

一种程序状态感知增强的固件模糊测试方法

孔德宝, 耿洋洋, 魏超杰, 魏 强

(信息工程大学, 河南 郑州 450001)

摘要: 针对物联网(IoT)固件符号剥离与状态逻辑复杂导致定向模糊测试(DF)路径阻塞及崩溃触发困难的问题, 提出基于程序状态感知的模糊测试方法 StateFuzz。首先, 利用静态污点分析提取与漏洞相关的关键切片; 其次, 采用基于调用图的层级语义恢复技术重构切片内语义, 并利用大语言模型(LLM)精准识别路径约束与漏洞触发两类关键变量; 最后, 构建“距离—状态”双模态协同制导机制, 结合运行时监控与动态反馈信息, 执行熵导向的能量调度与测试用例定向变异。实验结果表明, StateFuzz 较最优基线工具 AntDFuzz 在漏洞挖掘能力上提升了 25%, 测试执行效率提升了 50.5%, 并成功挖掘出 11 个未公开的 0-day 漏洞。消融实验进一步验证了各模块对提升漏洞挖掘效能的关键作用, 能够高效突破状态逻辑并精准触发漏洞。

关键词: 物联网; 固件; 定向模糊测试; 程序状态感知; 大语言模型

中图分类号: TP393.08

文献标识码: A

文章编号: 1671-0673(2026)03-0323-07

A Program State-Aware Enhanced Fuzzing Method for Firmware

KONG Debao, GENG Yangyang, WEI Chaojie, WEI Qiang

(Information Engineering University, Zhengzhou 450001, China)

Abstract: To address the issues of path blockage and crash triggering difficulties in directed fuzzing (DF) caused by symbol stripping and complex state logic in Internet of things (IoT) firmware, a program state-aware fuzzing method, StateFuzz, is proposed. Firstly, static taint analysis is utilized to extract critical slices related to vulnerabilities. Subsequently, a call-graph-based hierarchical semantic recovery technique is adopted to reconstruct semantics within the slices, and large language models (LLM) are leveraged to accurately identify two categories of critical variables which are path constraints and vulnerability triggers. Finally, a “distance-state” dual-modal collaborative guidance mechanism is constructed. By integrating runtime monitoring with dynamic feedback information, entropy-guided energy scheduling and directed mutation of test cases are executed. Experimental results demonstrate that StateFuzz improves vulnerability discovery capability by 25% compared to that of the state-of-the-art baseline tool AntDFuzz, increases execution efficiency by 50.5%, and successfully discovers 11 undisclosed 0-day vulnerabilities. Ablation experiments further verify the critical role of each module in enhancing vulnerability mining performance, demonstrating the ability to efficiently break through state logic and precisely trigger vulnerabilities.

Key words: IoT; firmware; directed fuzzing; program state awareness; large language model

0 引言

随着物联网(IoT)技术广泛应用于工业控制、智能家居等领域,其固件安全已直接关系到基础设施运行与用户隐私^[1-2]。由于开发周期短且安全标准不一,固件中潜伏的逻辑缺陷与内存破坏漏洞使其成为主要攻击目标^[3]。模糊测试(Fuzzing)作为一种通过向目标程序输入大量变异数据以触发异常崩溃的动态分析技术,已成为固件漏洞挖掘的主流手段^[4-5]。其中,定向模糊测试(Directed Fuzzing, DF)通过引导计算资源向预设的目标代码区域倾斜,在漏洞复现及安全性评估中展现出显著优势^[6-7]。

在通用模糊测试领域,AFLGo^[6]、Hawkeye^[7]等主流 DGF 工具主要依赖控制流图(Control Flow Graph, CFG)上的距离度量来调度种子,其核心假设是“距离目标越近的种子越有价值”。在固件模糊测试领域,FIRM-AFL^[8]、FirmAE^[9]等工作通过增强仿真技术提升了测试吞吐量;IoTfuzzer^[10]、Snipuzz^[11]等通过接口推断实现了黑盒测试;UCRF^[12]通过静态分析路径约束,生成满足特定输入格式的欠约束种子,进一步提升了模糊测试的初始探索能力;AntDFuzz^[13]利用污点分析定位敏感路径,解决了“测哪里”的问题。

在固件普遍符号剥离的环境下,现有工具因缺失语义导致无法区分关键状态变量,常因状态错误引发路径阻塞或漏洞触发困难^[14],本质是缺乏深层逻辑感知能力^[15-16]。虽然CSFuzz^[17]提出状态导向策略,但其严重依赖源码级ASan^[18]等工具,难以应用于无源码的物联网固件二进制环境,无法建立有效的制导机制。

本文利用大语言模型(Large Language Model, LLM)的深度代码理解能力^[19],基于反编译伪代码^[20]识别关键变量以增强状态感知。针对符号剥离导致LLM推理偏差^[21],造成语义缺失及状态感知

困难问题,提出StateFuzz方法。首先,通过静态污点分析提取关键切片;随后,利用基于调用图的层级语义恢复技术,以库函数为锚点自底向上重构上下文语义,辅助LLM精准识别路径约束与漏洞触发变量;通过运行时监控变量熵值与分支敏感度动态校准误报;最后,构建“距离—状态”双模态协同制导机制,执行差异化能量调度与定向变异,以突破逻辑阻塞并精准触发漏洞。

实验结果表明,在同一数据集上,StateFuzz较AntDFuzz在漏洞挖掘能力上提升26.7%,执行效率提升了50.5%。在针对多款真实物联网固件的测试中,该方法成功挖掘出11个0-day漏洞,充分验证了其突破复杂状态逻辑阻塞的实战价值。消融实验进一步验证了各模块对提升漏洞挖掘效能的关键作用,能够高效突破状态逻辑并精准触发漏洞。

1 方法设计

物联网固件通常以闭源且符号剥离的状态发布,使得传统依赖动静态结合或源码插桩的定向模糊测试方法难以有效感知程序内部的状态逻辑,导致路径探索受阻与崩溃触发困难的问题。针对上述问题,提出StateFuzz方法,构建“静态语义认知—动态状态校准—语义制导变异”的闭环流程。

1.1 系统概述

如图1所示,StateFuzz首先通过污点分析提取与漏洞相关的关键切片 S_c (Critical Slice),剔除冗余代码。随后,在切片内采用层级化策略重构语义,利用LLM识别路径约束与漏洞触发两类初始候选变量。最后,模糊测试阶段引入自适应状态反馈,通过运行时熵值与分支敏感度对变量集进行动态去噪与校准;并基于校准后的语义执行定向变异与“距离—状态”协同调度,引导测试用例精准突破逻辑并触发漏洞。

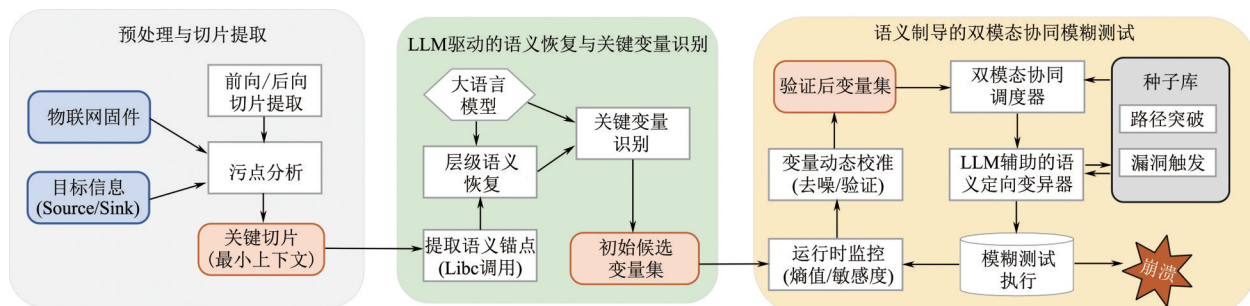


图1 StateFuzz系统框架

1.2 针对符号剥离固件反编译的语义恢复

StateFuzz 首先利用污点分析界定分析范围。通过解析网络接收函数确定外部输入源点集合 S_1 , 扫描敏感函数及内存指令确定漏洞汇点集合 S_2 。从 S_2 出发执行后向数据流分析, 将所有处于“源点—汇点”依赖路径上的函数及其传递闭包定义为关键切片 S_c 。该切片剔除了日志、UI 等冗余代码, 为语义恢复提供最小必要上下文。

在 S_c 内, 针对符号剥离导致的函数名与变量语义缺失问题, 采用自底向上的层级化语义恢复策略。如图 2 所示, 绿色节点代表语义充分的切片, 蓝色节点代表语义部分充分的切片, 红色节点为语义完全不充分的切片。该方法利用函数调用图(Call Graph)的拓扑结构, 首先通过签名匹配识别切片内的绿色叶子节点(通常为标准库函数), 这些函数已经有确切语义。随后, 将这些确定的库函数语义作为锚点向上层传播, 通过大语言模型推断上层调用者的功能摘要与函数名。

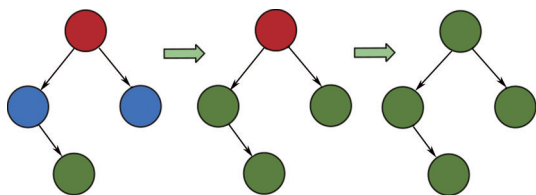


图2 层级语义传播过程示意

具体流程如算法 1 所示。首先识别切片内的叶子节点或仅依赖已知节点的中间节点作为推理锚点, 提取其所有下层锚点的语义摘要, 结合控制流结构构建提示词输入 LLM。一旦 LLM 推断出高层语义(如将 sub_40E0 识别为 process_auth), 该函数即标记为新锚点并参与更上层的推理。此过程不断迭代直至 S_c 内所有函数均获具名语义, 为精准识别关键变量奠定基础。

算法 1 基于 LLM 的层级化语义恢复算法

输入: 待分析目标函数 F_i , 全局语义锚点集合 $\mathcal{A}$ (包含标准库及已恢复函数)

输出: 推断生成的函数名与语义描述 S_i

```

1. function InferSemantics( $F_i, \mathcal{A}$ )
2. /* 步骤 1: 上下文锚点提取 */
3. Calleees  $\leftarrow$  GetCallees( $F_i$ ) /* 获取子切片列表 */
4. Context  $\leftarrow \emptyset$ 
5. for each  $c \in$  Calleees do
6.   if  $c \in \mathcal{A}$  then
7.     /* 映射底层语义, 如 read  $\rightarrow$  读取输入 */
8.     Context.add(FormatAnchor( $c, \mathcal{A}[c]$ ))
9.   end if

```

```

10. end for
11. /* 步骤 2: 构建语义提示词 */
12. Pcode  $\leftarrow$  Decompile( $F_i$ )
13. Prompt  $\leftarrow$  Prompt (Role= "Reverse Engineer",
    Context, Pcode)
14. /* 步骤 3: LLM 推理 */
15. Response  $\leftarrow$  QueryLLM(Prompt)
16.  $S_{target}$   $\leftarrow$  ParseJson(Response)
17. /* 步骤 4: 语义更新 */
18. if Verify( $S_i$ ) then
19.    $\mathcal{A}[F_i]$   $\leftarrow S_i$  /* 更新锚点集, 实现层级传播 */
20. end if
21. return  $S_i$ 
22. end function

```

1.3 LLM驱动的关键变量识别

StateFuzz 在语义恢复的基础上, 利用 LLM 对切片内的变量进行细粒度分类识别, 将关键变量定义为以下两类。

1) 路径约束变量(V_c)。决定程序控制流能否突破校验(如状态标志、校验和)并到达汇点基本块, 用于解决路径阻塞问题。

2) 漏洞触发变量(V_v)。直接作为危险操作参数(如 memcpy 的 size、数组 index), 决定操作危险程度, 用于解决崩溃触发难问题。

LLM 基于变量的定义—使用链和语义上下文输出初始候选集 V_d 。针对静态分析存在的循环计数器噪声误报, 系统后续通过运行时监控进行动态调整与去噪。

提示词 1 关键变量识别提示词

1. System: 你是一名二进制分析专家, 请根据提供的语义上下文, 识别目标代码中的关键变量。

2. Context: 语义锚点

3. 函数名: process_admin_request(处理管理员请求)

4. 已知调用: check_password(校验管理员密码)

5. 已知调用: memcpy(复制数据到缓冲区)

6. Target code:

```

7. int process_admin_request(char* input) {
8.   int v1 = *(int*)input; // 变量 A
9.   int v2 = *(int*)(input + 4); // 变量 B
10.  if (v1 != 0xADMIN) return -1; // 检查 A
11.  if (check_password(input + 8)) {
12.    char buf[128];
13.    memcpy(buf, input + 12, v2); // 使用 B
14.  }
15. }

```

16. Test: 分析变量 A 和变量 B 的作用。若控制路径可达性, 标记为“Path_Constraint”; 若控制危险操作, 标记为

“Vuln_Trigger”。

17. LLM result: {“变量A”: “Path_Constraint”, “变量B”: “Vuln_Trigger”}

提示词 1 展示了用于关键变量识别的提示词示例,这里的已知调用函数可以是本身就有语义的函数,也可以是本方法迭代恢复的函数。这些函数的函数名与摘要作为语义锚点,反编译代码作为主体,共同构成语义充足的 LLM 输入,从而极大提升 LLM 推断的准确率。在该示例中,LLM 根据语义上下文,成功将 v_1 识别为路径约束变量,将 v_2 识别为漏洞触发变量,并格式化输出结果便于后续系统自动化参数接收。

1.4 自适应状态反馈与变量动态校准

为了克服静态分析的误报,本文设计自适应状态反馈闭环,该机制并非简单的过滤,而是基于运行时行为特征,对变量的重要性权重进行动态校准。

1.4.1 运行时监控

实时采集变量的取值熵与分支敏感度两项动态特征。取值熵(Value Entropy) H 用于衡量变量取值的离散程度与随机性,其计算公式为

$$H(v) = -\sum_i P(x_i) \log_2 P(x_i) \quad (1)$$

式中, $P(x_i)$ 为时间窗口内变量 v 取值为 x_i 的频率。通常,关键状态变量表现为低熵(如枚举值),而随机噪声表现为高熵。

分支敏感度(Branch Sensitivity) S 用于衡量变量变化对路径探索的支配能力,其计算公式如下:

$$S(v) = \frac{\Delta E}{\Delta V(v)} \quad (2)$$

式中: ΔE 表示变量 v 取值变化期间触发的新边数量; $\Delta V(v)$ 表示变量 v 的不同取值数量。若变量值的改变能够频繁触发切片内新的控制流边,则表明其对路径具有强支配性。

1.4.2 关键变量集合的动态校准

系统设置周期性检查点,通过分析监控数据对候选变量进行的动态优化。首先执行噪声剔除,对于满足 $H(v) > \theta_{\text{high}}$ 且 $S(v) \approx 0$ 的变量(如循环计数器),判定为噪声并从监控列表中移除,以释放资源。其次执行状态确认,对于满足低熵且高敏感度特征的变量,确认为真实的关键变量,加入验证集 V_c 。这一过程实现了从静态假设到动态确证的认知升级,为后续的制导提供了精准的靶点。

1.5 语义制导的双模态协同模糊测试

基于动态校准后的关键变量集合 V_c , StateFuzz 构建了由粗到细的双模态协同制导机制。初期利用距离指标引导以覆盖关键切片;进入切片后切换至细粒度状态导向模式,在路径突破与漏洞触发间动态调度;并结合 LLM 语义上下文指导定向变异,克服传统测试处理复杂逻辑时的随机性与低效问题。

1.5.1 路径突破模式

当测试用例的控制流距离不再显著减少,或长期受阻于特定分支时,系统激活路径突破模式。此阶段的核心任务是利用语义信息突破控制流关卡,重点关注路径约束变量 V_c 。

1) 能量调度。系统优先调度能够促使路径约束变量发生状态跃迁的种子(seed)。能量 $E_a(s)$ 计算公式如式(3)所示:

$$E_a(s) = E_b \cdot (1 + \alpha \cdot I(V_c)) \quad (3)$$

式中: E_b 为基础能量; α 为增益系数; $I(V_c)$ 为状态跃迁指示函数,当种子导致 V_c 取值覆盖新状态时置 1。该策略旨在集中算力快速拓展状态空间。

2) 语义定向变异。针对盲目随机变异难以通过强校验(如 Magic Bytes)的问题, StateFuzz 利用 LLM 构建局部语义字典。具体而言,系统解析 V_c 的语义标签。例如:若识别为“协议标志”,则生成字典{“GET”, “POST”, “HEAD”};若识别为“魔数检查”,则提取汇编指令中的比较立即数(如 0x5AA5)。Fuzzer 将变异算子限制在修改种子中对应 V_c 偏移量的字节,并强制替换为语义字典中的值,从而高效打通通往汇点的路径。

1.5.2 漏洞触发模式

当测试用例成功覆盖至危险点所在切片时,系统切换至漏洞触发模式。此阶段的核心任务是在保持路径可达的前提下最大化危险操作,重点关注漏洞触发变量 V_l 。

1) 能量调度。系统优先调度能够影响漏洞触发变量 V_l 的种子。能量 $E_a(s)$ 计算公式如下式:

$$E_a(s) = E_b \cdot \beta \cdot f_e(V_l) \quad (4)$$

式中: E_b 为基础能量; β 为权重系数; $f_e(V_l)$ 的取值趋向于边界(如极大整数或极长字符串)或者包含 Shell 元字符(如“;”、“|”、“\$”)等高风险字符时,给予更高能量。

2) 语义定向变异。基于 LLM 对 V_l 的类型推断执行感知变异。例如:若 V_l 被识别为 memcpy 的 size 参数(整型),变异器将重点施加整数溢出策略

(如 MAX_INT, 0, -1);若 V_i 为缓冲区内容,则采用模式重复策略生成超长字符串。特别地,此阶段实施约束保持策略,通过锁定路径约束变量 V_i 的相关字节,防止变异破坏路径可达性,集中算力对 V_i 进行极限测试,最大化汇点参数的危险性,从而触发潜在的内存破坏漏洞。

2 系统实现与实验评估

为了验证 StateFuzz 提出的状态感知定向模糊测试方法的有效性,本文设计并实现了原型系统,并在真实物联网固件数据集上进行了广泛的实验评估。实验旨在回答以下 3 个核心问题。

1) StateFuzz 在真实环境下的漏洞挖掘能力如何?

2) 与现有的主流定向模糊测试工具相比, StateFuzz 在漏洞复现效率上是否有显著提升?

3) 系统中的各模块是否起到了预期作用?

2.1 原型系统实现

StateFuzz 原型系统由静态分析与动态模糊测试两个子系统构成。静态子系统通过编写 IDA Pro 的 IDAPython 脚本自动化提取控制流图与调用图,集成 Emtaint 引擎执行污点分析。编程语言选择为 Python。调用 DeepSeek-V3.2 模型的 20251201 版本作为 LLM 的 API 进行语义增强与变量推断,具体调用方式为通过 DeepSeek 官方提供的 OpenAI 兼容 API 接口进行远程推理,温度系数(Temperature)为 0.3,以确保逻辑推断的稳定性,最大生成长度(Max Tokens)设为 1 024。采用 QEMU 模式仿真固件,并设计了自定义 Hooks 函数,用于实时监控关键变量的熵值与分支敏感度。

2.2 实验设置

实验选取了来自 Tenda、D-Link、Netgear、Wav-

link、TOTOLINK 等知名厂商的 7 款基于 Linux 的路由器固件作为测试数据集。

如表 1 所示,这些固件涵盖了 MIPS 和 ARM 两种主流架构,且包含 httpd、boa 等常见的 Web 服务程序,具有广泛的代表性。

表 1 测试数据集

设备厂商	设备型号	二进制	架构	尺寸/MB
Tenda	TX3	httpd	ARM	40.8
	TX9	httpd	MIPS	11.3
	AC10	httpd	MIPS	7.3
D-LINK	DIR-823G	goahead	MIPS	6.2
Netgear	R7000	httpd	ARM	49.0
Wavlink	WN579A3	wireless.cgi	MIPS	9.6
TOTOLINK	A3002R	boa	MIPS	6.6

为了评估 StateFuzz 的性能,选取了 3 款主流模糊测试工具作为基准对比。

1) AFLGo:经典的定向模糊测试工具,主要依赖控制流距离进行引导。

2) UCRF:通过静态路径分析,提取分支跳转的约束条件,并据此生成满足特定输入格式约束要求的初始种子。

3) AntDFuzz:最新的基于静态分析敏感路径辅助的固件二进制定向模糊测试工具。

2.3 真实漏洞挖掘能力评估

为了验证 StateFuzz 在真实场景下的未知漏洞挖掘能力,本文选取 Tenda、Wavlink 等品牌的 5 款主流路由器固件进行实战测试。实验期间,StateFuzz 共触发了 15 条导致系统服务崩溃的未知漏洞,经人工分析与复核,最终确认了 11 个真实有效的 0-day 漏洞,其中包括 6 个缓冲区溢出漏洞和 5 个命令注入漏洞,详细信息如表 2 所示,充分证明了 StateFuzz 在复杂固件环境下的实战价值。

表 2 StateFuzz 发现的 0-day 漏洞列表

设备	漏洞编号	类型	触发位置
Tenda-TX3	待披露	缓冲区溢出	fromSetIpMacBind
	待披露	缓冲区溢出	sub_42D334
Tenda-TX9	待披露	缓冲区溢出	sub_432580
	待披露	缓冲区溢出	sub_42359C
	待披露	命令注入	sub_4022B4
Wavlink-WN579A3	待披露	命令注入	sub_401270
	待披露	命令注入	sub_403464
	待披露	命令注入	sub_403100
	待披露	命令注入	sub_4033B4
Tenda-AC10	CNVD-2024-25726	缓冲区溢出	sub_415C20
Tenda-X2000R	CNVD-2024-25541	缓冲区溢出	sub_403A82

2.4 漏洞挖掘有效性分析

本节从漏洞数量与任务效率维度,对比 StateFuzz 与 AntDFuzz、UCRF、AFLGo 的检测效能。表 3

列出了各工具挖掘的漏洞数和总耗时。StateFuzz 与 AntDFuzz 统计路径收敛时间,UCRF 与 AFLGo 因缺乏语义制导导致变异效率低,设定为 24 h 固定周期。

表 3 各方法在目标固件上的漏洞挖掘效果对比

固件设备	漏洞数				总耗时			
	StateFuzz	AntDFuzz	UCRF	AFLGo	StateFuzz	AntDFuzz	UCRF	AFLGo
TX3	21	15	12	5	3 h55 min	4 h35 min	24 h	24 h
TX9	28	23	21	7	4 h11min	7 h12 min	24 h	24 h
AC10	27	23	20	6	6 h11 min	9 h23 min	24 h	24 h
DIR-823G	15	13	11	6	2 h03 min	2 h41 min	24 h	24 h
R7000P	24	19	17	4	11 h45 min	18 h23min	24 h	24 h
WN579A3	27	21	18	12	7 h41 min	11h33min	24 h	24 h
A3002R	18	14	12	7	2 h55 min	4 h27 min	24 h	24 h
总计	160	128	111	47	38.68 h	58.23 h	168 h	168 h

在漏洞挖掘的广度上,StateFuzz 展现出显著优势,总计发现 160 个漏洞,相比 AntDFuzz 的 128 个漏洞提升了 25%。这种差异主要源于 AntDFuzz 仅靠污点分析追踪流向,难以解析复杂数据约束;而 StateFuzz 利用 LLM 语义字典实现了语义制导,有效解决了深层路径约束满足问题。在执行效率上,以 7 款固件总收敛耗时数据可得,StateFuzz 的测试执行效率较 AntDFuzz 提升了 50.5%。尽管预处理引入了少量 LLM 开销,但在后续测试中转化为了巨大的执行收益,通过语义制导规避了海量无效变异,实现了更高的投入产出比。

2.5 消融实验

为了进一步探究 StateFuzz 中各个核心组件的贡献度,以逻辑复杂度较高的 Tenda AC10 固件的 httpd 模块为对象设计消融实验,旨在验证各个模块是否取得预期效果。

2.5.1 实验变体与指标定义

实验构建 3 个独立变体:StateFuzz-NoLLM 移除语义模块,仅依赖静态切片与随机变异,模拟缺乏语义时的盲目探索;StateFuzz-NoCalib 移除动态校准模块,将所有静态候选变量纳入制导,模拟缺乏反馈的粗粒度制导;StateFuzz 为完整系统。评估指标包括关键切片覆盖率(反映路径约束突破)和崩溃数量(反映触发能力)。

2.5.2 实验结果与机理分析

各变体在相同条件下运行 24 h,结果如表 4 所示,关键变量数中“静态”指 LLM 初始识别数量,“运行时”指经动态校准后实际用于制导的数量。

StateFuzz-NoLLM 变体的测试效能显著低于其他组别,其切片覆盖率仅为 35.4%,且未检出任何

表 4 Tenda AC10 固件上的消融实验结果

消融变体	切片覆盖率/%	崩溃数量	关键变量数(静态/运行时)
StateFuzz-NoLLM	35.4	0	0 / 0
StateFuzz-NoCalib	62.1	8	128 / 128
StateFuzz	94.2	19	128 / 23

有效崩溃。这一现象归因 StateFuzz-NoLLM 变体缺乏语义先验的随机变异,难以通过 Tenda AC10 的前置校验,且无法感知漏洞触发变量对特殊值的依赖,导致测试停滞于浅层解析逻辑。StateFuzz-NoCalib 引入语义后覆盖率升至 62.1%,但仅发现 8 个崩溃。其缺陷在于将 128 个候选变量全部纳入调度,导致算力被大量循环计数器等噪声变量平摊,核心变量变异不充分。相比之下,完整版 StateFuzz 利用运行时反馈剔除 105 个噪声变量,实现算力精准聚焦。在变量数降至 23 个的情况下,漏洞产出提升 1.4 倍,证实了剔除无效变量对于提升定向测试效率的必要性。

3 结束语

本文针对物联网固件符号剥离与状态逻辑复杂导致定向模糊测试路径阻塞及崩溃触发困难的问题,提出程序状态感知增强的方法 StateFuzz。通过构建层级语义恢复机制与“距离—状态”双模态协同制导体系,有效弥补了固件二进制语义缺失的缺陷,突破了复杂状态逻辑导致的路径阻塞。然而对于非标准固件环境,语义恢复的准确率仍有提升空间。未来工作将探索如何利用 LLM 的长文本建模能力,实现更深层级的协同变异,并进一

步优化跨架构仿真的兼容性,以应对更加复杂的物联网安全挑战。

参考文献:

- [1] ALI AL-GARADI M, MOHAMED A, AL-ALI A K, et al. A survey of machine and deep learning methods for Internet of Things (IoT) security [J]. *IEEE Communications Surveys & Tutorials*, 2020, 22(3): 1646-1685.
- [2] KHAN S, MAZHAR T, SHAHZAD T, et al. Antenna systems for IoT applications: a review [J]. *Discover Sustainability*, 2024, 5(1): No.412.
- [3] COSTIN A, ZADDACH J, FRANCILLON A, et al. A large-scale analysis of the security of embedded firmwares [C]//*Proceedings of the 23rd USENIX Conference on Security Symposium*. Berkeley, USA: USENIX Association, 2014: 95-110.
- [4] 邵启龙,董卫宇,马航.基于反向污点分析的设备Web接口模糊测试[J]. *信息工程大学学报*, 2025, 26(4): 485-490.
- [5] 赵栋梁,顾纯祥,郑永辉,等.基于黑盒模糊测试的安全协议逻辑漏洞检测[J]. *信息工程大学学报*, 2025, 26(1): 105-112.
- [6] BÖHME M, PHAM V T, NGUYEN M D, et al. Directed greybox fuzzing [C]//*Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, USA: ACM, 2017: 2329-2344.
- [7] CHEN H X, XUE Y X, LI Y K, et al. Hawkeye: towards a desired directed grey-box fuzzer [C]//*Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. New York, USA: ACM, 2018: 2095-2108.
- [8] ZHENG Y W, DAVANIAN A, YIN H, et al. FIRM-AFL: high-throughput greybox fuzzing of IoT firmware *via* augmented process emulation [C]//*Proceedings of the 28th USENIX Conference on Security Symposium*. Berkeley, USA: USENIX Association, 2019: 1099-1114.
- [9] KIM M, KIM D, KIM E, et al. FirmAE: towards large-scale emulation of IoT firmware for dynamic analysis [C]//*Proceedings of the 36th Annual Computer Security Applications Conference*. New York, USA: ACM, 2020: 733-745.
- [10] CHEN J Y, DIAO W R, ZHAO Q C, et al. IoTfuzzer: discovering memory corruptions in IoT through app-based fuzzing [C]//*Proceedings 2018 Network and Distributed System Security Symposium*. San Diego, USA: Internet Society, 2018. DOI: 10.14722/ndss.2018.23159.
- [11] 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.
- [12] QIN C, PENG J Q, LIU P Z, et al. UCRF: static analyzing firmware to generate under-constrained seed for fuzzing SOHO router [J]. *Computers & Security*, 2023, 128: No. 103157.
- [13] CUI X W, WANG Y C, WEI Q. Sensitive target-guided directed fuzzing for IoT web services [J]. *Computers, Materials & Continua*, 2025, 83(3): 4939-4959.
- [14] REDINI N, MACHIRY A, WANG R Y, et al. Karonte: detecting insecure multi-binary interactions in embedded firmware [C]//*Proceedings of the 2020 IEEE Symposium on Security and Privacy*. Piscataway, USA: IEEE, 2020: 1544-1561.
- [15] MIFTAH S S, SRIVASTAVA A, KIM H, et al. SymbFuzz: symbolic execution guided hardware fuzzing [C]//*Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*. New York, USA: ACM, 2025: 1477-1490.
- [16] WANG Y H, JIA X K, LIU Y W, et al. Not all coverage measurements are equal: fuzzing by coverage accounting for input prioritization [C]//*Proceedings of the 2020 Network and Distributed System Security Symposium*. San Diego, USA: Internet Society, 2020. DOI: 10.14722/ndss.2020.24422.
- [17] CHEN X, CUI N N, PAN Z, et al. Critical variable state-aware directed greybox fuzzing [C]//*Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering*. Piscataway, USA: IEEE, 2025: 141-152.
- [18] SEREBRYANY K, BRUENING D, POTAPENKO A, et al. AddressSanitizer: a fast address sanity checker [C]//*Proceedings of the 2012 USEIX Conference on Annual Technical Conference*. Berkeley, USA: USENIX Association, 2012: No.12.
- [19] BROWN T B, MANN B, RYDER N, et al. Language models are few-shot learners [DB/OL]. (2020-05-28) [2026-03-18]. <https://arxiv.org/abs/2005.14165>.
- [20] CHEN Q B, LACOMIS J, SCHWARTZ E J, et al. Augmenting decompiler output with learned variable names and types [DB/OL]. (2021-08-13) [2026-03-18]. <https://arxiv.org/abs/2108.06363>.
- [21] HE J X, IVANOV P, TSANKOV P, et al. Debin: predicting debug information in stripped binaries [C]//*Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. New York, USA: ACM, 2018: 1667-1680.

(编辑: 冯 春)