

VxFuzzer: 基于全系统模拟的VxWorks灰盒模糊测试器

李松涛¹, 耿洋洋², 麻荣宽², 贾岩³, 彭建山², 曹琰^{1,4}

(1. 郑州大学网络空间安全学院, 河南 郑州 450001; 2. 信息工程大学, 河南 郑州 450001;

3. 南开大学密码与网络空间安全学院, 天津 300350; 4. 网络空间安全教育部重点实验室, 河南 郑州 450001)

摘要: VxWorks 作为一款被广泛应用于工业控制、航空航天等关键基础设施领域的实时操作系统, 其安全性直接关系到整个工业系统的稳定运行。然而, 由于执行环境受限、系统实现复杂及与现有工具兼容性差等问题, 对 VxWorks 系统进行模糊测试仍面临较大挑战。为应对这些挑战, 研究人员提出并实现了一种基于 QEMU 全系统仿真的模糊测试框架 VxFuzzer。该框架在 QEMU 全系统仿真的基础上, 集成了基本块插桩、异常状态监控和快照恢复功能, 能够全面感知目标程序的运行状态。同时, 该框架具备良好的兼容性, 能够与 AFL、Boofuzz 等多种模糊测试工具协同工作, 从而在 VxWorks 系统中实现自动化漏洞挖掘。实验结果表明, 即使在目标程序基本块数量超过 10^5 的复杂场景下, VxFuzzer 仍能稳定地记录程序覆盖率的变化, 并有效地检测运行过程中的异常行为, 展现出优异的覆盖率监控与异常检测性能。在实际测试中, VxFuzzer 成功发现了 4 个安全漏洞, 其中包括 URGENT/11 系列中的两个已知漏洞和一个 0-Day 漏洞, 验证了该框架在工业级嵌入式系统安全检测中的可行性与有效性。

关键词: VxWorks; 模糊测试; 全系统仿真; 状态监控; 异常监控; 漏洞挖掘

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2025046

VxFuzzer: Grey-Box fuzzing for VxWorks based on whole system simulation

LI Songtao¹, GENG Yangyang², MA Rongkuan², JIA Yan³, PENG Jianshan², CAO Yan^{1,4}

1. School of Cyber Science and Engineering, Zhengzhou University, Zhengzhou 450001, China

2. Information Engineering University, Zhengzhou 450001, China

3. College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China

4. Key Laboratory of Cybersecurity, Ministry of Education of China, Zhengzhou 450001, China

Abstract: VxWorks, a real-time operating system widely deployed in critical infrastructure sectors such as industrial control and aerospace, was studied in terms of its security due to its direct impact on the stable operation of en-

收稿日期: 2025-05-09; 修回日期: 2025-07-15

通信作者: 耿洋洋, young9471@163.com

基金项目: 网络空间安全教育部重点实验室开放基金 (KLCS20240211); 中央高校基本科研业务费专项资金 (079-63253221)

Foundation Items: Open Foundation of Key Laboratory of Cyberspace Security, Ministry of Education of China (KLCS20240211), The Fundamental Research Funds for the Central Universities (079-63253221)

引用格式: 李松涛, 耿洋洋, 麻荣宽, 等. VxFuzzer: 基于全系统模拟的 VxWorks 灰盒模糊测试器[J]. 网络与信息安全学报, 2025, 11(4): 117-126.

Citation Format: LI S T, GENG Y Y, MA R K, et al. VxFuzzer: Grey-Box fuzzing for VxWorks based on whole system simulation[J]. Chinese Journal of Network and Information Security, 2025, 11(4): 117-126.

tire industrial systems. However, fuzz testing of VxWorks remained highly challenging owing to constraints in its execution environment, the complexity of system implementation, and poor compatibility with existing tools. To address these challenges, a fuzz testing framework named VxFuzzer was proposed and implemented based on full-system simulation using QEMU. Built upon QEMU's full-system simulation capability, this framework integrated basic block instrumentation, exception status monitoring, and snapshot recovery functions, enabling comprehensive perception of the target program's execution status. Moreover, VxFuzzer was designed with high compatibility, allowing it to collaborate with multiple fuzzing tools such as AFL and Boofuzz, thereby facilitating automated vulnerability discovery in VxWorks systems. Experimental results show that even in complex scenarios where the number of basic blocks in the target program exceed one million, VxFuzzer could stably track changes in program coverage and effectively detect abnormal behaviors during execution, demonstrating excellent performance in both coverage monitoring and anomaly detection. In practical testing, VxFuzzer successfully identifies four security vulnerabilities, including two known vulnerabilities from the URGENT/11 series and one 0-day vulnerability, confirming the feasibility and effectiveness of the framework in security testing for industrial-grade embedded systems.

Keywords: VxWorks, fuzzing test, full system simulation, status monitoring, exception monitoring, vulnerability mining

0 引言

VxWorks 作为嵌入式实时操作系统 (real-time operating system, RTOS), 凭借其高实时性、资源占用少和高可靠性等特性, 在航空航天、工业控制和医疗设备等领域得到广泛应用^[1]。然而, 该系统的闭源特性与复杂架构为安全漏洞检测带来了显著挑战。2019 年, Armis 安全团队披露了 Urgent/11 漏洞 (包含 CVE-2019-12256 至 CVE-2019-12266), 揭示了 VxWorks 在无用户交互场景下存在的远程代码执行风险, 这一发现凸显了加强对该系统安全研究的必要性。

模糊测试作为一种高效的自动化漏洞挖掘技术^[2-4], 其核心原理是通过向目标程序注入随机化输入并监测异常行为来实现漏洞检测^[5]。

模糊测试生成测试用例的方法主要有两种^[6]。①基于变异的方法: 该方法通过监控程序运行时的路径信息获取反馈, 用于指导后续测试用例的生成。其代表工具有 AFL、LibFuzzer 和 T-Fuzz^[7]。②基于生成的方法: 该方法依据定义的输入格式或者协议规范, 构造大量符合语义的测试用例。其代表工具有 Peach^[8]和 Boofuzz。

尽管模糊测试技术在 Linux、Windows 等主流操作系统中测试效果显著^[9-10], 但在 VxWorks 系统场景下仍面临三重挑战, 具体如下。

1) 执行环境构建困难: VxWorks 通常部署在资源受限的嵌入式设备中, 其运行高度依赖专用

硬件。这种硬件依赖性导致测试环境难以在通用计算平台上准确复现, 极大地增加了模糊测试的复杂性。

2) 程序状态难以监控: 由于 VxWorks 属于闭源系统, 其内核 API 与数据结构信息均未公开。这使得传统插桩技术无法获取代码路径覆盖率、动态内存分配状态等关键反馈信息。

3) 现有工具链难以适配: 主流模糊测试工具通常依赖通用操作系统的调试接口或编译工具链, 而 VxWorks 采用专有的内核机制, 两者存在显著的兼容性差距。

为应对上述挑战, 本文提出一种基于全系统模拟的 VxWorks 灰盒模糊测试框架 VxFuzzer。该框架采用前后端分离的设计思想, 将测试用例生成模块与程序状态监控模块解耦, 分别构建为前端和后端组件。在后端组件中, VxFuzzer 基于 QEMU^[11]构建了 VxWorks 全系统模拟环境, 并设计了状态监控 (负责实时追踪目标程序运行状态)、异常监控 (专注于检测系统异常行为) 与快照恢复 (确保测试过程的可回溯性) 三大核心模块。在前端组件中, VxFuzzer 对主流模糊测试工具 AFL 和 Boofuzz 进行了针对性改进, 利用后端实时反馈的状态信息优化测试策略, 从而显著提升异常检测的准确性和测试效率。

为全面评估 VxFuzzer 的有效性, 本文设计了多维度的实验验证方案。实验主要分为以下 3 个部分。

1) 框架性能与功能验证: 在不同版本的VxWorks系统上测试VxFuzzer的基础性能指标, 包括内核启动时间、执行的基本块数量、函数调用数量及代码覆盖率。同时, 对比分析AFL前端和Boofuzz前端在相同测试时长下的路径覆盖率变化, 以验证框架核心功能的有效性。

2) 异常检测能力评估: 通过构造并输入300余个测试用例, 其中在200余个用例中发现错误, 证明异常监控模块能够准确识别目标系统的异常行为。

3) 真实漏洞挖掘验证: 在实际漏洞复现与挖掘场景中, VxFuzzer成功复现了VxWorks系统中的多个已知漏洞, 包括CVE-2015-7599、URGENT/11系列中的CVE-2019-12255和CVE-2019-12260。此外, 该框架还在应用程序libmodbus中发现了一个新的0-Day漏洞(CNVD-2025-09289), 进一步验证了其漏洞挖掘能力。

综上所述, 本文的贡献如下。

1) 设计并实现了VxFuzzer框架, 在QEMU全系统仿真环境下实现对VxWorks操作系统的内核与应用程序的模糊测试。

2) 扩展了VxWorks的异常状态监控功能, 通过启发式分析有效提升了异常识别能力。

3) 在实际环境中成功复现多个已知漏洞, 其中包含URGENT/11系列等漏洞, 并在libmodbus中发现1个未被披露的0-Day漏洞。

1 相关工作

本节主要介绍RTOS模糊测试方法及QEMU在二进制模糊测试中的应用。

1.1 RTOS模糊测试方法与工具

近年来, 针对嵌入式RTOS的模糊测试研究逐步深入, 但在VxWorks这类闭源、资源受限的RTOS中仍面临独特挑战。现有的针对VxWorks系统的模糊测试方法可以分为以下两类^[12-13]。

1) 轻量化静态分析驱动方法: Chen等^[14]提出一种基于程序切片的模糊测试方法SFuzz。其通过静态分析从RTOS固件中提取感兴趣的程序切片, 再借助Unicorn^[15]实现切片代码的指令级别模拟。SFuzz的优点是无须构建完整的运行环境即可对关键代码路径进行有效的模糊测试。

然而, 该方法过度依赖静态分析与静态切片的准确性, 且执行过程中上下文缺失容易导致误报。

2) 全系统模拟方法: 邵帅^[16]提出一种基于QEMU全系统模拟的模糊测试方法vxAFL。vxAFL集成了AFL与QEMU, 利用QEMU内部的GDB Server对目标程序进行输入注入, 并依赖VxWorks内部异常处理机制捕获目标程序异常行为。然而, 该方法依赖静态分析确定程序输入点, 且崩溃捕获高度依赖VxWorks内部的异常报告机制, 容易产生漏报。

与上述方法不同, 本文提出的VxFuzzer基于QEMU实现对VxWorks的全系统模拟, 但在此基础上对AFL进行了改进, 使其能够直接与目标应用程序通信。此外, 扩展了VxWorks异常监控功能, 采用启发式方法识别内存错误^[17], 提升了异常识别能力。

1.2 QEMU在二进制模糊测试中的应用

QEMU是一款基于动态二进制翻译(dynamic binary translation, DBT)^[18-19]的模拟器, 支持多种处理器架构(如X86、ARM、MIPS、PowerPC等), 可以实现全系统仿真和Linux用户模式仿真。在仿真运行过程中, QEMU首先将目标架构的二进制指令翻译为TCG(tiny code generator)中间代码, 然后将TCG中间代码动态转换为主机架构可执行的机器指令。为提升执行效率, QEMU在翻译过程中以基本块为单位, 并将翻译结果缓存在哈希表中。同时, 将多个连续执行的基本块通过跳转指令连接, 从而减少执行过程中上下文切换的开销。基于这一翻译机制, 只需在基本块翻译阶段插入自定义逻辑, 即可实现对目标程序的动态监控。目前已有多项研究在此基础上对QEMU进行扩展与应用。

FIRM-AFL^[20]是首个面向IoT(Internet of things)固件设计的基于代码覆盖率的灰盒模糊测试器。其通过在DECAF框架^[21]中引入模糊测试引擎, 并结合Linux用户态与内核态的特性, 实现了系统模式与用户模式的动态转换, 从而提高模糊测试效率。然而, 这些工作高度依赖操作系统, 无法应用于VxWorks这类高度定制化的RTOS。

AFL++^[22]是一种非常先进的模糊测试工具，被广泛应用于源代码与二进制程序的自动化漏洞挖掘。在黑盒测试场景下，AFL++集成了QASan（一种基于QEMU的堆内存错误检测技术）^[23]。QASan研究团队后续又提出了QM-San^[24]，进一步增强了AFL++在二进制模糊测试中对堆内存错误的感知能力。然而，AFL++主要面向Linux用户态程序，且高度依赖Linux系统接口（如Fork机制），无法直接应用于VxWorks系统。

Avatar^[25-26]框架将真实硬件设备与QEMU模拟器结合，实现嵌入式固件的混合执行与动态执行。HALucinator^[27]通过识别与替换硬件抽象层HAL接口，实现裸机固件的动态模拟执行。然而，这两种方法均未针对VxWorks系统进行适配，需要进行大量的定制化改造。

本文提出的VxFuzzer基于QEMU对VxWorks目标程序进行状态监控，并对AFL进行了针对性改进（包括模拟ForkServer行为、替换Fork机制及自定义程序状态通信等），使AFL能够直接在VxWorks系统中进行模糊测试。

2 系统设计与实现

本节从方法设计的角度阐述VxFuzzer的体系架构与关键模块实现。VxFuzzer整体架构如图1所示，VxFuzzer采用前后端分离模式：前端组件负责根据反馈信息生成并保存测试用例，后端组件在全系统仿真环境中执行目标程序、采集运行状态，并将分析结果实时反馈给前端。

2.1 后端组件

后端由基本块追踪模块、状态监控模块、快

速快照模块3部分构成，分别承担路径反馈、异常检测及执行环境快速恢复的核心功能。

2.1.1 基本块追踪模块

基本块追踪模块旨在为模糊测试提供精细化的路径反馈，并为后续异常分析提供上下文支持。其技术实现如下。

1) 基本块识别与插桩：利用QEMU以基本块为单位翻译执行的特性，在翻译阶段为每个基本块注入监控逻辑，记录基本块入口地址并在共享内存中维护路径位图，从而精确统计程序覆盖率。

2) 系统状态采集：QEMU模拟执行本质上是模拟CPU在运行时的状态变化。QEMU使用包含CPU运行状态的统一结构模拟程序执行期间寄存器的状态变化。VxFuzzer注入监控逻辑，从保存CPU状态的结构体中收集运行时的上下文信息（包括程序寄存器状态变化、栈帧变化等）。

3) 函数调用链监控：QEMU为不同架构（如X86与ARM）的函数调用指令提供专门的翻译接口（如X86架构的gen_CALL与ARM架构的trans_BL等）。在上述接口中插入监控逻辑，即可提取函数入口、目标地址及栈帧边界，实现跨基本块的调用链重建。

通过上述方法为前端提供实时上下文信息，助力生成针对性测试用例，以触发更多尚未覆盖的执行路径。

2.1.2 状态监控模块

状态监控模块旨在实时监控目标程序的状态并识别潜在的异常。核心功能包括：①捕获运行过程中的崩溃、非法内存访问等关键异常事件；

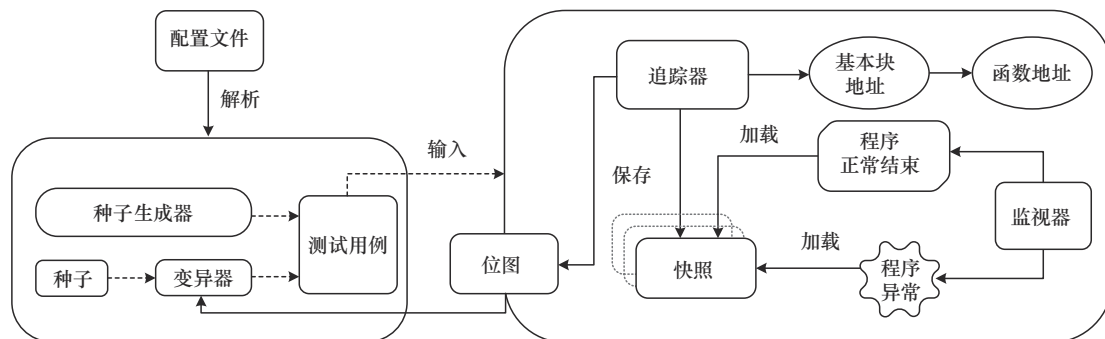


图1 VxFuzzer系统架构

Figure1 VxFuzzer system architecture

②基于函数调用信息检测潜在的栈溢出漏洞, 重构函数调用链; ③为前端组件实时反馈测试用例的执行状态。具体技术实现如下。

1) 异常行为监控。VxWorks 系统在运行期间通过异常处理函数 (如 `excStub1`) 接管大多数运行错误, 此类异常不会直接引发系统崩溃。为增强系统的可靠性, 状态监控模块采用指令级插桩技术, 在异常处理函数的入口点进行动态监测。该设计确保系统能在指令执行初期精准识别异常流程的触发。一旦捕获异常, 系统立即记录异常类型、发生位置及相关寄存器上下文等信息, 以供后续分析。

2) 基于调用行为的栈帧异常检测。部分溢出行为会绕过 VxWorks 的异常处理流程 (如少量字节的栈溢出修改了返回地址的低字节但仍落在合法地址范围内)。为识别此类隐蔽漏洞, 本文设计了一种启发式检测算法, 如算法 1 所示。

该算法在程序执行过程中动态构建函数调用树, 并记录每个函数的栈帧范围。当检测到涉及内存操作的危险函数 (如 `memcpy`、`strcpy`、`sprintf` 等) 被执行时, 算法将提取目标写入地址及写入长度, 并回溯整棵调用树, 查找目标地址的栈帧范围。若写入范围超过目标栈帧边界, 则视为潜在的栈溢出漏洞, 立即报告异常, 并输出对应的函数调用链, 辅助漏洞定位。

算法的时间复杂度为 $O(MN)$, 其中 N 为指令总数, M 为函数调用链的最大深度; 空间复杂度为 $O(N)$, 主要用于存储函数栈帧信息和调用树结构。由于实际嵌入式程序中调用链通常较浅, 算法时间复杂度趋向于 $O(N)$ 。

算法 1 启发式栈溢出检测

输入 I: 目标程序指令

输出 R: 栈溢出检测结果

```

1) CallTree  $\leftarrow \emptyset$ , StackMap  $\leftarrow \emptyset$ 
2) for instr  $\in$  I do
3)   if isCall(instr) then
4)     Func  $\leftarrow$  resolve_target(instr)
5)     StackMap[Func]  $\leftarrow$  read_stack_frame()
6)     node  $\leftarrow$  CallTree.insert(Func, StackMap[Func], parent)
7)     if Func  $\in$  DangerFuncs then
```

```

8)       (addr, len)  $\leftarrow$  get_mem_write_info(instr)
9)       while node do
10)        (top, bot)  $\leftarrow$  StackMap[node.func]
11)        if addr  $\in$  (bot, top) and addr + len > top then
12)          report_overflow(Func, CallTree.backtrace(node))
13)          break
14)        end if
15)        node  $\leftarrow$  node.parent
16)      end while
17)    end if
18)  end if
19) end for
```

最终, 状态监控模块会将捕获的所有异常事件 (包括系统级崩溃、非法内存访问及栈溢出行为) 统一封装为状态反馈信息, 并实时传递给前端组件。前端据此调整输入生成策略, 实施针对性的变异与路径探索。

2.1.3 快速快照模块

快速快照模块旨在每轮模糊测试结束后, 将系统高效恢复至初始状态, 消除残留状态对后续测试的影响。现有 QEMU 冷迁移机制虽能保证状态一致性, 但恢复阶段平均耗时约 0.4 s, 难以满足高频测试需求。为此, 本文设计了轻量级快照恢复机制, 可实现每秒几十次状态恢复, 从而提升测试吞吐量与漏洞触发概率。核心优化包含以下 3 个方面。

1) CPU 状态恢复: 在系统模拟中, QEMU 会为不同处理器架构维护一组记录当前运行状态的数据结构。为了向不同架构提供统一的接口, QEMU 对这些结构体进行了统一封装, 并构造了一个包含 CPU 运行信息的统一结构。VxFuzzer 保存快照时, 会将该结构的全部数据保存到预先分配的内存区域; 恢复时, 只需将保存的数据直接写回原位置, 即可恢复 CPU 的运行状态。

2) 内存状态恢复: 为加速虚拟地址到物理地址的转换, 现代 CPU 通常实现了 TLB (translation lookaside buffer) 机制。QEMU 在模拟过程

中也引入了类似机制。当程序访问内存时，系统首先查询当前地址是否命中 TLB。若未命中，则执行完整的地址转换过程。执行到快照点后，VxFuzzer 开始监控内存页的写入操作。当 TLB 未命中且对页面执行写入操作时，系统会记录该页面的全部数据，并将该页面的物理地址作为哈希表的 Key 存储。快照恢复时，将哈希表中存储的数据回写至对应的内存区域，避免了全量内存复制的性能损耗。

3) 设备状态恢复：鉴于设备读写状态关乎系统的稳定性与连续性，VxFuzzer 采用针对设备写操作的动态 Hook 技术。QEMU 在创建设备时会为各设备注册对应的读写函数，通过 QEMU 提供的监控机制获取系统中所有加载的设备，重写设备的写操作回调函数。以网络设备 e1000 为例，在快照点之后执行 Hook 逻辑。在后续执行时，Hook 函数优先执行对应的读操作函数，读取当前需要写入地址的原始数据，然后将写操作函数地址与函数参数存储到结构体中用于后续恢复。完成记录后，执行原始的写操作函数。

在快照恢复阶段，系统仅需遍历记录的设备写操作序列，依次调用对应的写操作函数并还原其状态，从而重建快照时刻的设备内部状态。

2.2 前端组件

前端组件的主要目标是在接收后端反馈状态的基础上，生成更加有效的测试用例，并记录触发程序崩溃的输入样本，以供后续漏洞分析与验证。

现有模糊测试的用例生成策略主要分为基于变异的策略与基于生成的策略两类。为兼顾不同输入类型与测试需求，VxFuzzer 集成了这两类策略中具有代表性的工具 AFL 与 Boofuzz，从而在测试时可以灵活选择不同的策略。

2.2.1 AFL 前端

AFL 采用基于路径覆盖率的变异策略。在模糊测试过程中，AFL 持续跟踪每个输入样本的执行路径，并记录代码覆盖率信息，优先选择能够触发新路径的样本进行变异。

AFL 的黑盒模糊测试基于 QEMU 用户态仿真实现，适用于标准 Linux 用户态程序。然而，

由于 VxWorks 可执行文件与 Linux 可执行文件在文件格式、加载方式及运行环境上存在显著差异，AFL 无法直接用于 VxWorks 应用程序的模糊测试。

为使 AFL 适配 VxWorks 系统，VxFuzzer 从以下 3 个层面进行了优化。

1) 自定义 ForkServer：在 QEMU 内部实现自定义的线程化 ForkServer，通过独立的通信线程与 AFL 前端交互状态信息，避免阻塞 QEMU 主线程。

2) 快速快照代替进程复制：采用轻量级的快照恢复机制替代传统的进程复制（Fork）机制，不仅减少了状态恢复开销，还可精准控制快照创建时机。

3) 状态反馈集成：利用状态监控模块收集程序执行路径、代码覆盖率及异常信息，将这些状态数据实时反馈至 AFL 前端，为变异策略提供动态优化依据。

2.2.2 Boofuzz 前端

Boofuzz 依赖协议规范或数据格式的先验知识，通过构造符合目标程序输入格式的测试用例，以更高的语义准确性触发程序潜在的漏洞。Boofuzz 允许用户根据具体的协议规范自定义变异结构，并在测试过程中进行系统化变异。

为实现 Boofuzz 与 VxWorks 的适配，VxFuzzer 在以下两个方面进行了扩展。

1) 通过扩展 Boofuzz 的字段定义接口，用户仅需指定目标协议格式及约束规则，系统即可自动生成结构合规的测试用例，并交由后端执行引擎进行动态测试。

2) 引入轻量级状态监控模块，通过进程间通信（管道）将目标系统的运行状态实时反馈给 Boofuzz，弥补其原生架构的嵌入式环境状态感知能力的不足。

VxFuzzer 通过前端组件与后端组件的协同工作，实现了对 VxWorks 系统的自动化模糊测试，显著提升了漏洞挖掘效率。

3 实验验证

本节通过实验评估 VxFuzzer 在漏洞检测与挖掘方面的功能与优势，主要回答 3 个问题：① VxFuzzer 的功能性验证；② VxFuzzer 在异常监控中

的优势;③VxFuzzer在漏洞挖掘方面的优势。

当前,工业控制系统仍广泛采用VxWorks7.0以下版本,其中VxWorks5.5、VxWorks6.7和VxWorks6.9等版本占据主导地位。为全面评估VxFuzzer在真实工业环境中的适用性,本文选取上述3个具有代表性的版本进行测试。

为验证VxFuzzer的功能性,本文动态统计了内核启动时间、启动过程中内核执行的基本块数量与函数数量、快照恢复时间等数据,具体数据如表1所示。

实验数据表明,从VxWorks 5.5到6.9版本,内核启动时间略有增加,但是剔除VxWorks引导的10 s,系统均在10 s内启动,处于可接受的范围。与此同时,VxWorks启动过程中执行的基本块与函数数量显著增加,验证了VxFuzzer在大规模系统测试场景下的稳定性。此外,快照恢复时间始终保持在0.04 s内,体现了所设快照机制的高效性。

为验证VxFuzzer的扩展性,本文以应用层协议Modbus为例,分别测试了VxFuzzer在结合AFL前端和Boofuzz前端时的覆盖率(黑盒场景下“覆盖率”实际上是指程序运行时经过的控制流转移(边)数量)变化情况。图2展示了系统Modbus服务的覆盖率变化趋势,其中Boofuzz的覆盖率始终高于AFL的覆盖率。在协议测试中,Boofuzz针对字段进行变异的效果往往优于AFL针对路径覆盖率进行变异的效果,实验结果符合预期。

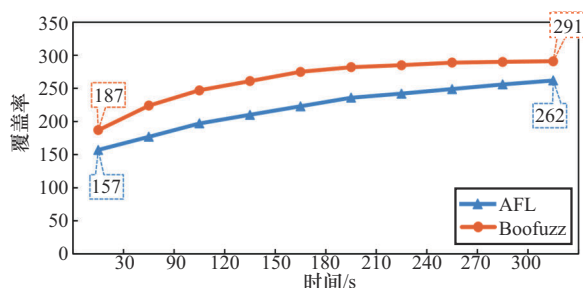


图2 AFL与Boofuzz覆盖率对比

Figure 2 Comparison of coverage between AFL and Boofuzz

基于CWE漏洞分类标准,本文手工构建了包含7类典型异常的测试集(单个用例规模控制在50行以内),以系统评估VxFuzzer异常监控模块的检测效能。实验结果如表2所示。

表2 VxFuzzer错误类型检测结果

Table 2 VxFuzzer error type detection results

类别	异常/错误类型	测试用例数	成功检测数	正确率
异常类型	局部变量溢出	60	49	81.77%
	函数无限递归	30	30	100%
	整数溢出	47	6	12.8%
	除以零	20	20	100%
错误类型	空指针解引用	40	40	100%
	格式化字符串漏洞	38	34	89.5%
	命令执行漏洞	35	19	54.3%

实验数据表明,VxFuzzer在检测函数无限递归、除以零异常和空指针解引用等内存错误时表现优异,检测准确率达到100%。在局部变量溢出和格式化字符串漏洞的检测方面,VxFuzzer同样展现出卓越的性能。然而,VxFuzzer对命令执行漏洞和整数溢出两类异常的检测效果欠佳。经后续分析发现,造成这一现象的主要原因包括:首先,整数溢出通常不会直接引发内存错误,因此运行时不易被系统捕获;其次,针对命令执行类漏洞,实际环境中往往存在命令过滤等防护机制,而VxFuzzer目前尚未集成相关的检测功能。

为验证VxFuzzer在真实环境下的漏洞检测能力,本文选取了具有代表性的CVE-2015-7599,以及URGENT/11系列漏洞中的CVE-2019-12255与CVE-2019-12260作为测试对象。在持续1 h的测试中,VxFuzzer成功触发并准确识别出上

表1 VxFuzzer功能测试结果

Table 1 Test results of VxFuzzer function

VxWorks 系统版本	内核启动时间/s	执行的基本块数量	执行的函数数量	快照恢复时间/s	边覆盖数量
5.5	11.945	3.9×10^6	27.5×10^4	0.037 2	6.9×10^3
6.7	15.646	11.3×10^6	131.6×10^4	0.036 6	17.1×10^3
6.9	18.984	299.7×10^6	56.2×10^4	0.039 3	16.8×10^3

述 3 个漏洞。表 3 详细记录了测试期间 VxFuzzer 执行的测试用例数量、触发的异常次数及是否真实漏洞等数据。

表 3 VxFuzzer 真实漏洞检测结果
Table 3 The real vulnerability detection results of VxFuzzer

CVE	CVSS	执行次数	触发异常次数	是否真实漏洞
CVE-2015-7599	8.1	11 877	288	是
CVE-2019-12255	9.8	1 226	56	是
CVE-2019-12260	9.8	2 164	87	是
CNVD-2025-9289	4.9	3 686	436	是

为客观评估 VxFuzzer 的检测性能, 本文设计了对比实验, 将其与现有工具 vxAFL 进行系统比较。需要说明的是, 虽然 vxAFL 在文献中提出了基于 QEMU 的 VxWorks 模糊测试方法, 但其原始实现未公开, 且文献中对测试环境与输入点的描述不够完整。这使得在统一实验环境下对 vxAFL 与 VxFuzzer 进行直接对比几乎不可行。为尽可能减少环境差异, 本文选取 vxAFL 中提及的 CVE-2015-7599 作为测试目标, 并参考其文中描述构造了等效实验环境。

实验结果显示, VxFuzzer 在持续约 1 h 的测试中共触发 288 次异常 (见表 3), 而 vxAFL 在 15 min 内仅检测出 3 次异常; 在路径覆盖方面, VxFuzzer 触发了 6 285 条内核函数执行路径 (内核协议栈基本块边数), 显著优于 vxAFL 的 200 余条。具体数据详见表 3 与图 3。这些对比数据充分证明, VxFuzzer 在异常触发和代码路径覆盖率方面均具有明显优势。

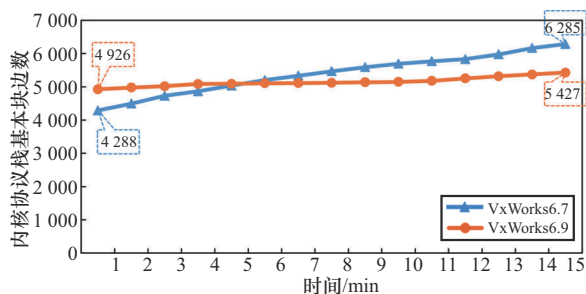


图 3 路径覆盖检测结果

Figure 3 Path coverage detection results

为进一步验证 VxFuzzer 在真实环境中的漏洞挖掘能力, 本文选取 VxWorks 内部应用程序 libmodbus 进行测试, 并结合 Boofuzz 前端生成相应测试用例。在持续约 1 h 的模糊测试中, VxFuzzer 共检测出 436 个异常 (见表 3)。由于触发异常的测试用例均由 Boofuzz 变异生成, 理论上样本不会重复, 但分析结果显示这些异常样本中存在重复路径。最终, 从样本中分析出一个未公开的 0-Day 漏洞 (CNVD-2025-9289)。

4 结束语

本文设计并实现了一个基于 QEMU 全系统仿真的模糊测试框架 VxFuzzer。该框架采用前后端分离的设计, 集成了基本块插桩、异常监控与轻量级快照恢复等关键机制。实验结果表明, VxFuzzer 能够高效地运行在多个发行的 VxWorks 版本上, 成功复现了多个已公开漏洞, 并在实际测试中新发现一个 0-Day 漏洞。

尽管 VxFuzzer 在 VxWorks 系统漏洞挖掘方面展现出良好的实用性, 但仍存在若干不足之处。首先, 在异常检测维度上, VxFuzzer 主要聚焦于栈溢出与程序崩溃异常的检测, 在堆内存相关漏洞检测方面存在不足。其次, 尽管已集成 AFL 和 Boofuzz, 但与其他主流模糊测试工具 (如 LibFuzzer、honggfuzz 等) 的兼容性不足。最后, 从自动化程度来看, VxFuzzer 仍依赖人工分析确定系统快照创建点与状态恢复入口, 影响了全流程自动化能力。

未来的工作将围绕以下几个方向展开: ①分析 VxWorks 堆内存的相关漏洞, 并添加到 VxFuzzer 异常检测模块中; ②进一步拓展对其他模糊测试工具的支持, 构建更统一的前端接口; ③优化快照恢复机制的入口点, 实现智能化入口点分析。

参考文献:

- [1] 张朝. 多核嵌入式实时操作系统(RTOS)综述[J]. 电脑知识与技术: 学术版, 2015(4X):3.
ZHANG Z. A survey on multi-core embedded real-time operating systems (RTOS) [J]. Computer Knowledge and Technology: Academic Edition, 2015(4X):3.
- [2] GODEFROID P. From blackbox fuzzing to whitebox fuzzing to-

- wards verification[C]//The 2010 International Symposium on Software Testing and Analysis. 2010.
- [3] MCNALLY R, YIU K, GROVE D, et al. Fuzzing: the state of the art[R]. Edinburgh: Defence Science and Technology Organisation, 2012.
- [4] 刘航天, 甘水滔, 张超, 等. 物联网设备固件自动化漏洞挖掘技术研究综述[J]. 网络与信息安全学报, 2025, 11(2): 26-49.
- LIU H T, GAN S T, ZHANG C, et al. Survey on automated vulnerability mining techniques for IoT device firmware[J]. Chinese Journal of Network and Information Security, 2025, 11(2): 26-49.
- [5] PAYER M. The fuzzing hype-train: how random testing triggers thousands of crashes[J]. IEEE Security & Privacy, 2019, 17(1): 78-82.
- [6] LIANG H, PEI X, JIA X, et al. Fuzzing: state of the art[J]. IEEE Transactions on Reliability, 2018, 67(3): 1199-1218.
- [7] JOHANSSON W, SVENSSON M, LARSON U E, et al. T-Fuzz: model-based fuzzing for robustness testing of telecommunication protocols[C]//Proceedings of 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation (ICST). Virginia: the Internet Society, 2014:323-332.
- [8] ZHANG H, ZHANG Z, TANG W. Improve peach: making network protocol fuzz testing more precisely[J]. Applied Mechanics & Materials, 2014, 551: 642-647.
- [9] JUN L, ZHAO B D, ZHANG C. Fuzzing: a survey[J]. Cybersecurity, 2018, 1(1): 1-13.
- [10] MANES V J M, HAN H, HAN C, et al. The Art, science, and engineering of fuzzing: a survey[J]. IEEE Transactions on Software Engineering, 2019, 47(11): 2312-2331.
- [11] BELLARD F. QEMU: a fast and portable dynamic translator[C]//Proceedings of the USENIX Annual Technical Conference. Vancouver: USENIX Association, 2005: 41-46.
- [12] YUN J, RUSTAMOV F, KIM J, et al. Fuzzing of embedded systems: a survey[J]. ACM Computing Surveys, 2023, 55(7): 1-33.
- [13] ZHENG W, ZHOU Y, WANG B. Design and implementation of VxWorks system vulnerability mining framework based on dynamic symbol execution[C]//Proceedings of the 9th International Conference on Computer Engineering and Networks. Singapore: Springer, 2021: 801-811.
- [14] CHEN L B, CAI Q P, MA Z B, et al. SFuzz: slice-based fuzzing for real-time operating systems[C]//Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2022: 485-498.
- [15] DANG H V, NGUYEN A Q. Unicorn: next generation CPU emulator framework[R]. BlackHat. 2015.
- [16] 邵帅. 基于模糊测试的工控PLC系统漏洞挖掘技术研究[D]. 杭州: 杭州电子科技大学, 2020.
- SHAO S. Research on vulnerability mining technology of industrial control PLC system based on fuzzing[D]. Hangzhou: Hangzhou Dianzi University, 2020.
- [17] SZEKERES L, PAYER M, WEI T, et al. SoK: eternal war in memory[C]//Proceedings of the 2013 IEEE Symposium on Security and Privacy. Piscataway: IEEE Press, 2013: 48-62.
- [18] PROBST M. Dynamic binary translation[C]//The UKUUG Linux Developers' Conference. Bristol, United Kingdom, 2002.
- [19] 张锦, 单泽虎, 刘晓东, 等. 动态二进制翻译技术综述[J]. 计算机科学与探索, 2024, 18(10): 2521-2550.
- ZHANG J, SHANE Z H, LIU X D, et al. Comprehensive review of research on dynamic binary translation techniques[J]. Journal of Frontiers of Computer Science and Technology, 2024, 18(10): 2521-2550.
- [20] 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 Security Symposium (USENIX Security 19). Berkeley: USENIX Association, 2019: 1099-1114.
- [21] HENDERSON A, YAN L K, HU X C, et al. DECAF: a platform-neutral whole-system dynamic binary analysis platform[J]. IEEE Transactions on Software Engineering, 2017, 43(2): 164-184.
- [22] FIORALDI A, MAIER D, EIBFELDT H, et al. AFL++: combining incremental steps of fuzzing research[C]//Proceedings of the 14th USENIX Workshop on Offensive Technologies (WOOT'20). Berkeley: USENIX Association, 2020.
- [23] FIORALDI A, D'ELIA D C, QUERZONI L. Fuzzing binaries for memory safety errors with QASan[C]//Proceedings of the 2020 IEEE Secure Development (SecDev). Piscataway: IEEE Press, 2020: 23-30.
- [24] MARINI M, D'ELIA D C, PAYER M, et al. QMSan: efficiently detecting uninitialized memory errors during fuzzing[C]//Proceedings 2025 Network and Distributed System Security Symposium. Virginia: Internet Society, 2025.
- [25] ZADDACH J, BRUNO L, FRANCILLON A, et al. Avatar: a framework to support dynamic security analysis of embedded systems' firmwares[C]//Proceedings 2014 Network and Distributed System Security Symposium. Internet Society, 2014.
- [26] MUENCH M, NISI D, FRANCILLON A, et al. Avatar²: a multi-target orchestration platform[C]//Proceedings 2018 Workshop on Binary Analysis Research. Virginia: the Internet Society, 2018.
- [27] CLEMENTS A A, GUSTAFSON E, SCHARNOWSKI T, et al. HALucinator: firmware re-hosting through abstraction layer emulation[J]. 2020.

[作者简介]



李松涛 (2001-), 男, 河南南阳人, 郑州大学硕士生, 主要研究方向为物联网安全。



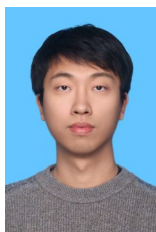
耿洋洋 (1993-), 男, 河南太康人, 信息工程大学讲师, 主要研究方向为工业控制系统安全、信息物理系统安全、漏洞挖掘。



麻荣宽（1992-），男，山东菏泽人，信息工程大学讲师，主要研究方向为漏洞挖掘、网络攻防与系统安全。



彭建山（1979-），男，江西吉安人，信息工程大学副教授，主要研究方向为网络空间安全。



贾岩（1992-），男，河北保定人，南开大学副教授，主要研究方向为漏洞挖掘、物联网安全、网络攻防与系统安全。



曹琰（1983-），男，河南郑州人，郑州大学副教授，主要研究方向为网络安全。