I p uzz和IoTF uzze R之间的差异。我们发现，两个设备上由Sn I p uzz触发的崩溃之一是由于破坏语法结构并在数据域和非数据域上发生突变而引起的。更具体地，突变的消息成功地绕过了消毒器并在函数执行期间触发了崩溃。我们推断

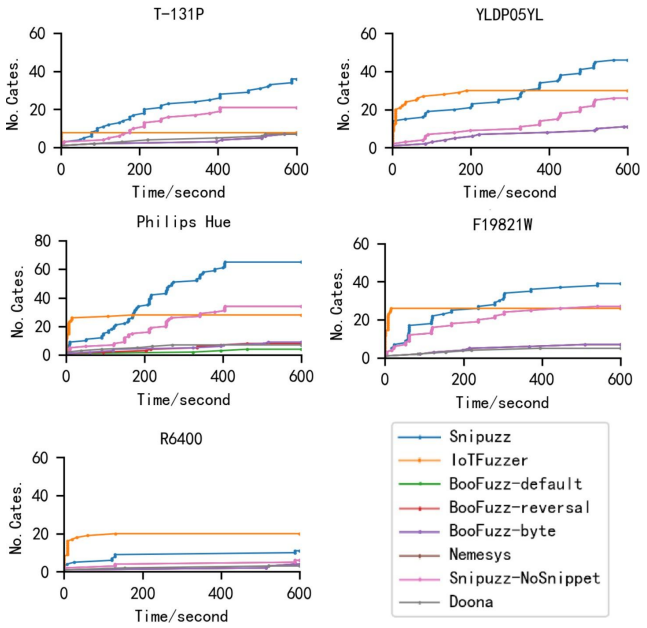

图5：随着时间的推移发现的类别数量。

这可能是由容易出错的第三方消毒剂引起的（更多详细信息可参见附录B）。另一方面，由于IoTF U zze R的设计，模糊化基于语法规则，因为IoTF U zze R倾向于以第一优先级满足语法要求，以便不被sanitizer拒绝并确保每个测试用例可以到达固件中的功能执行部分。与Snipuzz相比，这种策略限制了模糊化的测试范围及其覆盖消毒部分的能力。因此，我们认为，考虑到物联网固件测试的复杂性，迫切需要一种轻量级且有效的黑盒漏洞检测工具，例如Sn I p uzz。

5.3 运行时性能

图5示出了Snipuzz和其他七个模糊器如何在前**10分钟**期间探索设备固件。由于篇幅所限，我们在此仅展示了5件器械的结果，但在附录A中绘制了所有20件器械的结果。我们重复了10次模糊测试，并记录了每种方法发现的响应类别数的中间值，表明覆盖率已经探索。我们手动审查所呈现的响应类别，以消除由响应中的随机性或设备的响应机制引起的错误分类。

如图5所示，DOONA只能检测到少量的响应类别。Doon A是基于协议的模糊方法，其测试更偏向于协议内容。通信协议上的突变测试被设备单方面直接拒绝或忽略的概率很高，导致可以接收到的响应类别很少。

我们实现了三个模糊化策略的基础上Boo模糊zz，*即*。将整个消息作为字符串进行变异，将消息的每个字节进行变异，以及将非数据域进行变异。然而，测试结果表明，他们都探索了非常有限的类别

每个设备上的响应类别发现的局限性是由于Boo fuzz的变异策略，它用特定的预定义字符串替换目标内容。例如，使用字符串，诸如"/././././././."，以替换不同策略中的消息内容（*例如，*整个字符串、单个字节或非数据域的替换）导致违反消息格式，并且可以容易地被净化器拒绝。因此，Boo fuzz获得的大多数响应都属于"错误响应"的范畴

IotFU zzeR探索的响应类别的数量在短时间内迅速增长，然后停滞。在变异阶段，IOTFUZZE从原始候选输入中随机选择一组输入，并且随机变异一个或多个消息的数据域。它将继续重复此方法，直到设备崩溃或达到时间限制。这种基于随机性的方法有助于IotF uzze R在原始输入中对大量消息数据域进行变异和测试，并在开始时快速收集响应消息类别。然而，由于数据域突变，由IotF U zze R发现的响应类别的数量将很快达到限制。

在大多数设备中，Sn IPUZZ在大多数情况下保持稳定的上升趋势，并且在一段时间之后，Sn IPUZZ找到的响应类别的数量超过IotF U zze R。与IotFU zzeR不同，SnI pU zz主要通过**Snippet Determination**阶段搜索响应类别 根据消息片段探索策略，Sn IPUZZ首先尽可能多地探索某个消息的所有响应类别。在通过**Snippet Mutation**获得并测试了消息的片段之后，将以相同的方式处理下一条消息，直到初始消息序列中的所有消息都已测试完毕。遵循这种方法，SnI p uzz可能不会在短时间内得到大量的响应类别。当Sn I p uzz检测到消息片段时，消息内容中的每个字节都将包括在测试中。因此，如表3中的粗体数字所示，对于20个设备中的15个，与其他最先进的IoT模糊工具相比，SNIPUZZ在24小时模糊测试之后覆盖了最多数量的响应类别。

在5个设备上，SnI puzz-NoSnI pet在24小时内比SnI puzz收集更多的响应类别 Snipuzz-NoSnippet使用的突变方法类似于经典模糊器AFL [24]。它直接对单个字节或几个连续的字节进行变异。然而，SNIPUZZ-NOSNIPPET难以覆盖不是通过打破语法格式获得的响应类别（*例如，Snipuzz-NoSnippet*）。数据域中的数据出界理论上，尽管SNIPUZZ-NOSNIPPET突变方法不是那么有效，但它仍然具有探索大多数类别的响应的能力。

Nemesys比BooFU zz和DoonA探索了更多类别的响应，但没有超过SnI pU zz。Nemesys策略依次在消息的每个数据域上执行确定性突变，这使得其运行时性能的趋势与SnIPUZZ相似。然而，Nemesys的数据域确定策略不是基于来自IoT设备的响应。因此，消息中字节值的分布在覆盖更多响应类别方面没有益处。因此，Nemesys收集的响应类别数量有限。

有趣的是，观察到在R6400的情况下，在仅找到几个响应类别之后，Snipuzz也进入停滞我们的

表5：Snip uzz和Nemesys的推断结果

方法学	平均相似性	范例
SinI puzz	87.1%	{"on"：true，"sta"：140，"bri"：254}
内梅西斯	百分之六十四点五	{"on"：true，"sta"：140，"bri"254}
文法	百分之一百	{"on"：true，"sta"：140，"bri"：254}

仔细检查初始输入消息序列，发现消息的平均长度超过400字节，迫使Sn IPUZZ生成并发送大量探测消息以确定消息片段。因此，在前10分钟内，Sn I p uzz还在探索前几条消息的响应类别，因此没有超过IotF U zze r

5.4 在所有策略中，SNIPUZZ和Nemesys利用语义分割来评估其消息片段推理的性能。

我们比较了他们在模糊处理过程中产生的片段与API文档中定义的语法规则具体来说，对于一些成熟流行的语言，如JSON，我们按照其标准语法建立语法规则;对于自定义格式，如字符串或自定义字节，我们参考官方API文档，根据说明定义语法规则。

等式（2）量化了片段推断的质量，并且*相似性*指示片段确定的消息中正确分类的字节的百分比，，与基础相比truth，，从语法规则中手动提取

$$Similarity = 1 - \frac{count[\quad cate(\ ) \oplus (\ )]}{(\ )} \qquad (2)$$

其中以一系列“0”和“1”位返回每个消息字节的类别$catelencount$注意，在地面实况消息中，“0”指示非数据域（在表5中标记为蓝色），而“T指示数据域（在表5中标记为红色）。因此，是按位运算。XOR

此外，在等式（2）之后，我们计算由SNIPUZZ和Nemesys针对从实验获得的所有235个消息确定的片段（或数据域）的平均相似性需要注意的是，在计算平均相似度的过程中，对于每条消息，如果确定了多个片段集合，我们将选择相似度值最高的片段推断，因此一个片段可以尽可能多地反映语法规则，最大化消息语义分割的性能。Sn I p UZZ的平均相似度结果为87.1%，表明通过应用基于层次聚类方法的片段推理，Sn I p uzz可以有效地发现隐藏在消息中的语法规则。理想地，在SNIPUZZ中，集群的合并消除了由响应中的随机性和由回复消息机制本身引起的影响因此，消息片段将逐渐符合语法规则，这导致Sn_I_puzz得到更高的相似性结果。

但是，我们也发现片段推理方法和语法规则在一些结果上存在一些差异例如，给定表5中所示的示例，片段推断方法将属于数据域的字符串组合在语法规则（*即*、'true'、'140'和'254'），并带有一些占位符（如双引号和大括号）。在分析响应消息后，我们发现销毁这些数据域和销毁占位符后获得的响应都是关于无效格式的。这可能是由于以下事实：在固件中，当在解析格式中发生错误时，响应不报告错误的详细描述，而是返回一般格式错误。

另一方面，Nemesys利用协议中值变化的分布来确定不同数据域的边界，并实现消息的语义分割。这种方法的优点是，它不需要任何其他额外的信息，如语法规则或大量的训练数据集，除了消息本身。

Nemesys的平均相似性结果为64.5%，低于SnI puzz结果。给定表5中所示的示例，当以需要受限语法的格式分割消息时，例如Json和XML，Nemesys可以实现良好的语义分割性能，因为占位符通常使用数据域中不常用的符号。字节值的这种分布使Nemesys能够有效地找到数据域之间的边界然而，在IoT设备中，定制格式是普遍的。例如，智能灯泡BR30使用自定义字节作为通信手段，其中每个字节对应于特定含义（*即，每个字节对应于一个或多个字节*）。，*“0x61”*表示*“CHANGE_MODE”*并且*“0x0f”*表示*“TRUE”*）。在这种情况下，字符的值分布不能再用作数据域确定的指导，因此Nemesys确定的消息分段容易出错。

6 讨论和限制

SnI p uzz已经成功检查了20种不同的设备，并暴露了其中5种设备的安全漏洞然而，仍然存在与SNIPUZZ的效率和可扩展性相关的一些限制。我们在本节中讨论的局限性，并提出解决方案，作为未来的工作。

可扩展性和手动工作。如果有效网络分组是已知的，则 IoT设备可以由SNIPUZZ测试。在我们的原型中，我们通过运行API程序和监视网络通信来捕获通信数据包（注意，数据包也可以通过静态分析API程序而不运行它们来获得在没有API程序或文档的情况下，我们可以通过反编译和污点分析从物联网设备的官方应用程序中恢复消息格式。或者作为第二种方式，我们可以通过拦截APP和物联网设备之间的通信来解决这个问题，然后从捕获的数据包中恢复消息格式。第二种方式是可行的，我们已经在TP-Link的物联网控制APP KASA中进行了实验，可以进一步开发更多的物联网设备。然而，这两种方法都可能引入开销并且涉及手动工作。

回想在第4.1节中，Snipuzz需要手动工作，这在消息序列获取阶段期间每个设备花费5个工时来收集初始种子 手动工作主要涉及从API程序中清除分组，所述API程序从公开可用的第一和第三方资源获得。为了在将Sn IPUZZ应用于IoT设备时减轻这一限制，可以使用爬虫等技术来自动收集API

在未来的工作中与物联网设备相关联的程序此外，还可以通过脚本对关键字进行预处理来改进数据包清理过程，实现通信包的自动收集。

有效性的威胁 由于SNIPUZZ通过API程序和网络嗅探器收集初始消息序列，因此第一个威胁来自API程序的缺失。在这种情况下，我们可以根据IoT设备的配套应用程序（类似于IoT-FuzzeR）恢复消息格式，但可能需要更多的手动操作。第二，消息中的加密降低了片段确定的有效性，因为语义信息可能被破坏。加密问题的潜在解决方案是将解密模块集成到SNIPUZZ中。最后，固件的代码覆盖率可能受制于API程序的可访问性，因为SNIPUZZ只能检查API程序中覆盖的功能。重新组合来自不同种子的消息片段以生成新的有效输入可以减轻这种限制。

加密。 在消息采集过程中，我们注意到在一些API程序中，加密被用来保护通信。加密对消息序列变异过程没有影响，但片段确定过程基本失败。由于加密算法破坏了消息的原始格式，因此片段的分段对字符的位置很敏感此外，因为来自设备的响应消息也被加密，所以Snipuzz不能从它们获得有用的反馈类似地，可以将API程序中的加密和解密算法集成到Sn i puzz模块中来解决这一限制，或者可以从变异策略设计的角度

覆盖范围。 Sn I p uzz探索的固件的代码覆盖率取决于API程序。例如，如果灯泡的API程序仅支持打开电源的功能，则几乎不可能通过改变在打开电源期间捕获的消息来探索调节亮度的功能。在未来的工作中，在没有语法支持的情况下，我们将考虑重新组合消息片段以尝试生成新的有效输入。此方法可以帮助探索除所提供的原始输入之外的更多固件执行覆盖。

详细答复的要求 Snipuzz的检测有效性取决于消息片段的质量，消息片段的质量取决于可以从IoT设备的响应获得多少信息。换句话说，如果IoT设备不提供足够详细的响应，例如用统一的消息报告所有错误，则Sn IPUZZ可能难以确定消息片段。幸运的是，在许多IoT设备中，可以在调试模式中获得高级错误描述，这将显著改进Snipuzz中的消息片段的确定过程。

7 相关工作

我们的Snipuzz以黑盒方式执行，用于检测IoT设备中的漏洞与现有的用于IoT设备的黑盒模糊（black-box fuzzing）盲目地改变消息不同，Snipuzz通过利用响应来优化黑盒模糊的改变过程这反馈机制提高了缺陷发现的有效性例如，IoTFU zzeR [9]获得数据域，IoTFU zzeR在该数据域上执行盲突变。因此，IoTF用户缺乏所生成的输入的质量的知识，导致在低质量输入上的资源浪费。还有几种动态分析方法专注于物联网设备的网络模块。例如，SPFuzz定义了一种新的语言，用于描述协议规范、协议状态转换及其相关性[37]。SPFuzz可以保证会话状态下消息格式的正确性和协议的依赖性IoTHunter是一种模糊物联网固件状态协议的灰盒方法[47]。IoTHunter可以不断切换协议状态，以执行基于反馈的物联网设备探索。在最近的一个例子中，AFLnet充当客户端，并连续重放发送到目标的原始消息序列的变化（*即*服务器或设备）[33]。AFLnet使用响应代码（表示执行状态的数字）来识别目标的执行状态，并探索其网络模块的更多区域

物联网设备动态分析的另一个研究方向是使用仿真器。仿真的缺点是繁重的工程工作和固件的要求，尽管物联网固件的仿真可以比黑盒模糊更彻底地分析。物联网固件仿真的两个主要挑战是可扩展性和吞吐量。因此，提高仿真性能的努力包括全系统仿真[8，27]，提高仿真成功率[21]，硬件独立仿真[17，38]，以及用户和系统模式仿真的组合[51]。基于仿真，模糊可以集成到这些框架中，并且可以在固件中寻找缺陷[38，51]。

固件的静态分析是动态分析的补充。语义相似性是静态分析成功的主要技术之一研究人员通过比较文件和模块[13]、控制流图（CFG）[14]、解析器和复杂处理逻辑[11]以及多二进制交互[35]来分析语义相似性。还有许多基于相似性的方法可以跨不同的固件架构检测漏洞。它们通常从CFG中的每个节点的固件中提取各种与架构无关的特征来表示函数，然后检查两个函数的CFG表示是否相似[15，32]。

8 结束语

在本文中，我们提出了一个黑盒模糊框架Sn i puzz设计用于检测隐藏在物联网设备中的漏洞。与其他黑盒网络模糊测试不同，Sn I p uzz利用设备返回的响应消息建立反馈机制，指导模糊变异过程。此外，Sn I p uzz基于来自设备的响应来推断消息中的每个字节的语法角色，使得Sn I p UZZ可以在没有语法规则的指导下生成满足设备的语法的测试用例。 我们已经使用了市场上的20个消费级物联网设备来测试Sn I p UZZ，它已经成功地在5个不同的设备上发现了5个零日漏洞。

参考文献

[1] 2020.物联网架构所需的三个软件栈*IoT Eclipse（白皮书）*（2020）。

[2] Cornelius Aschermann， Tommaso Frassetto， Thorsten Holz， Patrick Jauernig， Ahmad-Reza Sadeghi， and Daniel Teuchert.2019 年 。NAUTILUS：Fishing for deep bugs with grammars.. *网络和分布式系统安全研讨会（NDSS）*

[3] 一、Ashraf，X.马，B。Jiang和W.K的。陈2020年。GasFuzzer：模糊以太坊智能合约二进制文件，以暴露面向气体的异常安全漏洞。*IEEE Access*（2020）。

[4] Marcel Böhme 、 Van-Thuan Pham 、 Manh-Dung Nguyen 和 Abhik Roychoudhury。2017. 直接灰盒模糊。2017*年ACM SIGSAC计算机和通信安全会议论文集*。

[5] Marcel Böhme、Van-Thuan Pham和Abhik Roychoudhury。2016年。基于覆盖的灰盒模糊马尔可夫链。2016*年ACM SIGSAC计算机和通信安全会议论文集*。

[6] 卡莉机器人 2019. 床。 https://gitlab.com/kalilinux/packages/bed。

[7] Z的。Berkay Celik Patrick McDaniel和Gang Tan。2018年。Soteria：自动化物联网安全和安全分析。2018年*USENIX年度技术会议（USENIX ATC 18）*。

[8] Daming D Chen，Maverick Woo，David Brumley，and Manuel Egele.2016年。面向Linux嵌入式固件的自动化动态分析 *网络和分布式系统安全研讨会（NDSS）*

[9] Jiongyi Chen，Wenrui Diao，Qingchuan Zhao，Chaoshun Zuo，Zhiqiang Lin，Xiaofeng Wang，Wing Cheong Lau，Menghan Sun，Ronghai Yang，and Kehuan Zhang.2018. IOTFUZZER：通过基于应用程序的模糊检测发现物联网中的内存损坏。*网络和分布式系统安全研讨会（NDSS）*

[10] AbrahamClements、Eric Gustafson、Tobias Scharnowski、Paul Grosen、DavidFritz、Christopher Kruegel、Giovanni Vigna、Saurabh Bagchi和Mathias Payer。2020. HALucinator：通过抽象层仿真重新托管固件*第29届USENIX安全研讨会（USENIX '20）*。

[11] Lucian Cojocar、Jonas Zaddach、Roel Verdult、Herbert Bos、Aurélien Francillon和Davide Balzarotti。2015年。PIE：嵌入式系统中的解析器识别

[12] Jake Corina、Aravind Machiry、Christopher Salls、Yan Shoshitaishvili、Shuang Hao、Christopher Kruegel和Giovanni Vigna。2017年。Difuze：内核驱动程序的接口感知模糊2017*年ACM SIGSAC计算机和通信安全会议论文集*。

[13] Andrei Costin， Jonas Zaddach， Aurélien Francillon， and Davide Balzarotti.2014年嵌入式固件安全性的大规模分析第*23届USENIX安全研讨会（USENIX Security 14）*。

[14] Thomas Dullien和Rolf Rolles。2005年基于图形的可执行对象比较（英文版）。*Journal of Computer Virology and Hacking Techniques*（2005）.

[15] Sebastian Eschweiler、Khaled Yakdan和Elmar Gerhards-Padilla。2016年。discovRE：高效的跨体系结构识别二进制代码中的错误。*网络和分布式系统安全（NDSS）*

[16] Pwnie Express.2020年。是*什么让物联网如此容易受到攻击？*技术报告。前哨24

[17] Bo Feng，Alejandro Mera，and Long Lu.2020年。P2IM：通过自动外围接口建模进行可扩展和独立于硬件的固件测试。 第*29届USENIX安全研讨会（USENIX Security 20）*

[18] Qian Feng，Rundong Zhou，Chengcheng Xu，Yao Cheng，Brian Testa，and Heng Yin.2016年。可扩展的基于图形的固件映像错误搜索*2016年ACM SIGSAC计算机和通信安全会议论文集*。

[19] Fitblip 2019. 小苏 https://github.com/OpenRCE/sulley的网站。

[20] EricGustafson， Marius Muench，Chad Spensky，Nilo Redini，Aravind Machiry，Yanick Fratantonio，Davide Balzarotti，Aurélien Francillon，Yung Ryn Choe，Christophe Kruegel，et al. 2019年。通过自动化重新托管实现嵌入式固件的分析。第*22届攻击、入侵和防御研究国际研讨会（RAID 2019）*

[21] MingeunKim、Dongkwan Kim、Eunsoo Kim、Suryeon Kim、Yeongjin Jang和Yongdae Kim。2020. FirmAE：面向物联网固件的大规模仿真，以进行动态分析。*年度计算机安全应用会议*。

[22] Stephan Kleber，Henning Kopp，and Frank Kargl. 2018. NEMESYS：通过分析单个消息的内在结构进行网络消息语法逆向工程第*12届USENIX进攻性技术研讨会（WOOT18）*

[23] 卡拉·兰特2017年。到2020年，地球上每个人将拥有4台设备*未来主义*（2017）。

[24] lcamtuf. 2017. 劳联 https://lcamtuf.coredump.cx/afl/。

[25] 趋势科技2020. *Mirai僵尸网络通过CVE-2020-5902攻击物联网设备*。技术报告。安全情报博客。

[26] 趋势科技2020年。*智能但有缺陷：物联网设备漏洞*解释 技术报告。BBC News.

[27] Marius Muench，Jan Stijohann，Frank Kargl，Aurélien Francillon，and Davide Balzarotti.2018年。你破坏的不是你崩溃的：模糊化的挑战嵌入式设备在*NDSS 2018中，网络和分布式系统安全系统*。

[28] 林赛·2020年。*超过一半的物联网设备容易受到严重攻击*。技术报告。威胁邮报

[29] Sebastian Österlund、Kaveh Razavi、Herbert Bos和Cristiano Giuffrida。2020年。ParmeSan：Sanitizer引导的灰盒模糊。 第*29届USENIX安全研讨会（USENIX Security 20）*

[30] Peachtech 2021. PEACH：PEACH模糊器平台。https://www.peach.tech/products/peach-fuzzer/访问日期：2021-01.

[31] 约书亚·佩雷达2017. boofuzz：用于人类的网络协议模糊。https://boofuzz.readthedocs.io/en/stable/的网站。

[32] JannikPewny， Behrad Garmany， Robert Gawlik， Christian Rossow， and ThorstenHolz. 2015年。二进制可执行文件中的跨体系结构错误搜索2015年*IEEE安全与隐私研讨会（SP）*。

[33] Van-Thuan Pham， Marcel Böhme， and Abhik Roychoudhury.2020年。AFLNET：一个网络协议的灰盒模糊器。IEEE*International Conference on Software Testing，Verification and Validation（ICST）2020*。

[34] Van-Thuan Pham， Marcel Böhme， Andrew Edward Santosa， Alexandru Razvan Caciulescu， and Abhik Roychoudhury. 2019. 智能灰盒模糊。*IEEE Trans-actions on Software Engineering*（2019）。

[35] Nilo Redini、Aravind Machiry、Ruoyu Wang、Chad Spensky、Andrea Continella、Yan Shoshitaishvili、Christopher Kruegel和Giovanni Vigna。2020年。Karonte：检测嵌入式固件中不安全的多二进制交互。2020年*IEEE安全与隐私研讨会（SP）*

[36] Sergej Schumilo，Cornelius Aschermann，Robert Gawlik，Sebastian Schinzel，and Thorsten Holz. 2017. kAFL：用于操作系统内核的硬件辅助反馈模糊。第*26届USENIX安全研讨会（USENIX Security 17）*。

[37] 宋聪西、于波、周旭、杨强。2019年。SPFuzz：一个用于有状态网络协议模糊的分层调度框架。*IEEE Access*（2019）.

[38] PrashastSrivastava， Hui Peng， Jiahao Li， Hamed Okhravi， Howard Shrobe，andMathias Payer. 2019年。FirmFuzz：自动物联网固件自检和分析。*第二届国际ACM物联网安全与隐私研讨会论文集*。

[39] 连姆·董2017年。*物联网设备的数量今年将首次超过世界*技术报告。ZDNet.

[40] Junjie Wang，Bihuan Chen，Lei Wei，and Yang Liu.2017年。Skyfire：数据驱动的模糊种子生成。*2017*年IEEE安全与隐私研讨会（SP）。

[41] YanhaoWang，Xiangkun Jia，Yuwei Liu，Kyle Zeng，Tiffany Bao，Dinghao Wu，andPurui Su. 2020年。不是所有的覆盖率度量都是相等的：模糊覆盖率占输入优先级。*网络和分布式系统安全研讨会（NDSS）*

[42] 维基百科。 2021. 编辑距离。https://en.wikipedia.org/wiki/Edit_distance。

[43] 维基百科。 2021. 层次聚类。 https://en.wikipedia.org/wiki/_clustering.

[44] wireghoul。 2019. 杜娜 https://github.com/wireghoul/doona。

[45] 有线通信。 2020. 关于wireshark https://www.wireshark.org/about.html。

[46] Xiaojun Xu，Chang Liu，Qian Feng，Heng Yin，Le Song，and Dawn Song.2017年。基于神经网络的图嵌入跨平台二进制代码相似性检测。*2017年ACM SIGSAC计算机和通信安全会议论文集*。

[47] Bo Yu，Pengfei Wang，Tai Yue，and Yong Tang.2019年。海报：通过多阶段消息生成模糊物联网固件。*2019年ACM SIGSAC计算机和通信安全会议论文集*。

[48] Y的。Yu，Z.Chen，S.Gan和X.小王。2020年。SGPFuzzer：一个用于网络协议实现的状态驱动智能灰盒协议模糊器。*IEEE Access*（2020）。

[49] Tai Yue，Pengfei Wang，Yong Tang，Enze Wang，Bo Yu，Kai Lu，and Xu Zhou.2020年。EcoFuzz：自适应节能灰盒模糊，作为对抗性多臂强盗的变体。第*29届USENIX安全研讨会（USENIX Security 20）*

[50] Jonas Zaddach，Luca Bruno，Aurelien Francillon，Davide Balzarotti，et al.2014年AVATAR：一个支持嵌入式系统固件动态安全分析的框架。*网络和分布式系统安全研讨会（NDSS）*

[51] YaowenZheng， Ali Davanian， Heng Yin， Chengyu Song， Hongsong Zhu，andLimin Sun. 2019. FIRM-AFL：通过增强的进程仿真对IoT固件进行高吞吐量灰盒模糊测试。第*28届USENIX安全研讨会（USENIXSecurity 19）*。

附录

A 运行时性能

图6示出了在前**10分钟**期间Sn_I_puzz和其他七个基线的运行时性能。在大多数基准测试中，Sn I p uzz发现了最多数量的类别。既然SNIpuzz花了

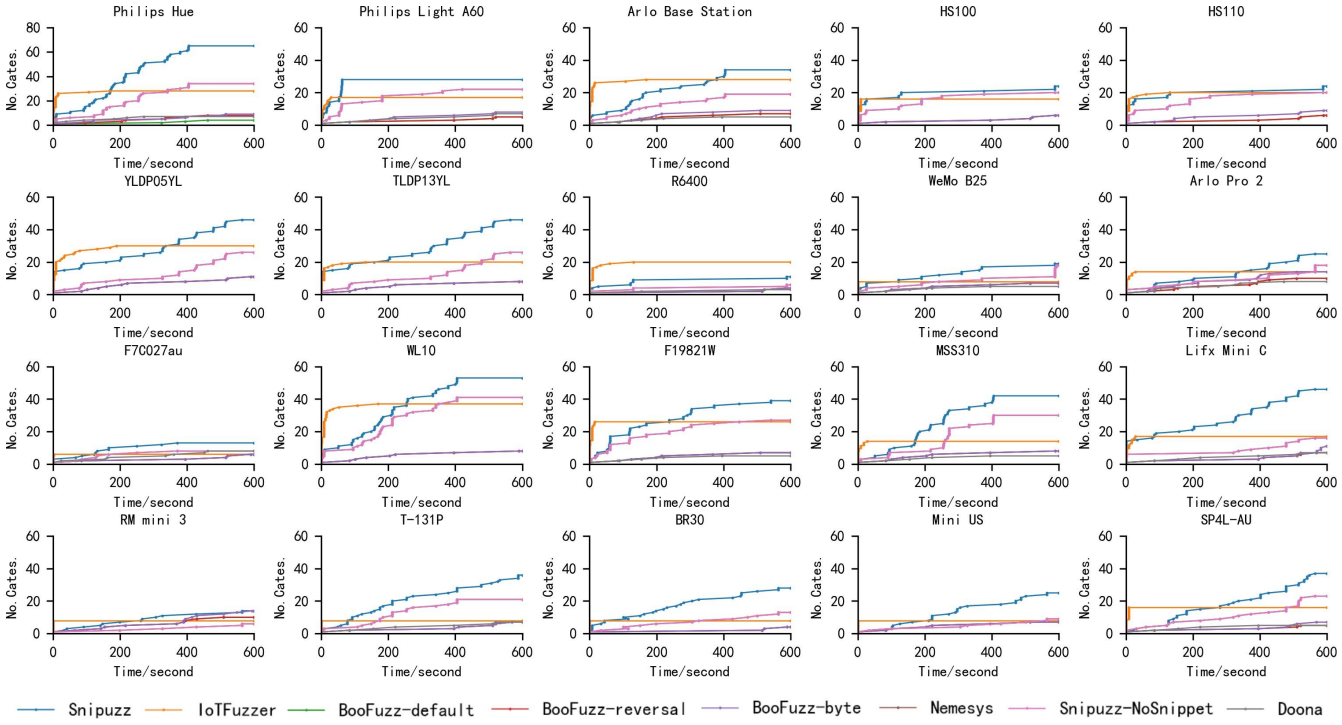

图6：运行时性能。10分钟内在所有20台物联网设备上发现的类别数量Snip uzz在19台设备上的表现最好。

在片段确定的时间上，它发现比开始时的IoTF更少的类别。然而，IoTFU zzeR很快达到顶峰，无法发现新的类别。相反，在片段确定的阶段之后，Sn I p U z逐渐发现比IoTF U z z z R和其他基线更多的类别。更详细的分析见第5.3节。

B 突变有效性：一个案例研究

TP-Link生产的HS 100和HS 110是两款经典的市场消费级智能插头。在Chen *et al.* [9]，他们使用固件版本为1.3.1的HS110来测试IoTF。他们的实验结果表明，IoTF uzze R通过改变消息中的数据域（将"light"更改**为0）触发了设备中的漏洞**。

但是，在更新版本的固件（1. 5. 2）中，IoT-F U zze R没有发现任何漏洞，但Sn I p U zz发现了。图7显示了原始输入消息和突变消息中可能触发漏洞的突变片段（红色框内）的示例在这种情况下，SnI p uzz通过破坏消息中的JSON语法结构触发了与固件输入相关的漏洞原始消息的意图是更改某些属性（*例如*，，在变异的消息中，SnI p uzz随机删除了一些内容（在蓝色框架内），这破坏了JSON语法。

```
{"method":"passthrough", "params": {"deviceId": "800...12A",
 "requestData":
    "{\"schedule\":
        {\"edit_rule\":
            {\"stime_opt\":0,\"wday\":[0,1,1,1,1,1,0],
            \"smin\":1411,\"enable\":0,
            \"repeat\":1,\"etime_opt\":-1,
            \"id\":\"27D236000E8B21C641A59DCB5C93ECBB\",
            \"name\":\"\",\"eact\":-1,
            \"month\":0,\"sact\":0,\"year\":0,
            \"longitude\":0,\"day\":0,\"force\":0,\"latitude\":0,
            \"emin\":0},\"set_overall_enable\":{\"enable\":1
            }
        }
    }"
    }
}
```

```
la    $t9, cJSON_GetObjectItem
la    $a1, aSchedule
jalr  $t9; cJSON_GetObjectItem
lw    $gp, 0x40+var_30($sp)
beqz  $v0, loc_415E7C
lui   $a1, 0x47
move  $a0, $v0
```

图8：HS110固件中的易受攻击代码片段。

这可能导致有关解析消息或传递由固件不正确处理的参数的错误，并且因此使设备崩溃。

为了进一步确定崩溃的根本原因，我们获得了固件源代码。图8显示了

固件，使用cJSON，[2]一个流行的开源轻量级JSON解析器（在GitHub中为5.4k星），来解释输入消息片段。jalr指令将cJson_GetObjectItem的结果保存在$t9中，并且无条件地跳转到该地址（参见图8中的行3），这意味着固件将选取对应于"schedule"的值。在原始消息中，对应于"schedule"的值是以"edit_rule"开头的JSON对象（从第4行到第16行）。注意，在Snipuzz中实现的上述基于片段的变异策略能够打破语法结构并且在数据域和非数据域两者上变异有趣的是，虽然移除两个左花括号破坏了JSON语法，但它不被cJSON解析器识别，因此变异的消息成功绕过语法验证并进入固件中的功能代码。当固件尝试访问"schedule"中的后继JSON对象时，*即*，对象以'edit_rule'开始，因为对应的值不再是JSON对象，而是数组，所以触发空指针异常。

由于IoTF规则的设计，基于语法规则的模糊将提供满足变异过程中的语法要求的优先级，以便不被固件语法检测器拒绝这样做的好处是确保每个测试用例都能到达固件的功能执行部分然而，在这种情况下，基于语法规则的模糊测试范围不能覆盖固件消毒部分。

综上所述，崩溃的根本原因有两个因素：1）消息语法的验证严重依赖第三方库;
2）固件没有正确处理数据类型不匹配导致虽然要求供应商完全从头开始开发产品是不合理的，但我们认为对开源库进行彻底的测试和验证是必不可少的。考虑到物联网固件测试的复杂性，迫切需要一种轻量级且有效的黑盒漏洞检测工具，如Sn IPUZZ。

[2]https://github.com/DaveGamble/cJSON