DOI:10.3969/j.issn.1671-0673. 2025. 04. 016

基于反向污点分析的设备Web接口模糊测试

邵启龙，董卫宇，马　航
（信息工程大学,河南 郑州 450001）

摘要:针对现有物联网设备Web接口模糊测试技术对危险函数不敏感以及难以满足参数控制依赖的问题,提出一种基于反向污点分析和模糊测试的物联网设备Web接口漏洞检测方法。首先,通过反向污点分析对危险函数参数至请求参数之间的路径进行可达性分析,过滤掉不能触发危险函数的请求参数;其次,根据定义的测试用例生成规则构造符合不同类型漏洞特点的测试用例;最后,使用基于远程调试的异常检测方法辅助确认漏洞。实验结果表明,在反向污点分析方面,该方法相比于SaTC提取的参数数量减少了34.3%;在漏洞检测方面,该方法共检测出23个公开漏洞,其中3个为存在控制依赖的漏洞,且漏洞检测时间均少于Boofuzz,验证了该方法的有效性。

关键词:物联网设备;危险函数;污点分析;模糊测试;漏洞检测
中图分类号:TN393.08　　**文献标识码**:A　　**文章编号**:1671-0673(2025)04-0485-06

Fuzzing for Web Interface of Devices Based on Reverse Taint Analysis

SHAO Qilong, DONG Weiyu, MA Hang
(*Information Engineering University*, *Zhengzhou* 450001, *China*)

Abstract: To address the problem that the existing fuzzing techniques for IoT device Web interface are insensitive to dangerous functions and difficult to satisfy the dependency between parameters, an IoT device Web interface fuzzing method combined with reverse taint analysis is proposed. Firstly, a reaching definition analysis is performed on the path from the parameters of dangerous functions to the request parameters through reverse taint analysis, filtering out the request parameters that cannot trigger the dangerous functions. Secondly, test cases are constructed to conform to the characteristics of different types of vulnerabilities based on the defined test case generation rules. Finally, an anomaly detection method based on remote debugging is used to assist in confirming the vulnerabilities. The experimental results show that in terms of reverse taint analysis, the number of parameters extracted by this method is reduced by 34.3% compared that of SaTC. In terms of vulnerability detection, this method detects 23 public vulnerabilities, three of which are vulnerabilities with control dependent parameters, and the vulnerability detection time is less than that of Boofuzz, which verifies the effectiveness of the method.

Key words: IoT devices; dangerous function; taint analysis; fuzzing; vulnerability detection

万物互联时代,物联网(Internet of Tings, IoT)技术广泛应用于公共事业、智能制造、智慧家具、车联网等各个领域,根据全球调研机构IoT Analytics发布的报告[1],2025年全球将有大约270亿台IoT设备。IoT设备通常在有限的硬件资源上运行嵌入式系统,由于难以部署安全防护机制,且厂商在固件开发过程中缺乏安全性检查,导致IoT设备固件中存在大量安全漏洞。Vedere Labs在研究

收稿日期:2024-11-19;**修回日期**:2024-12-27
作者简介:邵启龙(1990—),男,硕士生,主要研究方向为逆向工程与漏洞挖掘。E-mail:nightingale1997@126.com

1 900万台联网设备的基础上发布的《2024年最危险联网设备资产报告》表明：物联网设备漏洞占比从2023年的14%上升到33%，增加了136%[2]。

针对IoT设备的漏洞挖掘技术主要包括模糊测试、污点分析和符号执行等[3]。其中，模糊测试通过构造非预期的输入数据并监视目标程序在运行过程中的异常结果来发现程序中的潜在漏洞，是当前IoT设备漏洞挖掘使用最广泛的方法之一，具有自动化程度高、误报率低的特点。文献[4]通过自动解析官方移动应用的程序逻辑来生成相应的测试用例。文献[5]将测试种子字段的内容存储到权重型消息解析树（Weighted Message Parse Tree, WMPT）的节点中，通过对节点执行随机变异生成测试用例。文献[6]采用上下文驱动的方式生成测试用例，使用全系统仿真（Full-system Emulation）对目标进行持续的灰盒模糊测试。文献[7]提出了一种适用于IoT网络模糊测试的状态消息生成机制（Stateful Message Generation, SMG）。文献[8]利用模糊触发器（Fuzzing Triggers）生成不受应用端验证代码约束且结构良好的输入，从而提高模糊测试的效率。文献[9]以轻量级的程序调试工具“gdb+gdbserver”实时获取嵌入式设备固件程序执行过程中的覆盖率信息并以遗传算法模型引导测试用例进化。文献[10]使用启发式算法将每条消息大致划分为初始片段，然后使用分层聚类方法来细化消息片段，并将新的聚类进一步用于突变。

目前基于模糊测试的IoT设备漏洞挖掘技术一般采用黑盒模糊测试或者覆盖率引导的灰盒模糊测试。然而，部分参数在被后端二进制组件处理的过程中并未调用能够触发漏洞的危险函数，一些参数还存在控制依赖，现有模糊测试方法在变异时，并未对存在控制依赖的参数做出限制，在模糊测试的过程中产生了大量的无效测试用例，影响测试效率。

针对上述问题，本文提出了一种基于反向污点分析和模糊测试的IoT设备Web接口漏洞检测方法。该方法通过筛选能够触发危险函数的请求参数以及提取控制依赖参数，有效提升了模糊测试的效率和准确度。基于上述方法设计并实现了原型系统Suricatta，并利用两款设备的固件对Suricatta的反向污点分析能力及漏洞检测能力进行评估。实验结果表明，在反向污点分析方面，该方法相比于SaTC提取的参数数量减少了34.3%；在漏洞检测方面，该方法共检测出23个公开漏洞，其中3个为存在控制依赖的漏洞，且漏洞检测时间均少于Boofuzz。而Boofuzz[11]仅检测出10个公开漏洞，且无法检测存在控制依赖的漏洞。

1 研究动机

模糊测试根据程序执行反馈的获取情况，通常分为黑盒模糊测试、白盒模糊测试和灰盒模糊测试3种。由于IoT设备通常不开放源代码，难以进行白盒模糊测试。黑盒模糊测试按照约定规则对测试用例进行随机变异。灰盒模糊测试通过插桩来收集程序的覆盖率信息，主要分为覆盖率引导的灰盒模糊测试（Coverage-Based Greybox Fuzzing, CGF）和定向灰盒模糊测试（Directed Greybox Fuzzing, DGF）两种。CGF致力于最大化模糊测试的覆盖率，DGF通过能量调度引导模糊测试到给定的位置进行漏洞检测。现有的模糊测试方法没有考虑请求参数是否触发危险函数以及参数控制依赖问题，在变异的过程中，产生了大量无效测试用例。

1.1 模糊测试对危险函数不敏感

现有方法对危险函数不敏感，无法判断请求参数是否触发危险函数。对不能触发危险函数的请求参数进行测试会消耗大量时间和资源，降低模糊测试的效率。以图1中的代码为例，websGetVar函数将参数isGuest的值传递给变量v1，变量v1经atoi函数转换为整数后用于if语句的条件判断，函数后续代码不再对变量str进行处理。

```
void test(char *str)
{
    char *p;
    char v1 [512];
    p= strdup(str);
    v1 = websGetVar(wp, "isGuest", default);
    if(atoi(v1)) {
        printf("result: %s\n", p);
        free(p);
    }
}
```

图1 请求参数未触发危险函数示例

1.2 模糊测试未考虑参数依赖

目前针对IoT设备的模糊测试技术主要通过基于生成或基于变异的方式生成测试用例。基于生成的模糊测试根据生成规则生成参数的测试数据。基于变异的模糊测试通过变异策略对参数进行变异。由于难以获取参数的控制依赖信息，现有方法会产生大量不满足控制依赖约束的测试用例，导致程序难以执行到目标漏洞位置，从而降低模糊测试的效率。以图2中的代码为例，只有满足

filter 的值为 white 时，才能将 deviceName 参数的值传递给 strcpy 函数。

```
void test()
{
    char *buf;
    char *str;
    char *vl;
    buf= malloc(40);
    v1 = websGetVar(wp, "filter", default);
    str = websGetVar(wp, "deviceName", default
    if(v1 == "white") {
        strcpy(buf, str);
        printf("result: %s\n", buf);
        free(buf);
    }
    else:
        return;
}
```

图 2　请求参数的控制依赖示例

为解决上述两个问题，提出一种基于反向污点分析和模糊测试的 IoT 设备漏洞检测方法。首先，通过基于反向污点分析的参数识别方法分析危险函数到参数处理函数之间执行路径的可达性，过滤掉不能触发危险函数的参数，并提取控制依赖参数；然后，根据反向污点分析获取的请求参数信息，采用基于生成的多参数模糊测试技术对请求参数及其控制依赖参数进行动态测试。

2　基于反向污点分析的模糊测试

本文设计并实现了一种基于反向污点分析和模糊测试的 IoT 设备 Web 接口漏洞检测框架 Suricatta，整体架构如图 3 所示。框架分为基于反向污点分析的参数识别和基于生成的多参数模糊测试两个阶段。其中，基于反向污点分析的参数识别方法主要完成危险函数执行路径的可达性分析以及请求参数和控制依赖参数提取。基于生成的模糊测试根据反向污点分析提取的请求参数和控制依赖参数，通过定义的规则生成测试用例进行模糊测试，最后利用基于远程调试的异常检测辅助确认漏洞。

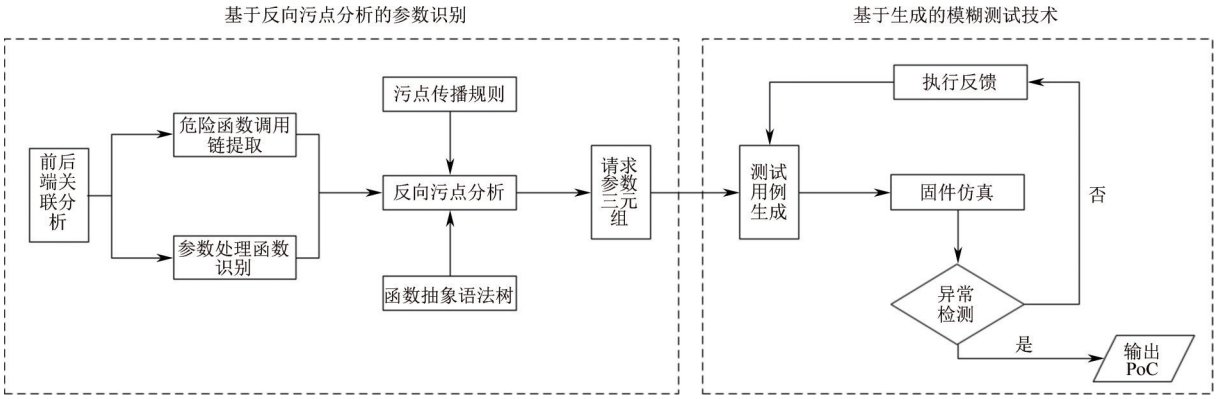


图 3　Suricatta 系统架构图

2.1　基于反向污点分析的参数识别

2.1.1　前后端关联分析

前后端关联分析负责对前端页面和后端二进制组件中的参数进行关联分析。通过前端页面分析提取提取构造测试用例所需的请求参数。通过后端文件识别负责筛选 Web 服务的后端二进制组件。

2.1.2　参数处理函数识别

参数处理函数识别负责识别 Web 服务后端二进制组件中用于处理前端请求参数的函数。在 IDA Pro 反汇编的基础上调用 IDC API 提取调用参数的函数列表；最后，通过取不同请求参数的函数调用列表的交集实现参数处理函数的识别。

2.1.3　危险函数调用链提取

危险函数调用链提取主要完成危险函数的函数调用链构建工作。首先，获取 idb 中调用危险函数的所有函数的地址；然后，遍历列表元素，通过循环遍历的方式获取对地址的所有调用；最后，当地址不能被解析为指令或者该地址不存在调用时，结束循环。

2.1.4　反向污点分析

为对危险函数的参数进行反向污点分析，首先在 Hex-Rays 反编译的基础上使用 Bip[12] 插件构建函数伪代码的抽象语法树；然后根据定义的污点传播规则，对抽象语法树进行反向污点分析，提取路径可达的请求参数及其控制依赖参数。

1）构建伪代码抽象语法树。Bip 使用两种不同的方式表示 C 函数的 AST 节点：HxCItem 和 CNode。HxCItem 等效于 IDA Pro 的 CItem_t 类，但是该对象不提供到派生它的函数的链接。为保持到父函数和父节点的单一链接，本文使用 CNode 抽象类创建 C 函数的抽象语法树，通过 buildC-

Node装饰器动态生成CNode子类来表示AST的实际节点。

2)定义污点传播规则。在对AST进行反向污点分析前,需要针对不同的CNodeStmt节点和CNodeExpr节点定义不同的污点传播规则。Bip提供了一个类层次结构,用于表示不同类型的AST节点。其中,CNodeStmt节点表示语句,CNodeExpr节点表示表达式。本节在Bip生成的伪代码抽象语法树的基础上,根据CNodeExpr节点的类型,主要定义8种数据流污点传播规则。根据CNodeStmt节点的类型,主要定义3种控制依赖污点传播规则。具体定义如表1所示。

表1 污点传播规则

节点类型	节点操作	污点传播规则	规则说明
CNodeExprCall	函数调用	$\tau(\text{first_op})\rightarrow\tau(\text{args})$	将左值的污点标记传递给函数参数
		$\tau(\text{lvar})\rightarrow\tau(\text{args})$	将变量的污点标记传递给函数其他参数
		$\tau(\text{args})\rightarrow\tau(\text{first_op})$	将函数参数的污点标记传递给左值
CNodeExprTernary	三目运算	$\tau(\text{cond})\rightarrow\tau(\text{expr1}),\tau(\text{expr2})$	将条件变量污点标记分别传递给两个表达式的变量
CNodeExprAssignment	赋值	$\tau(\text{dst})\rightarrow\tau(\text{src})$	将左值的污点标记传递给右值
CNodeExprDoubleOperation	双目运算	$\tau(\text{first_op})\rightarrow\tau(\text{second_op})$	将左值的污点标记传递给右值
CNodeExprMemAccess	内存访问	$\tau(\text{lvar})\rightarrow\tau(\text{obj})$	将变量的污点标记传递给右值
CNodeExprFinal	基础数据	$\tau(\text{lvar})\leftarrow\varnothing$	清空变量的污点标记
CNodeStmtLoop	循环判定	$\tau(\text{lvar})\rightarrow\tau(\text{cond})$	将变量的污点标记传递给条件变量
CNodeStmtIf	条件判断	$\tau(\text{lvar})\rightarrow\tau(\text{cond})$	将变量的污点标记传递给条件变量
CNodeStmtSwitch	分支选择	$\tau(\text{lvar})\rightarrow\tau(\text{expr})$	将变量的污点标记传递给表达式变量

3)反向污点分析。本文将参数处理函数的参数作为污点源,将危险函数的参数作为汇聚点。反向污点分析的主要过程为:首先,对抽象语法树进行深度优先遍历;然后,从危险函数参数所在CNode节点向前回溯,根据节点类型调用不同的污点传播规则进行数据流分析和控制依赖分析,并使用元组记录污点传播过程;最后,当污点源引入的数据能够影响汇聚点的敏感操作时,则提取接口函数名以及请求参数、控制依赖参数构成的参数列表,并以<function, parameters, type>的形式保存分析结果。其中:function是接口函数名;parameters记录了请求参数和控制依赖参数;type代表漏洞类型。

2.2 基于生成的多参数模糊测试

2.2.1 固件仿真

使用QEMU[13]对IoT设备固件进行全系统仿真,将整个基本块转换为主机系统的指令集然后执行转换后的指令,从而实现多种处理器架构的IoT设备的操作系统和硬件环境的模拟,包括处理器、内存、外设等硬件设备以及操作系统等。

2.2.2 测试用例生成

高质量的测试用例更容易触发目标系统中的潜在漏洞[14]。为避免盲目、低效的测试用例生成,基于生成的多参数模糊测试方法通过前端参数默认值构建的字典生成控制依赖参数的测试用例,根据整数溢出、缓冲区溢出以及命令注入漏洞的原理制定符合不同类型漏洞特点的测试用例生成规则,如表2所示。对于整数溢出漏洞,使用边界值作为请求参数的测试用例;对于缓冲区溢出漏洞,采用2^n重复的方式增加测试用例的长度;对于命令注入漏洞,通过增加、替换字典中预定义命令的分隔符来构造测试用例。

表2 测试用例生成规则

参数类型	漏洞类型	填充类型	生成规则
请求参数	整数溢出	整数	使用边界值填充
	缓冲区溢出	字符串	2^n重复
	命令执行	分隔符	增加、替换
		命令	使用字典进行填充
控制依赖参数		整数	使用字典进行填充
		字符串	

2.2.3 异常检测

异常检测负责检测Web服务接收测试用例后的运行状态,主要采用基于gdb远程调试的异常检测方法检测触发漏洞产生的异常信号。对于整数溢出漏洞(整数溢出通常不会直接导致任意代码执行,因此本文主要研究整数溢出导致缓冲区溢出的情况)和缓冲区溢出漏洞,通过捕获异常信号检测漏洞造成的程序异常。对于命令执行漏洞,对于命令注入漏洞,当测试用例触发设置的断点时,利用gdb调试命令获取此时目标进程的上下文

信息，然后，通过检测上下文信息中是否包含执行的命令以最终确认漏洞。

3　实验评估

3.1　实验设置

本文的实验环境为 Kali Linux Rolling 64 位操作系统，4 个虚拟 CPU，4 GB 内存的虚拟机，物理 CPU 为 AMD Ryzen 7 4800H。选取 Tenda AC15 和 TOTOLINK X5000R 两款设备的 Web 接口程序作为实验对象。其中，Tenda AC15 的固件版本为 V15.03.05.19，TOTOLINK X5000R 的固件版本为 V9.1.0u.6118_B20201102。为了验证本文方法的有效性，实验选取 SaTC 和 Boofuzz 两款当前流行的物联网设备漏洞挖掘工具，使用参数减少百分比评估 Suricatta 的反向污点分析能力，计算方式如公式(1)所示。从漏洞检测数量和漏洞触发时间两个方面评估 Suricatta 的漏洞检测能力。

$$R = \frac{K_s - K_a}{K_s} \tag{1}$$

式中：K_s 表示 SaTC 分析的请求参数个数；K_a 表示 Suricatta 通过反向污点分析提取的请求参数个数。

3.2　实验结果和分析

3.2.1　反向污点分析能力评估

本文从参数提取数量方面将 Suricatta 与 SaTC 进行对比，实验结果如表 3 所示。

表 3　参数提取数量对比

设备	接口程序	SaTC	Suricatta	*R*/%
Tenda AC15	httpd	270	161	40.4
TOTOLINK X5000R	cstecgi.cgi	133	103	22.5
平均		201	132	31.5

实验结果表明，Suricatta 对每个接口程序平均分析 132 个请求参数，比 SaTC 平均减少了 34.3%。SaTC 会对提取的所有参数进行污点传播分析，而 Suricatta 通过反向污点分析可以有效筛选能够触发危险函数的请求参数。更少的参数，能够有效降低系统的运行开销和时间消耗。

3.2.2　漏洞检测能力评估

为验证 Suricatta 的漏洞检测能力，本文从漏洞检测数量和漏洞触发时间两个方面与 Boofuzz 进行了对比，实验结果如表 4 所示。

表 4　漏洞触发情况

设备型号	漏洞编号	漏洞类型	Boofuzz		Suricatta	
			漏洞触发	触发时间/s	漏洞触发	触发时间/s
Tenda AC15	CVE-2018-18706	BoF	是	1 170	是	281
	CVE-2018-18707	BoF	否		是	638
	CVE-2018-18709	BoF	否		是	216
	CVE-2018-18727	BoF	否		是	313
	CVE-2018-18728	CI	否		是	120
	CVE-2020-10987	CI	否		是	120
	CVE-2020-13389	BoF	否		是	211
	CVE-2020-13390	BoF	是	2 527	是	360
	CVE-2020-13391	BoF	是	691	是	210
	CVE-2020-13393	BoF	否		是	2045
	CVE-2020-13394	BoF	是	1 339	是	330
	CVE-2022-40862	BoF	是	4 840	是	286
	CVE-2022-40864	BoF	否		是	331
	CVE-2024-2855	BoF	否		是	782
TOTOLINK X5000R	CVE-2021-45736	BoF	是	1 232	是	321
	CVE-2021-45738	CI	否		是	110
	CVE-2021-45741	BoF	是	6 343	是	191
	CVE-2023-31569	CI	否		是	110
	CVE-2023-33485	BoF	否		是	413
	CVE-2023-33487	CI	否		是	110
	CVE-2023-36950	BoF	是	5 261	是	277
	CVE-2023-45984	BoF	否		是	353
	CVE-2023-45985	BoF	是	2 553	是	245

表中,BoF代表缓冲区溢出漏洞,CI代表命令注入漏洞,“ ”表示不存在或时间大于24 h。

实验结果表明,Suricatta漏洞检测能力明显优于Boofuzz。Suricatta在两个型号的固件上共检测出23个公开漏洞,而Boofuzz只检测出9个公开漏洞,且Suricatta能够成功检测CVE-2018-18727、CVE-2020-13389、CVE-2024-2855等存在控制依赖参数的漏洞,Boofuzz则无法检测此类漏洞。在漏洞触发时间上,对于相同的公开漏洞,Suricatta所用时间均少于Boofuzz,可以在更短的时间内触发漏洞。原因在于Boofuzz通过字典中预定义的字符串对物联网设备Web接口进行黑盒模糊测试,生成的测试用例质量较低,难以触发漏洞。

4 结束语

提出了一种基于反向污点分析和模糊测试的IoT设备漏洞检测方法,通过基于反向污点分析的参数识别方法分析危险函数到参数处理函数之间执行路径的可达性,过滤掉不能触发危险函数的请求参数,并提取其控制依赖参数;通过基于生成的多参数模糊测试技术,根据不同的策略对请求参数和控制依赖参数进行动态测试。实验结果表明,该方法能够有效提升IoT设备Web接口漏洞挖掘的效率和准确度。

参考文献:

[1] NAIR R, WEGNER P, et al. Industrial IoT & industry 4.0 case study report[R/OL]. (2023-11-15)[2024-11-19]. https://iot-analytics.com/product/industrial-iot-industry-4-0-case-study-report-2023/.

[2] Vedere Labs. The riskiest connected deviced in 2024[R/OL]. (2024-01-10)[2024-11-19]. https://www.forescout.com/resources/2024-riskiest-connected-devices/.

[3] 于颖超,陈左宁,甘水滔,等.嵌入式设备固件安全分析技术研究[J].计算机学报,2021,44(5):859-881.

[4] CHEN J Y, DIAO W R, ZHAO Q C, et al. IoTFuzzer: discovering memory corruptions in IoT through app-based fuzzing[C]//Proceedings of the 2018 Network and Distributed System Security Symposium. Reston, USA: Internet Society, 2018. DOI: 10.14722/ndss.2018.23159.

[5] WANG D, ZHANG X S, CHEN T, et al. Discovering vulnerabilities in COTS IoT devices through blackbox fuzzing web management interface[J]. Security and Communication Networks, 2019,1:No.5076324.

[6] SRIVASTAVA P, PENG H, LI J H, et al. FirmFuzz: automated IoT firmware introspection and analysis[C]//Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things. New York, USA: ACM, 2019:15-21.

[7] ZHANG H W, LU K, ZHOU X, et al. SIoTFuzzer: fuzzing web interface in IoT firmware via stateful message generation[J]. Applied Sciences, 2021,11(7):No.3120.

[8] REDINI N, CONTINELLA A, DAS D, et al. Diane: identifying fuzzing triggers in apps to generate under-constrained inputs for IoT devices[C]//Proceedings of the 2021 IEEE Symposium on Security and Privacy. Piscataway, USA: IEEE, 2021:484-500.

[9] GAO Z C, DONG W Y, CHANG R, et al. Fw-fuzz: a code coverage-guided fuzzing framework for network protocols on firmware[J]. Concurrency and Computation: Practice and Experience, 2022,34(16):No.5756.

[10] FENG X T, SUN R X, ZHU X G, et al. Snipuzz: black-box fuzzing of IoT firmware via message snippet inference[C]//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. New York, USA: ACM, 2021:337-350.

[11] PEREYDA J. Boofuzz:network protocol fuzzing for humans[EB/OL]. (2023-10-6)[2024-09-01]. https://github.com/jtpereyda/boofuzz.

[12] KHIEM D. Bip[EB/OL]. (2020-04-14)[2024-11-19]. https://github.com/synacktiv/bip.

[13] BELLARD F. QEMU, a fast and portable dynamic translator[C]//Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference. Berkeley, USA: USENIX, 2005:41-46.

[14] 徐威,武泽慧,王子木,等.基于测试用例自动化生成的协议模糊测试方法[J].计算机科学,2023,50(12):58-65.

(编辑:冯 春)