计算机科学
COMPUTER SCIENCE

嵌入式软件模糊测试研究综述

孙琪明, 侯刚, 靳文杰, 黄晨, 孔维强

引用本文

孙琪明, 侯刚, 靳文杰, 黄晨, 孔维强. 嵌入式软件模糊测试研究综述[J]. 计算机科学, 2025, 52(7): 13-25.

SUN Qiming, HOU Gang, JIN Wenjie, HUANG Chen, KONG Weiqiang. Survey on Fuzzing of Embedded Software [J]. Computer Science, 2025, 52(7): 13-25.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

基于局部加密网格的LBM程序负载均衡方法研究

Investigation on Load Balancing Strategies for Lattice Boltzmann Method with Local Grid Refinement

计算机科学, 2025, 52(5): 101-108. https://doi.org/10.11896/jsjkx.241100169

Python虚拟机本地代码的安全性实证研究

Empirical Security Study of Native Code in Python Virtual Machines

计算机科学, 2022, 49(6A): 474-479. https://doi.org/10.11896/jsjkx.210600200

以太坊Solidity智能合约漏洞检测方法综述

Overview of Vulnerability Detection Methods for Ethereum Solidity Smart Contracts

计算机科学, 2022, 49(3): 52-61. https://doi.org/10.11896/jsjkx.210700004

一种结合AADL与Z的嵌入式软件可靠性建模与评估方法

Embedded Software Reliability Model and Evaluation Method Combining AADL and Z

计算机科学, 2019, 46(8): 217-223. https://doi.org/10.11896/j.issn.1002-137X.2019.08.036

构件化嵌入式软件设计模型非功能性质验证的工具实现

Tool Implementation of Non-functional Verification for Component-based Embedded Software Designs

计算机科学, 2010, 37(8): 156-163.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240800068

嵌入式软件模糊测试研究综述

孙琪明[1]　侯　刚[1]　靳文杰[1]　黄　晨[2]　孔维强[1]
1 大连理工大学软件学院　辽宁 大连 116620
2 北京控制工程研究所　北京 100190
(lzdz@mail.dlut.edu.cn)

摘　要　嵌入式软件目前已广泛应用于国防军工、航空航天、物联网通信等各类安全攸关系统,面临着日益严峻的安全挑战,因此,如何快速发现并修复嵌入式软件安全漏洞变得至关重要。模糊测试作为一种高效的软件测试技术,能够通过自动生成大量随机数据测试软件系统的可靠性,目前已逐步应用于各类嵌入式软件的漏洞发现。文中首先介绍了模糊测试技术、嵌入式系统及其固件设备;其次,概述了嵌入式软件模糊测试的流程,分析了其与传统软件模糊测试的区别和面临的挑战;然后,系统地介绍了嵌入式软件模糊测试的研究现状和主要方法,包括直接模糊测试和基于仿真的模糊测试;最后,分析了嵌入式软件模糊测试效果提升可采用的优化方法并展望了未来可能的技术方向。

关键词:嵌入式软件;固件设备;安全漏洞;直接模糊测试;基于仿真的模糊测试
中图分类号　TP309

Survey on Fuzzing of Embedded Software

SUN Qiming[1],HOU Gang[1],JIN Wenjie[1],HUANG Chen[2] and KONG Weiqiang[1]
1 College of Software,Dalian University of Technology,Dalian,Liaoning 116620,China
2 Beijing Institute of Control Engineering,Beijing 100190,China

Abstract　Embedded software is now widely used in various safety-critical systems,such as national defense,aerospace,and IoT communications,which face increasingly severe security challenges. Therefore,it is crucial to quickly identify and fix security vulnerabilities in embedded software. Fuzz testing,as an efficient software testing technique,can automatically generate a large amount of random data to test the reliability of software systems and has gradually been applied to the discovery of vulnerabilities in various embedded software. This paper first introduces the concepts of fuzz testing,embedded systems,and their firmware devices. Then,it provides an overview of the fuzz testing process for embedded software,analyzes the differences from traditional software fuzz testing and the faced challenges. Following that,it systematically introduces the current research status and main methods of fuzz testing for embedded software,including direct fuzz testing and simulation-based fuzz testing. Finally,this paper discusses optimization methods that can be used to improve the effectiveness of embedded software fuzz testing and looks ahead to potential future technological directions.

Keywords　Embedded software,Firmware devices,Security vulnerability,Direct fuzz testing,Simulation-based fuzz testing

1　引言

随着微控制单元技术和移动互联技术的迅猛发展,嵌入式系统得到了广泛的应用。从航空航天到武器装备,从工业控制设备到消费电子产品,嵌入式系统几乎涵盖了人们生产生活的各个领域。预计到2030年,全球嵌入式设备的数量将从2020年的151亿台增加到290亿台,几乎翻一番。到2030年,中国的嵌入式设备数量将达到80亿台。

然而,随着嵌入式设备的广泛使用,嵌入式软件漏洞带来的安全威胁也在加剧。例如,2010年"震网"病毒成功攻击了伊朗核电站中的工业控制嵌入式软件,直接导致了离心机等关键设备的损毁,对伊朗的国家安全产生了严重影响[1];2016年Mirai恶意软件感染了数百万个IoT(Internet of Things,物联网)设备,并利用这些设备发起了大规模的网络攻击,导致全球范围内数以万计的网络服务器陷入拒绝服务状态[2]。同时,根据公开数据,2021年上半年已经检测到针对50亿嵌入式设备的超过15亿次的网络攻击。面对日益严重的安全威胁,如何高效发现并修复嵌入式软件漏洞已成为工业界和学术界关注的难点问题。

模糊测试是一种自动化测试方法,其原理是使用随机或

到稿日期:2024-08-12　返修日期:2024-11-11
基金项目:高可信嵌入式软件工程技术实验室开放基金课题(LHCESET202306)
This work was supported by the Lab of High Confidence Embedded Software Engineering Technology Open Fund Project(LHCESET202306).
通信作者:侯刚(hg.dut@163.com)

半随机数据作为软件输入来生成大量的测试用例，以便发现软件中可能存在的漏洞或异常行为。虽然模糊测试被普遍认为是一种高效的漏洞发现技术，但将其直接应用到嵌入式软件上却面临着一系列挑战。首先是多架构支持，因为嵌入式系统通常包含多种处理器架构，需要确保模糊测试工具能够兼容并有效地运行在这些不同的架构上。其次，崩溃检测也是一个挑战。由于嵌入式设备往往不具备完整的操作系统支持，因此难以检测和处理崩溃。此外，嵌入式系统通常具有资源有限的特点，如有限的存储空间和处理能力，因此需要考虑如何有效地利用有限资源进行模糊测试，以确保测试效率和覆盖范围。针对上述挑战，相关学者开展了大量研究，有效提升了模糊测试在嵌入式软件漏洞发现方面的实用性。

本文旨在综述嵌入式软件模糊测试技术的现状和发展趋势。首先，介绍了模糊测试技术、嵌入式系统及其固件仿真技术；随后，概述了嵌入式软件模糊测试的典型流程，详细比较了其与传统软件模糊测试的主要区别和面临的挑战；然后系统地介绍了当前嵌入式软件模糊测试的主要研究方法和成果，包括直接模糊测试、基于仿真的模糊测试；最后，对嵌入式软件模糊测试技术的发展进行展望，探讨了未来可能的技术方向，旨在为该领域的研究者和实践者提供参考和启发。

2 背景知识

本章首先介绍模糊测试的基础知识，包括测试流程、测试分类以及相关技术；再介绍嵌入式系统的概念及其分类；最后介绍固件仿真技术，并对现有的仿真器功能进行分析。

2.1 模糊测试技术

模糊测试是一种软件测试方法，其核心思想是在程序中自动或半自动地生成随机数据作为输入，并监控程序的异常行为，如崩溃、断言失败等，从而发现潜在的程序错误，如内存泄漏、越界访问等。

2.1.1 模糊测试介绍

模糊测试的概念最早可追溯到 1988 年，Miller 发表了他的开创性论文[3]，并进行了第一批模糊测试实验，通过向 UNIX 系统的命令行工具输入随机数据，发现了多个漏洞。此后，模糊测试开始获得安全研究人员和软件开发者的关注，并逐渐应用在基本的黑盒测试上，重点识别容易崩溃的软件错误[4]。2000 年初，更多学者开始探索不同类型的模糊测试技术，提出了白盒模糊测试[5]。2007 年，微软开始使用模糊测试作为其安全开发周期的一部分，大大扩展了模糊测试在工业界的应用和认知。有别于黑盒模糊测试，白盒模糊测试要求能够充分了解软件内部信息，以获得更好、更精准的测试输入数据，从而实现更高的代码覆盖率和更好的测试效果。但是在充分获得软件内部信息的过程中，会消耗大量的时间和资源，在一定程度上降低了白盒模糊测试的效率。因此，出现了使用较少内部信息进行测试的灰盒模糊测试方法，并在 2012 年灰盒模糊测试工具 AFL(American Fuzzy Lop，美国模糊兔子)发布[6]，该工具通过插桩，利用多种变异策略，使得模糊测试更加高效和智能。

2.1.2 模糊测试的流程

模糊测试基本的工作流程可以分为 5 步：预处理、输入构造、发送输入、目标监视、结果分析，其中主要的测试环节是输入构造、发送输入和目标监视环节。

模糊测试过程如图 1 所示。在预处理阶段搜集目标相关信息并制定模糊测试策略后，首先进入输入构造环节。此环节中，需获取一定数量的初始种子，确定种子的能量分配策略和变异策略，以此生成大量的种子数据。得到输入种子数据后，在发送输入环节中，经过合适的筛选后将有效的种子数据输入，以提高模糊测试的覆盖率。在目标监视环节，主要利用合理的崩溃检测方法来监测捕获模糊测试中发生的崩溃，通过对崩溃的筛选发现真实的软件漏洞。

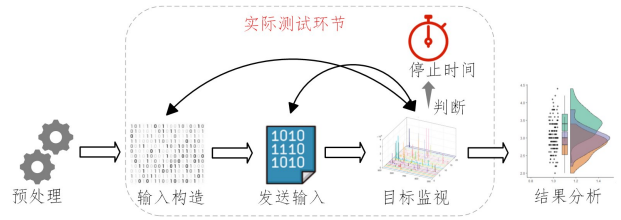


图 1 模糊测试流程

Fig. 1 Fuzzing process

2.1.3 模糊测试的分类

模糊测试根据在测试期间需要的被测程序的信息量，可以划分为黑盒、白盒和灰盒模糊测试 3 个类别[7]。

1)黑盒模糊测试。黑盒测试也被称为输入输出驱动测试或数据驱动测试。在软件工程中的黑盒测试通常只确定程序的接口，而不在意目标程序内部细节[8]。因此，黑盒模糊器基于预先设定好的策略随机变异种子测试用例，而不识别被测程序内部信息。黑盒模糊测试工具具有设计简单、方便使用的优点，开发速度快。但其缺少对内部信息的分析，容易造成大量的无效输入，导致覆盖率偏低。

2)白盒模糊测试。白盒模糊测试会获得充足的被测程序内部信息，是一种基于程序内部结构和信息来生成测试用例的技术，具体来讲，它通过利用程序分析技术，如动态符号执行技术、静态分析技术等，生成合适的测试用例。白盒模糊测试的优点是能够生成高质量的测试输入数据，相较于黑盒模糊测试拥有更高的代码覆盖率和更深层的漏洞检测能力。但在实际应用中，由于白盒模糊测试对被测程序的信息量要求大，对内部信息的分析程度要求很高，因此往往需要消耗大量的资源，严重影响了模糊测试的效率。

3)灰盒模糊测试。灰盒模糊测试是介于黑盒模糊测试和白盒模糊测试之间的一种模糊测试方法。它通常只需要目标程序的部分信息，这部分信息通常是通过插桩产生的代码覆盖率或通过污点分析产生的污点流信息。灰盒模糊测试实际上也是白盒模糊测试的一种变体，它只能获得部分程序内部信息用于模糊测试。例如，灰盒模糊测试中最流行的模糊器 AFL 通过编译时插桩，运行时收集路径覆盖信息，并有效利用覆盖信息来变异测试用例并进行模糊测试，极大改善了模糊测试效果。在 AFL 之后，更先进的灰盒模糊测试工具[9-11]层出不穷，体现了灰盒模糊测试的实用价值。

2.1.4 模糊测试的相关技术

1）插桩技术[12]：插桩技术通过在目标代码中插入特定的代码片段，旨在获取程序的静态或动态执行信息，例如程序的抽象语法树、覆盖率以及函数内部变量的取值等。

常见的插桩方法包括静态插桩和动态插桩两类。静态插桩通常在源代码或中间代码的编译阶段进行，其优势在于速度较快且节省时间，但对程序源代码有一定依赖性。相比之下，动态插桩是在程序运行时对执行代码进行插桩。例如，利用 QEMU[13]（Quick Emulator，快速仿真器）等模拟技术实现的动态插桩，可以实时获取程序的运行信息。插桩技术已被广泛应用于模糊测试领域，例如 AFL 工具中包含了静态插桩和动态插桩两种模式，从而使 AFL 能够对源代码和二进制代码进行模糊测试。

2）符号执行技术[14]。符号执行是一种程序分析方法，它通过使用符号值代替实际数值来模拟程序的执行过程。该方法通过分析程序将符号变量作为目标程序或函数参数来替换具体值，并仿真执行程序中的指令。每条指令的操作都基于符号变量进行，有助于提高模糊测试的覆盖率，同时能深入探查程序中的潜在漏洞。

符号执行可以分为静态符号执行和动态符号执行。静态符号执行在程序分析中遇到循环或递归时容易产生路径爆炸的问题，并且路径约束中涉及的操作（如取 Hash 值）可能导致约束求解失败。为了解决这些问题，动态符号执行通过将程序的实际运行与符号化执行相结合，来确保程序的实际状态与符号化状态保持同步。对于那些难以解决的约束，采用替换为实际数值的方式，从而缓解了静态符号执行中的部分难题。此外，动态符号执行还通过深度优先搜索策略更深入地探查目标程序。

然而，动态符号执行也存在诸多挑战。首先，由于程序的复杂性，路径爆炸的问题仍然存在，且随着程序的复杂程度增加，这一问题会变得更加严重，解决方法是通过启发式方法选择更重要的路径进行探查。此外，虽然动态符号执行通过实际值替换部分静态符号执行的约束，但可能会导致部分路径丢失，进而影响探查结果的完整性。

3）污点分析技术[15]。污点分析旨在监测程序中受到了预设污染源（如输入的污染）影响的数据，目的是追踪污染源与汇聚点（如包含敏感信息的数据）之间的信息流动。具体来说，在模糊测试中，使用该技术将输入数据标记为污点数据，随后跟踪这些数据的流向，通过检测污点信息是否被用于汇聚点上的操作来识别程序中的漏洞问题。

污点分析分为静态污点分析和动态污点分析两种方法。静态污点分析不依赖程序的实际执行，通过分析代码结构生成程序的控制流图，并根据数据流和依赖关系进行分析。动态污点分析则是在程序运行时，通过监控动态执行来进行分析。相比之下，动态污点分析的准确性更高，但需要更高的程序覆盖率，并消耗更多的资源。静态污点分析类似于符号执行，可能会面临路径爆炸的问题，而简化后的静态污点分析可能会导致严重的污染问题。

2.2 嵌入式系统及固件

2.2.1 嵌入式系统

嵌入式系统是一种专用的计算机系统，它是为了完成控制机器或设备等特定功能而设计的。这类系统通常嵌入到它们所控制的设备内部，与其合并为一个整体。与传统的通用计算机相比，嵌入式系统通常在硬件资源、电源消耗、尺寸和成本方面有严格的限制。

嵌入式系统的核心部分通常包括一个或多个微控制单元（或微控制器）以及存储器，这些处理器直接控制硬件设备，处理数据并执行程序。而用于操作嵌入式系统的软件存储在只读存储器或闪存芯片中，这些固化在只读存储器或闪存芯片中的软件被称为固件。固件通常由引导加载程序、操作系统和文件系统组成，功能包括初始化硬件设备、提供操作系统的底层支持以及执行硬件级别的控制和监控任务。因此，固件通常是嵌入式系统的典型攻击目标。

2.2.2 嵌入式设备的分类

嵌入式设备可以根据多种标准进行分类，由于本文关注模糊测试，因此从软件测试角度将嵌入式系统分为 3 类，如表 1 所列。

表 1 不同类型的嵌入式设备及其比较

Table 1 Various types of embedded devices and their comparisons

类型	功能	内存管理	安全机制	文件系统
第一类	丰富	健全	有	有
第二类	较简单	不健全	无	有
第三类	简单	不健全	无	无

第一类是基于通用操作系统的嵌入式设备。由于通用操作系统具有大量功能性、兼容性和开发人员的持续支持，因此它们也被用于嵌入式设备。但由于性能限制，通用操作系统往往以最小形态用于嵌入式设备，如实时 Linux、嵌入式 Windows 和 RaspberryPi，这些操作系统针对嵌入式系统环境进行了修改，保留了许多桌面级功能，但采用了精简的组件，并与 busybox 或 uClibc 等轻量级用户空间环境相结合。

第二类是基于专用嵌入式操作系统的嵌入式设备。该类设备使用专门为嵌入式系统开发的操作系统，它们通常是商业产品和封闭源。例如实时操作系统，因其没有缓冲延迟，一般用于处理数据的实时应用程序。此外，还有一些操作系统适用于低功耗设备，例如 Clinux，ZephyrOS 和 VxWorks。上述系统通常是单一用途的控制系统，这些系统的一个重要特性是操作系统和应用程序可以独立编译，并且不是从桌面操作系统派生的。

第三类是不具备操作系统的嵌入式设备。该类设备通常没有操作系统，由应用程序直接对底层硬件进行控制。

2.3 固件仿真技术

嵌入式系统固件由于设备的计算资源有限和对实际硬件的高度依赖，导致安全分析面临诸多挑战。这些限制使得传统的软件分析技术难以直接应用于嵌入式固件的安全分析。固件仿真技术是一种解决方案，该技术将目标固件提取出来并在指定的虚拟环境中运行，目前主要的固件仿真器包括 Simics[16]，QEMU，Unicorn[17] 和 Qiling[18]。其中，Simics，QEMU 和 Qiling 能支持从单一应用程序（用户级仿真）到整

个固件系统(系统级仿真)的仿真。相比之下,Unicorn 专注于 CPU 指令级的仿真,适用于更具体和深入的分析需求。这些工具为嵌入式固件的安全性分析提供了帮助,极大地增强了分析的效率和效果,具体的分类及特点如表 2 所列。

表 2 嵌入式固件仿真器分类及特点

Table 2 Classification and characteristics of embedded firmware emulators

仿真器	仿真器分类	特点	应用场景
Qiling	用户级仿真	针对特定嵌入式固件程序的仿真,但有可能无法完全模拟固件与硬件交互导致错误	开发测试中的应用程序逻辑分析;嵌入式软件开发和单元测试
Simic			
QEMU	系统级仿真	模拟固件内核及其与所有外设交互的完整过程,但资源需求量高,执行速度慢;能提供全面的仿真环境	全方面嵌入式系统测试;嵌入式固件与硬件之间的兼容性测试
Unicorn	CPU 指令仿真	高度定制仿真环境;适合模拟特定 CPU 指令集或处理特殊的硬件行为	需要高度定制仿真环境的研究和开发,如模拟特定的 CPU 指令

3 嵌入式软件模糊测试概述

3.1 嵌入式软件模糊测试流程

嵌入式软件模糊测试总体上可分为 5 个步骤,分别是预处理、测试用例生成、目标程序执行、目标监控和异常分析,如图 2 所示。

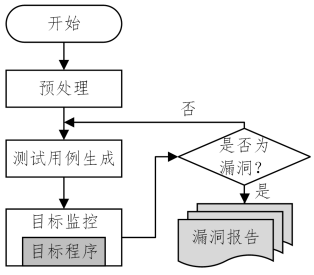


图 2 嵌入式软件的模糊测试流程

Fig. 2 Fuzzing process for embedded software

1)预处理阶段。测试人员首先确定目标系统和模糊测试策略。如果可以直接访问目标系统,则识别目标系统的接口;否则,测试人员将获取目标系统的固件,提取文件系统和内核信息,并在仿真环境中模拟固件。需要注意的是,获取固件可能涉及到对目标系统进行物理或逻辑访问。

2)测试用例生成。在这一阶段,测试人员根据给定的模糊测试策略以及目标系统的特征,采用各种技术和方法来生成多样化、具有挑战性的测试用例,尽可能地模拟真实世界中的各种输入情况,以发现目标系统可能存在的漏洞和异常行为。

3)目标程序执行。测试人员将生成的测试用例发送到目标系统,开始循环执行模糊测试,直到收到终止信号。

4)目标监控。在每一次迭代中,测试人员持续地监控目标系统的反馈,以确保系统在模糊测试过程中的稳定性和健壮性。通过监控系统的响应和行为,测试人员能够及时捕获异常情况,包括但不限于崩溃、错误输出和未预期的行为。

5)异常分析。一旦检测到异常,测试人员将提取测试相关的信息并对异常进行分析。如果检测到的异常是漏洞,则报告漏洞;否则,将根据分析的信息调整输入,继续进行模糊测试。

3.2 与传统模糊测试的区别

由于嵌入式软件的特殊性,嵌入式模糊测试与传统模糊测试存在一定差异,如表 3 所列。

表 3 传统的模糊测试和嵌入式模糊测试的差异

Table 3 Differences between traditional fuzzing and embedded fuzzing

对比项	传统的模糊测试	嵌入式模糊测试
硬件依赖	弱	强
崩溃检测	简单	困难
插桩	简单	困难
性能	较强	有限
可扩展性	强	弱

3.2.1 强硬件依赖

嵌入式系统通常依赖特定的硬件平台,因此嵌入式软件设计与硬件高度耦合。模糊测试必须在实际硬件或高度仿真的环境中进行,这意味着测试人员需要获取并配置适当的硬件设备,并且测试过程可能会受到硬件资源的限制。由于嵌入式系统硬件的依赖性,测试环境的搭建和管理也更为复杂。测试工具可能需要与硬件接口进行交互,这涉及到低级的硬件控制和信号处理。当模糊测试发现潜在漏洞或崩溃时,调试和故障分析也需要更深入的硬件层次知识,特别是在调试低级代码或硬件驱动程序时。

与传统通用模糊器相比,嵌入式模糊器必须支持多种架构。根据 Chen 等的研究[19],32 位大/小端 MIPS 架构占调查固件的 79.4%,而 32 位大/小端 ARM 占 10.0%。此外,还有如 PowerPC,Motorola 和 x86-64 等常见架构。由于 MIPS 和 ARM 架构占据 90%以上的固件市场,嵌入式模糊器至少需支持这两种架构,以有效覆盖大多数固件。

3.2.2 崩溃检测

崩溃检测是模糊测试中的关键步骤,在传统模糊测试中相对容易。桌面系统通常配备了多种崩溃或故障检测机制,如错误消息、安全警告和系统日志。在 Linux 系统中,当执行的程序由于致命信号(如分段错误)终止时,模糊器会检测到崩溃。这是因为内存错误导致返回地址被覆盖为任意值,从而引发分段错误或程序终止。这种方法简单高效,模糊器直接捕获信号,无需操作系统介入。

然而,嵌入式系统的情况更为复杂。大多数嵌入式操作系统缺乏类似的故障检测机制,模糊器难以直接检测到崩溃。此外,由于嵌入式系统通常缺乏输出设备,用户无法实时监测崩溃情况,因此嵌入式模糊器往往会使用活跃性检查机制,通过心跳消息、内存检查工具(如 MemorySanitizer[20],AddressSanitizer[21],ThreadSanitizer[22] 等)或调试端口(如 UART[23],

JTAG[24])来实现。这些技术有助于及时发现嵌入式设备中的问题。

3.2.3 插桩

在软件测试中，插桩是一种在软件代码中插入额外的指令或代码片段来收集运行时信息的方法。插桩方法分为两种类型：源代码插桩和二进制插桩。源代码插桩是在静态分析期间对源代码进行的，而二进制插桩是对编译后的二进制文件进行的[25]。

对于嵌入式模糊测试，通常来说，待测试的嵌入式设备固件或应用程序源代码是不可获得的，因此源代码插桩并不适用；此外，由于嵌入式设备具有各自独特的 CPU 架构、操作系统和 I/O 设备，重新编译程序几乎不可能。因此，嵌入式设备测试人员很难利用插桩。

3.2.4 性能

通常情况下，模糊测试需要重新执行被测试程序，以确保每个测试输入都处于干净的状态。由于可以通过还原虚拟机快照来迅速恢复程序的初始状态，这种技术在桌面系统中很容易实现。然而，对于嵌入式设备来说，重置设备需要耗费大量时间，因此这种方法相当困难。此外，在桌面系统中，可以通过并行执行模糊测试来提高效率。但在嵌入式设备中，并行化通常是不可行的，尤其是在没有嵌入式操作系统的设备中。因此，对于嵌入式系统来说，重复模糊测试可能需要花费相当长的时间。

3.2.5 可扩展性

处理增加需求的能力称为可扩展性，在模糊测试领域中体现为并行执行。尽管并行模糊测试通常可以在桌面系统中实现，但其在嵌入式系统中却不可行。这是因为在嵌入式系统中，建立并行模糊测试的环境需要大量实际设备，成本相当高昂。因此，不能期望嵌入式模糊测试具有可扩展性。

3.3 嵌入式软件模糊测试面临的挑战

首先，嵌入式系统涉及多种处理器架构，开发支持不同架构的模糊测试工具复杂且耗时。其次，崩溃检测在嵌入式系统中更为困难。桌面系统通常有完善的崩溃检测机制，如错误消息和系统日志，但嵌入式系统由于资源有限且缺乏标准化操作系统，崩溃往往难以及时或完全检测到。最后，由于测试工具需要一定的计算资源来确保有效性，因此嵌入式系统的资源限制（如处理能力、内存和存储空间）影响了模糊测试的高效实施。

4 嵌入式软件模糊测试现状

目前嵌入式软件模糊测试技术根据模糊测试工具和目标嵌入式系统之间的连接方式可分为：直接模糊测试和基于仿真的模糊测试。直接模糊测试是通过直接连接目标设备，在不需要干预的情况下测试系统，其包括通过网络进行的模糊测试和通过调试接口进行的模糊测试；基于仿真的模糊测试使用通过 UART 或 JTAG 从嵌入式设备中提取的或者从固件更新网站下载的固件，并通过仿真进行模糊测试，其包括部分仿真方法和全仿真方法。

4.1 直接模糊测试

直接模糊测试通过向目标软件输入非预期、随机或异常数据，以测试其对于意外输入的处理能力。该方法生成具有挑战性的输入，并注入到目标软件中，以触发潜在的漏洞和异常。通过监测崩溃、异常行为和资源耗尽等现象，并结合代码覆盖率分析，发现如缓冲区溢出、格式字符串等安全漏洞。为了获得最佳结果，直接模糊测试常常与其他测试技术和方法相结合，并进一步进行漏洞分析和利用，以验证漏洞的存在与严重性。目前，主要有通过网络进行的直接模糊测试和通过调试接口进行的直接模糊测试这两种方法，具体方法如表 4 所列。

表 4 直接模糊测试的相关方法

Table 4 Related methods of direct fuzzing

工具	年份	分类	测试对象	方法特性
SRFuzzer[26]	2019	网络	物理 SOHO 路由器设备	基于变异的网络模糊测试
ESRFuzzer[27]	2021	网络	物理 SOHO 路由器设备	输入语义模型和多种监控机制结合
Snipuzz[28]	2021	网络	IoT 固件	反馈驱动
DIANE[29]	2021	网络	IoT 固件	动态、静态联合分析
Fw-fuzz[30]	2022	网络	网络服务程序	遗传算法驱动
PeriScope[31]	2019	调试接口	无线通信外设/驱动程序	内存映射故障注入
HyFuzz[32]	2020	调试接口	Android 内核	动态接口恢复
TEEzz[33]	2023	调试接口	可信应用程序	动态二进制插桩
Gramatron[34]	2022	调试接口	数据处理固件	利用硬件跟踪的覆盖率引导
μAFL[35]	2022	调试接口	微控制器固件	反馈驱动
GDBFuzz[36]	2023	调试接口	嵌入式系统	硬件断点驱动
SHIFT[37]	2024	调试接口	微控制器固件	半托管驱动分析
LLM-BusyFuzz[38]	2024	调试接口	BusyBox 组件	基于 LLM 的种子生成并重用崩溃

4.1.1 通过网络进行的直接模糊测试方法

通过网络进行的直接模糊测试方法是一种利用网络传输协议的漏洞检测技术，它通过向目标网络应用程序发送异常或随机数据，来触发可能的漏洞或错误条件。

Zhang 等提出了一个用于发现物理 SOHO（Small Office/Home Office，小型办公室/家庭办公室）路由器设备多类型漏洞的自动模糊测试框架 SRFuzzer[26]，它由种子生成器、变异器、监视器、电源控制和配置模块组成。种子生成器收集原始请求并生成测试用例种子，变异器根据漏洞类型和属性标签生成变异后的请求，监视器检查连接状态和响应内容，电源控制模块通过智能插座控制设备的电源以持续进行模糊测试。此后，他们又提出 ESRFuzzer[27] 作为 SRFuzzer 的增强版本，

增加了配置生成器、精确模拟异常行为、改进监控机制等功能。ESRFuzzer 通过 K-V 模型和 C-R 模型精确建模数据格式和请求序列，旨在通过模拟异常行为和多种监控机制捕获漏洞，持续提高漏洞发现的效率，从而提高物理 SOHO 路由器设备的安全性。

针对传统黑盒模糊测试方法在对消息进行变异时缺乏对输入质量的了解而导致资源浪费和低质量输入的问题，Feng 等提出了一种优化的黑盒模糊测试框架 Snipuzz[28]。该框架利用设备返回的响应消息和推断消息的语法角色建立反馈机制，生成更具针对性的测试用例，以解决 IoT 设备固件中存在的漏洞。Snipuzz 首先向 IoT 设备发送原始输入消息并捕获响应消息；然后根据响应消息中的信息推断出消息中每个字节的语法角色；接着利用聚类算法对响应消息进行分析，将相似的消息片段聚类在一起；最后，根据聚类结果生成新的测试用例，通过对消息片段进行变异来探索潜在的漏洞。Snipuzz 有效地发现了隐藏在 IoT 设备固件中的漏洞，其中包括多个零日漏洞。

Redini 等提出了一种基于伴侣应用程序的动态模糊测试方法 DIANE[29]，其目标是通过识别应用程序中的模糊触发器来生成对 IoT 设备的输入，以发现潜在的安全漏洞。DIANE 通过静态分析识别可能向 IoT 设备发送网络消息的函数，然后使用动态分析捕获运行时生成的网络流量；随后利用混合分析确定数据转换函数和模糊触发器，最终利用这些触发器生成模糊数据进行测试，监视设备响应以检测漏洞和异常行为。通过这种方式，DIANE 能够识别潜在的漏洞和安全问题，并生成对 IoT 设备的模糊输入，以进行安全性评估和漏洞发现。

Gao 等提出了 Fw-fuzz 方法[30]，其通过动态插桩和遗传算法模型生成高代码覆盖率的测试用例。Fw-fuzz 已实现原型系统，并在不同架构的网络服务程序上得到应用。它解决了传统模糊测试效率低的问题，提高了固件模糊测试的效果和准确性。

4.1.2 通过调试接口进行的直接模糊测试

通过调试接口进行的模糊测试旨在通过自动生成有效输入，来测试系统内核驱动程序并发现潜在漏洞。

Song 等提出了一个用于硬件操作系统边界的探测和模糊测试框架 PeriScope[31]，旨在发现无线通信外设和驱动程序中的漏洞。它结合了两种外设接口机制 MMIO(Memory-Mapped Input/Output，内存映射输入输出)和 DMA(Direct Memory Access，直接内存访问)的分析需求，并通过模糊测试组件 PERIFUZZ 帮助用户在受损设备上找到可访问的设备驱动程序中的漏洞。PeriScope 的设计使其与驱动程序无关，可以分析各种类型的外设。其工作原理是自动检测目标设备驱动程序创建的 MMIO 或 DMA 的内存映射，并将支持的监视映射标记为不存在于内核页面表中，从而触发页面故障。当发生内核页面故障时，PeriScope 会调用钩子函数进行处理，以实现对驱动程序通信的监视和分析。

对 Android 设备的模糊测试也是一个重要议题。Lu 等提出了一种 Android 内核模糊测试方法[32]，可以有效检测攻击者获得 root 权限的内核漏洞。该研究提出了一种混合接口恢复方法，在无法访问源代码的情况下恢复内核驱动程序的接口，通过动态接口恢复和逆向工程技术，在白盒和黑盒条件下分别识别接口知识并推断参数类型，提升了 Android 系统的安全性。与此同时，也有就商用 Android 设备上对受信任应用程序的模糊测试。Busch 等提出了 TEEzz[33]，它通过分析和记录与受信任应用程序之间的接口交互来进行模糊测试。首先识别出与受信任应用程序交互的客户端应用程序库，并生成接口定义和消费者来触发交互；然后使用动态二进制插桩来记录与受信任应用程序的交互，并通过分析交互数据的依赖关系来推断受信任应用程序的 API 模型。

Beckmann 等提出了一种覆盖率引导的灰盒模糊测试技术[34]，能够有效识别输入数据处理中的安全漏洞。该技术利用硬件跟踪接口进行测试，避免了仿真平台的局限性和不准确性，同时无需更改目标系统，具备高实用性和可扩展性。作者还开发了一种启发式种子选择策略，结合硬件断点和轻量级静态分析实现了对新应用行为的可靠检测。

基于重新托管的解决方案因不能准确地模拟外设行为而无法用于模糊相应驱动程序代码的问题，Li 等提出了一种针对微控制器固件的非侵入式反馈驱动模糊测试方法 μAFL[35]。该方法通过在实际硬件上运行目标固件，并利用调试工具与 PC 通信，实现了对固件的全栈测试。μAFL 还利用 ETM(Embedded Trace Macrocell，嵌入式跟踪宏单元)收集代码覆盖信息，以支持灰盒反馈驱动的模糊测试。其模块化设计可扩展，能与现有模糊测试工具集成，为开发人员定位固件漏洞提供了有效途径。

嵌入式系统的微控制单元、体系结构和操作系统高度多样化，且微控制器板上的软件难以更改。为此，Eisele 等提出了 GDBFuzz[36]，一种基于硬件断点和控制流图的覆盖率引导模糊测试方法，用于测试嵌入式系统的鲁棒性。该方法利用大多数微控制器具有的调试接口的特点，通过调试探针(通常用 GNU 调试器 GDB 进行控制)设置有限数量的硬件断点，提取部分覆盖反馈，甚至是未经检测的二进制代码，从而实现有效的模糊测试。GDBFuzz 是首个基于硬件的、架构无关的、源代码无关的、非侵入式的、易于应用的嵌入式系统覆盖率引导模糊测试的方法。实验表明，GDBFuzz 可以在多种微控制器板和用户应用程序上轻松应用，并且可以检测到许多已知和新的漏洞。

针对传统仿真测试常导致的误报和低保真问题，Mera 等提出了 SHiFT[37]，它专为嵌入式微控制器设计，其核心思想是通过在目标硬件上运行大部分模糊测试任务来提升测试的精确度和相关性，同时减少测试中的误报。SHiFT 通过将模糊测试的计算任务和反馈机制直接绑定到实际设备上，来更真实地模拟嵌入式系统的运行环境，并快速捕获漏洞及异常行为。它兼容多种微控制器架构，适合高效且深度的嵌入式固件安全评估。

目前基于大语言模型(Large Language Model，LLM)的嵌入式软件模糊测试取得了重要进展。Asmita 等提出了一种结合 LLM 和崩溃重用的测试方法[38]，主要应用于 BusyBox 组件的漏洞检测。该方法首先通过 LLM 生成目标特定且符合目标系统语法的高质量种子，大幅提升了模糊测

试的效率和崩溃发现率。其次，研究还引入了崩溃重用技术，通过复用之前测试中收集的崩溃输入，快速评估新目标中的相似漏洞，无需重新进行完整的模糊测试。两种技术的结合，不仅提高了测试的效率，还显著提升了漏洞检测的速度。

4.2 基于仿真的模糊测试

相对于通用平台来说，嵌入式系统在性能和扩展性方面存在一定的劣势，直接对实际嵌入式设备进行动态分析可能面临以下问题：1）测试速度慢，因为嵌入式设备通常使用微控制单元，计算能力相当有限；2）消耗大量人力，因为现有的分析工具无法直接在嵌入式系统上使用，需要依靠人工分析来辅助运行；3）成本高，实际运行待测试软件时批量购买设备将消耗较高成本。为解决这些问题，提出了固件仿真技术，该技术在通用平台上提供了一个完全虚拟的环境，在固件能够正确模拟的情况下利用现有的模糊器和通用平台上的资源完成高效的测试。

目前，根据外围设备是否被仿真，基于固件仿真的模糊测试主要包括部分仿真、全仿真两种方法。大多数固件仿真通常采用全仿真技术，这意味着它模拟了整个目标系统的硬件和环境，包括处理器、内存、外设和操作系统等，并且使用现有的模糊测试工具进行测试分析。然而，在实际的模糊测试过程中，根据具体的测试需求，可以仅对相关的硬件进行仿真，以降低仿真所带来的资源消耗，这种方法被称为部分仿真技术。

4.2.1 部分仿真方法

固件在运行时频繁与外设交互，因此其正确运行依赖于外设支持，而部分仿真方法是一种在模糊测试过程中选择性模拟目标应用程序的部分特定行为、条件或环境的技术。代表性方法如下。

Markus 等提出了一种结合外围设备通信缓存和运行时程序状态近似的模糊测试方法[39]，以改善嵌入式固件的安全分析效果。该方法旨在解决虚拟分析环境无法访问外围设备和分析环境减速问题。其核心思想在于利用缓存外围设备通信和运行时程序状态近似来提高嵌入式固件安全分析的效果和效率，但该方法受限于程序状态爆炸问题。

2014 年，Zaddach 等提出了 Avatar[40] 框架，这是最早基于 QEMU 设计的可实现硬件转发的框架，通过将模拟器和真实硬件的执行协同起来，实现复杂的动态分析。它提供了基本机制来转发模拟器的 I/O 访问至嵌入式设备，同时采用多种技术来提高系统性能。其被应用于逆向工程、漏洞发现和后门检测等不同场景，并展示了在各种设备上的灵活性。Avatar 为研究人员提供了一个用于准确分析嵌入式系统固件二进制文件安全性的强大工具。

与 Avatar 类似，为了使分析人员能够利用现有的动态分析工具对嵌入式设备进行分析，Platzer 等提出了 PROSPECT[41]，该系统克服了动态代码分析的困难，并实现了在任意分析环境中对嵌入式二进制代码的动态分析。与 Avatar 相比，PROSPECT 通过利用总线连接来转发交互的方法提高了仿真器与真实硬件之间的交互效率，提高了仿真速度。

2015 年，Koscher 等提出了一个能够近乎实时地模拟和插桩嵌入式系统的 SURROGATES[42] 系统。该系统通过使用自定义的 FPGA（Field-Programmable Gate Array，现场可编程门阵列）桥接器将主机的 PCIe 总线与嵌入式系统连接，实现了近乎实时的嵌入式系统仿真。这使得模拟器可以准确地访问系统外围设备，并支持各种动态分析技术。SURROGATES 的系统与外设硬件之间的转发和状态传输速度更快、更稳定，但同时增加了硬件成本。

在 Avatar 的基础上，2018 年 Muench 等进一步提出了 Avatar2[43] 框架，该框架是一种允许各种工具交互的动态多目标编排框架，旨在实现不同动态二进制分析框架、调试器、模拟器和实际物理设备之间的互操作性。它允许分析人员将多个工具整合到一个统一环境中，以更好地利用其功能并解决复杂问题，通过实现拓扑结构和二进制代码的跨系统执行，提供了灵活性和可配置性。

2018 年，Corteggiani 等提出了 Inception[44]，其主要将基于 KLEE 的高级命令转换为低级的 JTAG 信号，这也是经典的使用硬件在环方法的固件仿真器，尽管从 AVATAR 到 Inception 都在不断优化硬件在环的固件仿真器的运行效率，但是依旧不能满足如今大规模固件自动动态分析的需要。

2019 年，Gustafson 等提出 Pretender[45] 系统，旨在将硬件环境和外设仿真分开。该系统记录了固件在真实硬件环境中的反馈，并利用机器学习生成不同的外设仿真模型，以支持全系统固件仿真。然而，Pretender 仍然依赖于硬件。

具体部分仿真模糊测试方法、工具如表 5 所列。

表 5 部分仿真模糊测试的方法与工具

Table 5 Partial simulation fuzzing approaches and tools

工具	年份	测试方法	主要优点
Avatar[40]	2014	硬件转发	硬件转发，支持复杂动态分析
PROSPECT[41]	2014	通过总线连接转发	高效总线连接，提高仿真速度
SURROGATES[42]	2015	FPGA 桥接实时仿真	FPGA 桥接，近乎实时的仿真
Avatar2[43]	2018	动态多目标编排	动态多目标编排，工具交互性强
Inception[44]	2018	JTAG 信号转换	JTAG 信号转换，全面固件仿真
Pretender（全仿真）[45]	2019	机器学习生成外设仿真	分离硬件环境，外设仿真灵活

4.2.2 全仿真方法

由于部分仿真方法需获取实际外设，其扩展性和效率受限于外设交互，为实现嵌入式设备的可扩展分析，需对固件内核及外设进行仿真虚拟化，即全仿真。在全仿真模式下，不同固件仿真器采用多种方法来处理外设信息，基于这些方法的差异，本节将固件仿真器分为以下 6 类（见表 6）：基于高吞吐量的固件仿真模型、基于固件重托管的外设仿真模型、基于符号执行的外设仿真模型、基于预设访问模式的外设仿真模型、基于 MMIO 建模的外设仿真模型和基于硬件规范的固件仿真模型。

1)基于高吞吐量的固件仿真模型

2019 年,Zheng 等提出了 FIRM-AFL[46],以解决 IoT 模糊测试中的兼容性和性能瓶颈问题。它通过系统仿真器实现对 POSIX 兼容固件的模糊测试,结合系统级与用户模式仿真的增强过程仿真技术,实现了高兼容性和高吞吐量测试。评估表明,FIRM-AFL 能发现真实世界中 IoT 程序的漏洞,其吞吐量平均比基于系统级仿真的模糊测试高 8.2 倍,且能更快地发现 1-day 漏洞和 0-day 漏洞。

同年,Kim 等提出了 FirmAE[47],旨在提供大规模的 IoT 固件仿真以进行动态分析。通过对 437 个仿真失败案例的实证研究,FirmAE 系统化了故障处理的启发式方法,并将其应用于仿真环境,较 Firmadyne[48]的仿真成功率高 487%。

2)基于固件重托管的外设仿真模型

2020 年,Clements 等提出了 HALucinator[49],HALucinator 利用通用系统模拟器来实现二进制固件的仿真,无需依赖实际硬件。该工具将固件分解为可重用的抽象组件,从而支持多种芯片型号和复杂外设。其既可用于交互式动态分析,也可用于模糊测试。

3)基于符号执行的外设仿真模型

Chen 等提出了基于符号执行的外设仿真方法 Laelaps[50],该方法通过符号执行探索可达路径,并生成相应的路径约束作为外设反馈模拟的依据。Laelaps 首次提出结合符号执行来进行外设反馈模拟,能够在无需预先了解目标设备的情况下运行微控制器设备的软件。Laelaps 通过符号执行辅助外设仿真,生成适当的输入以在运行时引导具体执行。该技术可用于动态分析 IoT 设备,并有助于防范针对低端微控制器设备的安全漏洞。

2021 年提出的 μEmu[51]方法是基于符号执行生成合法路径约束的外设仿真模型。该方法通过监测程序的异常状态自动判断合理的执行路径,实现了外设建模与模拟执行的分离。首先,利用符号执行生成合理路径约束并构建外设反馈知识库;然后在仿真过程中利用静态知识库避免频繁切换符号执行状态,解决了类似 Laelaps 的性能开销问题。

类似于 μEmu,2021 年推出了名为 Jetset[52]的工具,其通过路径约束条件来实现固件的重新托管。它可以推断嵌入式系统中外设的预期行为,并精确模拟硬件设备,使固件能够在非本地环境中运行。该工具在多种体系结构、应用领域和操作系统的 13 个固件上进行了测试,并成功执行了这些固件。

4)基于预设访问模式的外设仿真模型

2020 年提出的 P2IM[53]技术框架使用了基于人为假设的寄存器访问模式的外设仿真方案,通过外设抽象化和自动生成的模型来处理固件 I/O,从而实现对固件的持续执行和动态测试。该方法不依赖于任何硬件,也不模拟外围设备,只要模拟器提供固件,固件就可以在没有真实的或完全模拟的外设的模拟器上执行。P2IM 可以与商用模糊器集成,从而使固件测试更具可扩展性和硬件独立性。

2021 年发布的 DICE[54]也采取了类似方法,帮助固件分析器模拟 DMA 输入通道,从而提高了对于嵌入式设备的安全性分析。DICE 适用于不同类型的 MCU 固件和嵌入式架构,具有硬件无关性。实验评估表明,DICE 能够高度准确地识别并模拟 DMA 输入通道,发现现有固件分析器未能检测到的新漏洞。

5)基于 MMIO 建模的外设仿真模型

Gao 等提出了名为 EM-Fuzz[55]的固件漏洞检测技术,该技术结合了模糊测试和实时内存检查,通过记录内存敏感操作来指导模糊测试,利用传统分支覆盖率生成高质量种子,并通过实时内存检查器暴露内存漏洞。EM-Fuzz 在多个实际嵌入式固件上发现了 23 个漏洞,其中 16 个是之前未知的。EM-Fuzz 提高了固件漏洞检测的性能和效率,为提高嵌入式系统安全性提供了一种有效方案。

2022 年,为提高代码覆盖率,解决重托管和自动化硬件重托管技术中不完全的开销消除的问题,Scharnowski 等提出了 Fuzzware[56]。Fuzzware 使用精确的 MMIO 建模方法来识别和模拟硬件设备的行为,从而减小模糊测试中的输入开销。实验结果表明,Fuzzware 可以消除高达 95.5%的输入开销,使模糊测试更加高效。

2023 年,Scharnowski 等对 Fuzzware 进行了改进,提出了 Hoedur[57],这是一种基于多流输入的嵌入式固件模糊测试方法,将以前的扁平、顺序和不透明的输入重新组织为多个严格类型和连贯的流,从而使模糊器能够执行类型感知的变异并保持其进展。评估表明,Hoedur 的覆盖率比现有的固件模糊器高出 5 倍,发现已知漏洞的速度可提升 550 倍,且发现了 23 个未知漏洞。

另一种是由 Chesser 等提出的 ICICLE[58]方法。ICICLE 是一个重新设计的灰盒固件模糊测试仿真器,专为支持多架构固件模糊测试而设计,通过动态检测和反馈优化测试输入,以探索和发现潜在的漏洞。

2023 年,Ember-IO[59]作为一种新型固件模糊测试框架被提出。该框架采用模型无关的内存映射 IO、外设输入回放和 FERMCov 等技术,来解决固件测试中的输入空间搜索问题,进一步提高了代码覆盖率。实验表明,Ember-IO 在多个固件测试中取得了显著的性能提升。

2024 年,Chesser 等提出了 MULTIFUZZ[60]。MULTIFUZZ 通过提升固件与外设交互的测试精度实现了更高效的模糊测试。其核心原理是将 MMIO 寄存器交互划分为独立的输入流,减少外设间的干扰,从而确保输入的稳定性和精度。MULTIFUZZ 能够自动检测和优化输入流,显著提高了测试效率。

6)基于硬件规范的固件仿真模型

针对大规模外设仿真的复杂性和准确性不足的问题,Zhou 等开发了 SEmu[61]。该框架利用自然语言处理技术将外设行为描述转换成结构化的条件-动作规则,这些规则根据固件的实际执行动态合成,能显著提高仿真保真度。SEmu 还引入了基于无效性的引导机制,识别并校正可能出现的错误或缺失的条件-动作规则,以提高仿真准确性。

表 6 全仿真模糊测试方法

Table 6 Full simulation fuzzing methods

方法	年份	测试种类	主要优点
FIRM-AFL[46]	2019	基于高吞吐量的固件仿真	提高 IoT 模糊测试兼容性和性能
P2IM[53]	2020	基于预设外设访问模式的外设仿真	实现硬件无关、动态测试
Laelaps[50]	2020	基于符号执行的外设仿真	符号执行辅助外设反馈仿真
HALucinator[49]	2020	基于固件重托管的外设仿真	支持多种芯片和复杂外设，交互式动态分析
FirmAE[47]	2020	基于高吞吐量的固件仿真	高仿真成功率，系统化故障处理启发式算法
EM-Fuzz[55]	2020	基于 MMIO 建模的外设仿真	结合模糊测试和内存检查，提高漏洞检测效率
Firmcorn[62]	2020	基于符号执行的外设仿真	操作系统无关，提高测试覆盖率和自动化程度
Jetset[52]	2021	基于符号执行的外设仿真	精确模拟硬件设备，推断外设预期行为
DICE[54]	2021	基于预设外设访问模式的外设仿真	模拟 DMA 输入通道，提高安全性分析
μEMU[51]	2021	基于符号执行的外设仿真	分离外设建模与模拟执行，减少性能开销
PGFUZZ[63]	2021	基于高吞吐量的固件仿真	高效生成测试大量输入，自动化程度高
FIRM-COV[64]	2021	基于 MMIO 建模的外设仿真	通过量化测试的有效性，提高代码覆盖率
Semu[61]	2022	基于硬件规范的固件仿真	高级仿真框架，自动化生成外设仿真模型
FIRMWIRE[65]	2022	基于符号执行的外设仿真	结合符号执行进一步优化输入生成
Fuzzware[56]	2022	基于 MMIO 建模的外设仿真	精确 MMIO 建模，减少输入开销
EQUAFL[66]	2022	基于高吞吐量的固件仿真	通过等价类划分减少冗余测试
Hoedur[57]	2023	基于 MMIO 建模的外设仿真	多流输入提高覆盖率和漏洞发现速度
Ember-IO[59]	2023	基于 MMIO 建模的外设仿真	提高测试效率和代码覆盖率
ICICLE[58]	2023	基于 MMIO 建模的外设仿真	动态检测和反馈优化测试输入
MULTIFUZZ[60]	2024	基于 MMIO 建模的外设仿真	外设交互的高效处理和测试精度提升

5 嵌入式软件模糊测试技术优化与展望

目前，嵌入式模糊测试技术发展迅速，但相比于通用模糊测试，在测试覆盖率以及漏洞发现效率方面仍有较大提升空间。下文将从优化方法和未来技术展望的角度探讨嵌入式模糊测试的发展趋势。

5.1 嵌入式软件模糊测试优化方法

5.1.1 基于 MMIO 建模的优化方法

模糊器在生成模糊输入时，并不了解具体的 I/O 语义，例如外设读取的长度、位掩码、有效值范围等，导致大量模糊输入对嵌入式设备无效。为此，可以使用符号执行技术对 MMIO 访问进行动态建模，通过建模将原始模糊输入翻译成符合固件要求的值，从而提高模糊输入的利用效率。代表性的方法有 Fuzzware，DICE，P2IM 和 μEmu。

以 Fuzzware 为例，基于 MMIO 建模方法的基本流程如图 3 所示。Fuzzware 使用动态符号执行进行细粒度建模，通过自动生成和应用 MMIO 访问模型显著提高模糊测试的效率和覆盖率。评估显示，Fuzzware 平均代码覆盖率比 P2IM 高出约 44%，比 μEmu 高出约 61%，并且发现了 12 个未知漏洞。

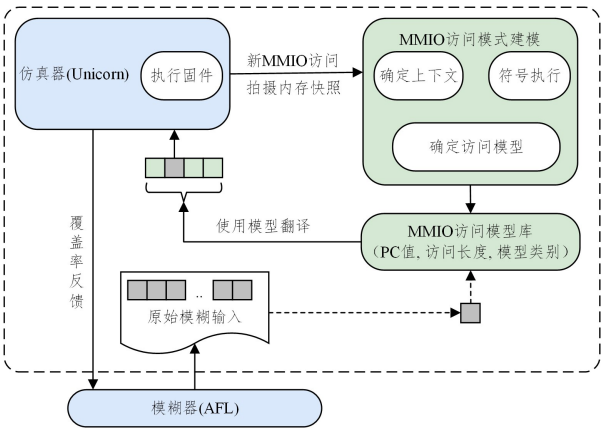


图 3 基于 MMIO 建模方法的基本流程

Fig. 3 Basic process of MMIO-based modeling method

综上所述，通过使用 MMIO 访问模型对嵌入式模糊测试输入进行相关约束可以有效提升模糊测试效率和代码覆盖率，提高代码漏洞发现能力。

5.1.2 基于多模糊器处理的优化方法

在嵌入式模糊测试研究中，处理不同外设访问的数据格式是一个关键问题。现有的工作通常使用单一模糊器为各种固件提供模糊输入，导致不同外设访问的数据掺杂在一起，难以提供高质量的模糊输入。针对这一问题，可以设计多个模糊器分别处理不同类型的外设，通过修改仿真器以支持多流输入，从而有效生成变异输入。

以 Hoedur 为例，其基于多模糊器处理的模糊测试平台的基本框架如图 4 所示。Hoedur 采用静态分析和动态跟踪结合的方法，专注于发现深层次的固件漏洞。它利用静态分析获取固件代码的结构和控制流信息，然后通过动态跟踪捕获运行时行为，生成高效的模糊测试输入。Hoedur 的显著特点在于其处理复杂控制流和多分支路径时的高效性，以及其多模糊器架构能够针对不同类型的外设生成专门的变异输入，从而提高模糊测试的覆盖率和漏洞发现能力。

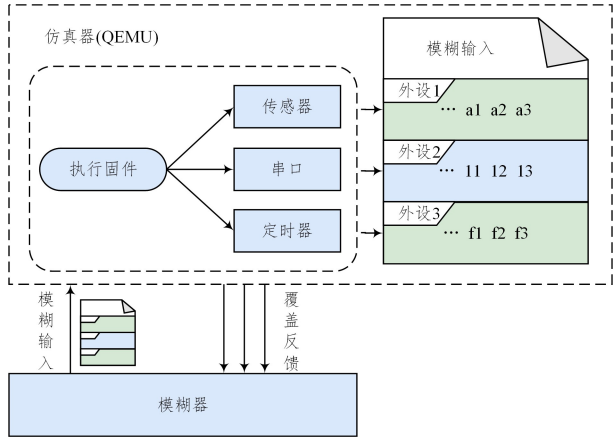


图 4 基于多模糊器处理的模糊测试平台

Fig. 4 Fuzzing platform based on multi-fuzzer processing

多模糊器方法通过针对不同类型外设生成专门的变异输入，并通过仿真器的修改支持多流输入，显著提升了模糊测试

的覆盖率和漏洞发现能力。这种方法不仅提高了测试的广度和深度，还确保了对嵌入式系统中潜在漏洞的更全面覆盖。

5.1.3 基于模糊器优化的优化方法

在嵌入式模糊器优化方面，有3种优化的方法：静态分析、突变策略和种子调度。这3种方法在模糊器工作流程中的位置如图5所示。

静态分析通过对程序源代码或反汇编代码的分析挖掘代码结构、变量和控制流信息，识别出代码中的关键部分，如复杂函数、历史出错位置或安全敏感区域。这些信息为模糊测试提供了先验知识，使得模糊测试更有方向性和有效性。例如，定向模糊测试技术（如AFLGo，fuzzguard，WindRanger）以及符号执行技术（如Driller，KLEE）都采用了静态分析技术来提高模糊测试效果。

突变策略是在模糊测试中选择最佳变异操作符和变异位点的过程。合适的突变策略可以显著提升模糊测试在发现新边缘覆盖和暴露崩溃方面的效果。现有的突变改进方法根据历史突变数据优化变异算子的概率分布，并找到最佳的变异位置。例如，Fairfuzz，MOPT，DARWIN等工具通过调整突变策略提高了模糊测试的效果。

种子调度是在模糊测试过程中确定将使用的种子输入，并决定它们的顺序和优先级。原始种子通过突变产生的大量新输入中可能存在对测试没有贡献的输入，通过种子调度可以避免执行这些无用的输入，从而提高测试效率。例如，AFLFast，Collafl，Ecofuzz，SLIME，AFL++等工具对种子调度算法进行了优化，以提高模糊测试的效果。

这3种方法的结合能够显著提升嵌入式模糊测试的效率和效果，使其更高效地发现潜在的安全漏洞和缺陷。通过静态分析提供方向性指导、突变策略优化变异过程和种子调度提升输入管理，使模糊测试更加精准、有效，从而确保更全面的漏洞挖掘。

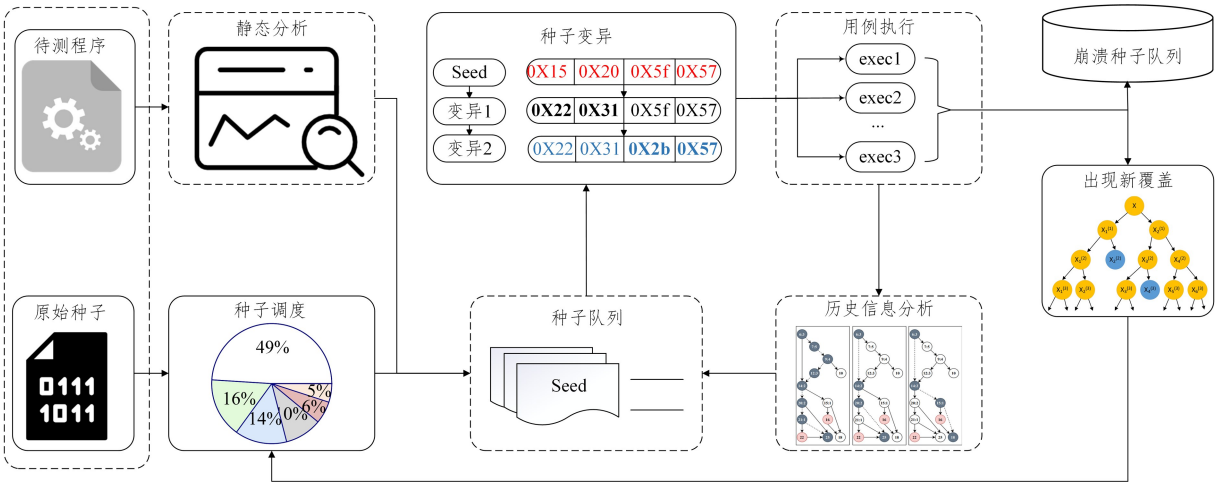


图5 模糊器的工作流程

Fig. 5 Workflow of fuzzer

5.2 嵌入式软件模糊测试技术展望

5.2.1 基于人工智能技术的嵌入式模糊测试

人工智能技术在应对最新的安全挑战方面日益受到关注。在嵌入式系统领域，一些方法采用机器学习技术重新托管嵌入式系统的固件，通过训练模型来模拟外围设备，在仿真环境中执行固件样本并进行后续的模拟分析。

同时，随着LLM在人工智能领域的迅速发展，将其与模糊测试相结合的技术也逐渐应用于嵌入式模糊测试领域，这为嵌入式系统的安全评估和漏洞检测带来了新的突破。随着LLM驱动的模糊测试技术不断成熟，它将为嵌入式系统的安全测试提供更高效、更智能的解决方案。

5.2.2 针对实时嵌入式系统的模糊测试技术

实时嵌入式系统在国防军工、关键基础设施、医疗设备等领域有着广泛应用，其可靠性不仅依赖于功能的正确执行，还取决于功能执行的实时性约束。通过模糊测试生成和发送具有精确时间特性的输入数据，以测试系统在特定时间条件下的响应，进而发现与时间相关的漏洞。然而，目前针对实时性漏洞检测的嵌入式模糊测试技术尚缺少具有实用价值的工作，未来可以进行进一步深入探索。

结束语 随着嵌入式设备的广泛应用，嵌入式系统面临的安全威胁日益增加。这些安全挑战使嵌入式软件模糊测试技术得到了广泛关注，成为研究热点。本文针对嵌入式系统特点，对模糊测试技术在该领域的应用进行了深入讨论，系统介绍了嵌入式软件模糊测试的相关方法和工具，分析了其所面临的挑战，并讨论了未来发展方向。通过上述工作，希望能够促进对嵌入式软件模糊测试的理解，更好地应对未来的安全挑战。

参 考 文 献

[1] LI D. Analysis of the Stuxnet Virus Incident and Insights for Enhancing Industrial Control Security Protection [J]. Network Security Technology & Application, 2019, 1: 9-10, 24.

[2] ANTONAKAKIS M, APRIL T, BAILEY M, et al. Understanding the mirai botnet [C]// Proceedings of the 26th USENIX Conference on Security Symposium. Berkeley: USENIX Association, 2017: 1093-1110.

[3] MILLER B P, FREDRIKSEN L, SO B. An empirical study of the reliability of UNIX utilities [J]. Communications of the ACM, 1990, 33(12): 32-44.

[4] KAKSONEN R, LAAKSO M, TAKANEN A. Software Security

Assessment Through Specification Mutations and Fault Injection [M]. Berlin: Springer, 2001: 173-183.

[5] GODEFROID P. Random testing for security: blackbox vs. whitebox fuzzing [C]// Proceedings of the 2nd International Workshop on Random testing: Co-Located with the 22nd IEEE/ACM International Conference on Automated Software Engineering. Los Alamitos: IEEE Computer Society Press, 2007: 206-215.

[6] ZALEWSKI M. American fuzzy lop [EB/OL]. (2020-11-11) [2023-08-18]. https://lcamtuf.coredump.cx/afl/.

[7] LI J, ZHAO B, ZHANG C. Fuzzing: a survey [J]. Cybersecurity, 2018, 1: 1-13.

[8] NIDHRA S, DONDETI J. Black box and white box testing techniques-a literature review [J]. International Journal of Embedded Systems and Applications, 2012, 2(2): 29-50.

[9] 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: Association for Computing Machinery, 2017: 2329-2344.

[10] BÖHME M, PHAM V T, ROYCHOUDHURY A. Coverage-based Greybox Fuzzing as Markov Chain [J]. IEEE Transactions on Software Engineering, 2019, 45(5): 489-506.

[11] CHEN P, CHEN H. Angora: Efficient fuzzing by principled search [C]// 2018 IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2018: 711-725.

[12] NAGY S, HICKS M. Full-speed fuzzing reducing fuzzing overhead through coverage-guided tracing [C]// Proceedings of 2019 IEEE Symp on Security and Privacy. San Francisco: IEEE, 2019: 787-802.

[13] BELLARD F. QEM U. a fast and portable dynamic translator [C]// USENIX Annual Technical Conference. Berkeley: USENIX Association, 2005: 41-46.

[14] SCHWARTZ E J, AVGERINOS T, BRUMLEY D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution(but might have been afraid to ask) [C]// Proceedings of 2010 IEEE Symp on Security and Privacy. San Francisco: IEEE, 2010: 317-331.

[15] KIM J, KIM T, IM E G. Survey of dynamic taint analysis [C]// 4th IEEE International Conference on Network Infrastructure and Digital Content. Beijing: IEEE, 2014: 269-272.

[16] MAGNUSSON P S, CHRISTENSSON M, ESKILSON J, et al. Simics: A full system simulation platform [J]. Computer, 2002, 35(2): 50-58.

[17] QUYNH N A. Unicorn Engine [EB/OL]. [2023-08-18]. https://www.unicornengine.org/.

[18] NGUYEN A Q, ZALEWSKI M. Qiling Framework Documentation [EB/OL]. [2024-07-10]. https://docs.qiling.io/en/latest/.

[19] CHEN D D, WOO M, BRUMLEY D, et al. Towards automated dynamic analysis for linux-based embedded firmware [C]// 23rd Annual Network and Distributed Systems Security Symposium. San Diego: ISOC, 2016: 1-16.

[20] STEPANOV E, SEREBRYANY K. Memory- Sanitizer: Fast detector of uninitialized memory use in C++ [C]// 2015 IEEE/ACM International Symposium on Code Generation and Optimization. Los Alamitos: IEEE COMPUTER SOC, 2015: 46-55.

[21] SEREBRYANY K, BRUENING D, A POTAPENKO, et al. AddressSanitizer: A fast address sanity checker [C]// Proceedings of the 2012 USENIX Conference on Annual Technical Conference. Berkeley: USENIX Association, 2012: 309-318.

[22] SEREBRYANY K, ISKHODZHANOV T. ThreadSanitizer: Data race detection in practice [C]// Proceedings of the Workshop on Binary Instrumentation and Applications. New York: Association for Computing Machinery, 2009: 62-71.

[23] NORHUZAIMIN J, MAIMUN H H. The design of high speed UART [C]// 2005 Asia-Pacific Conference on Applied Electromagnetics. Johor: IEEE, 2005.

[24] ROSENFELD K, KARRI R. Attacks and defenses for JTAG [J]. IEEE Design & Test of Computers, 2010, 27(1): 36-47.

[25] GOSAIN A, SHARMA G. A survey of dynamic program analysis techniques and tools [C]// Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications. Berlin: Springer Verlag, 2015: 113-122.

[26] ZHANG Y, HUO W, JIAN K, et al. SRFuzzer: An Automatic Fuzzing Framework for Physical SOHO Router Devices to Discover Multi-Type Vulnerabilities [C]// Proceedings of the 35th Annual Computer Security Applications Conference. New York: Association for Computing Machinery, 2019: 544-556.

[27] ZHANG Y, HUO W, JIAN K, et al. ESRFuzzer: an enhanced fuzzing framework for physical SOHO router devices to discover multi-Type vulnerabilities [J]. Cybersecurity, 2021, 4: 1-22.

[28] FENG X, SUN R, ZHU X, 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: Association for Computing Machinery, 2021: 337-350.

[29] REDINI N, CONTINELLA A, DAS D, et al. DIANE: Identifying Fuzzing Triggers in Apps to Generate Under-constrained Inputs for IoT Devices [C]// 2021 IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2021: 484-500.

[30] GAO Z, DONG W, CHANG R. Fw-fuzz: A code coverage-guided fuzzing framework for network protocols on firmware [J]. Concurrency and Computation: Practice and Experience, 2020, 34(16): 5756.

[31] SONG D, HETZELT F, DAS D, et al. PeriScope: An Effective Probing and Fuzzing Framework for the Hardware-OS Boundary [C]// 2019 Network and Distributed Systems Security Symposium. San Diego: Internet Society, 2019: 1-15.

[32] LU S, KUANG X, NIE Y, et al. A Hybrid Interface Recovery Method for Android Kernels Fuzzing [C]// 2020 IEEE 20th International Conference on Software Quality, Reliability and Security. New York: IEEE, 2020: 335-346.

[33] BUSCH M, MACHIRY A, SPENSKY C, et al. TEEzz: Fuzzing Trusted Applications on COTS Android Devices [C]// 2023 IEEE Symposium on Security and Privacy. San Francisc: IEEE, 2023: 1204-1219.

[34] BECHMANN M, STEFFAN J. Coverage-Guided Fuzzing of Embedded Systems Leveraging Hardware Tracing [C]// European Symposium on Research in Computer Security. Copenhagen: Springer International Publishing, 2022: 362-378.

[35] LI W, SHI J, LI F, et al. μAFL: Non-intrusive Feedback-driven Fuzzing for Microcontroller Firmware [C]// Proceedings of the 44th International Conference on Software Engineering. New York: Association for Computing Machinery, 2022: 1-12.

[36] EISELE M, EBERT D, HUTH C, et al. Fuzzing Embedded Systems using Debug Interfaces [C]// Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: Association for Computing Machinery, 2023: 1031-1042.

[37] MERA A, LIU C, SUN R, et al. SHiFT: Semi-hosted Fuzz Testing for Embedded Applications [C]// 33rd USENIX Security Symposium. Berkeley: USENIX Association, 2024: 5323-5340.

[38] ASMITA, OLIINYK Y, SCOTT M, et al. Fuzzing BusyBox: Leveraging LLM and Crash Reuse for Embedded Bug Unearthing [C]// 33rd USENIX Security Symposium. Berkeley: USENIX Association, 2024: 883-900.

[39] KAMMERSTEETTR M, BURIAN D, KSASTNER W. Embedded security testing with peripheral device caching and runtime program state approximation [C]// 10th International Conference on Emerging Security Information, Systems and Technologies. Heidelberg: Springer Verlag, 2016: 21-26.

[40] ZADDACH J, BRUNO L, FRANCILLON A, et al. AVATAR: a framework to support dynamic security analysis of embedded systems' firmwares [C]// Network and Distributed Systems Security Symposium. San Diego: ISOC, 2014: 1-16.

[41] KAMMERSTETTER M, PLATZER C, KASTNER W. Prospect: peripheral proxying supported embedded code testing [C]// Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. New York: ACM, 2014: 329-340.

[42] KOSCHER K, KOHNO T, MOLNAR D. SURROGATES: enabling near-real-time dynamic analyses of embedded systems [J]. WOOT, 2015, 15: 7-16.

[43] MUENCH M, NISI D, FRANCILLON A, et al. AVATAR2: A multi-target orchestration platform [C]// Proceedings of 2018 Network and Distributed Systems Security Symposium. San Diego: ISOC, 2018: 1-11.

[44] CORTEGGIANI N, CAMURATI G, FRANCILLO A. Inception: System-wide security testing of real-world embedded systems software [C]// Proceedings of the 27th USENIX Security Symposium. Berkeley: USENIX Association, 2018: 309-326.

[45] GUSTAFSON E, MUENCH M, SPENSKY C, et al. Toward the analysis of embedded firmware through automated re-hosting [C]// Proceedings of the 22nd Int Symp on Research in Attacks, Intrusions and Defenses. Berkeley: USENIX Association, 2019: 135-150.

[46] ZHENG Y, DAVANIAN A, YIN H, et al. FIRM-AFL: high-throughput greybox fuzzing of IoT firmware via augmented process emulation [C]// USENIX Security Symposium. Berkeley: USENIX Association, 2019: 1099-1114.

[47] KIM M, KIM D, KIM E, et al. FirmAE: Towards largescale emulation of IoT firmware for dynamic analysis [C]// Proceedings of Annual Computer Security Applications Conference. New York: ACM, 2020: 733-745.

[48] CHEN D D, WOO M, BRUMLEY D, et al. Towards automated dynamic analysis for Linux-based embedded firmware [C]// Proceedings of 2016 Network and Distributed Systems Security Symposium. San Diego: ISOC, 2016.

[49] CLEMENTS A A, GUSTAFSON E, SCHARNOWSKI T, et al. HALucinator: Firmware Re-hosting through abstraction layer emulation [C]// Proceedings of the 29th USENIX Security Symposium. Berkeley: USENIX Association, 2020: 1201-1218.

[50] CHEN C, LE G, JIANG M, et al. Device-agnostic firmware execution is possible: A concolic execution approach for peripheral emulation [C]// Proceedings of In Annual Computer Security Applications Conference. New York: ACM, 2020: 746-759.

[51] ZHOU W, GUAN L, LIU P, et al. Automatic firmware emulation through invalidity-guided knowledge inference [C]// Proceedings of the 30th USENIX Security Symposium. Berkeley: USENIX Association, 2021: 2007-2024.

[52] JOHNSON E, BLAND M, ZHU Y F, et al. Jetset: targeted firmware rehosting for embedded systems [C]// USENIX Security Symposium. Berkeley: USENIXAssociation, 2021: 321-338.

[53] FENG B, MERA A, LU L. P2im: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling [C]// Proceedings of the 29th USENIX Conference on Security Symposium. Berkeley: USENIXAssociation, 2020: 1237-1254.

[54] MERA A, FENG B, LU L, et al. DICE: Automatic Emulation of DMA Input Channels for Dynamic Firmware Analysis [C]// IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2021: 1938-1954.

[55] GAO J, XU Y, JIANG Y, et al. Em-fuzz: augmented firmware fuzzing via memory checking [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(11): 3420-3432.

[56] SCHARNOWSKI T, BARS N, SCHLOEGEL M, et al. Fuzzware: Using precise MMIO modeling for effective firmware fuzzing [C]// Proceedings of the 31st USENIX Security Symposium. Berkeley: USENIX Association, 2022: 1239-1256.

[57] SCHARNOWSKI T, WÖRNER S, BUCHMANN F, et al. Hoedur: Embedded Firmware Fuzzing using Multi-Stream Inputs [C]// USENIX Security Symposium 2023. Berkeley: USENIX Association, 2023: 2885-2902.

[58] CHESSER M, NEPAL S, RANASINGHE D C. Icicle: a re-designed emulator for grey-box firmware fuzzing [C]// Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: Association for Computing Machinery, 2023: 76-88.

[59] FARRELLY G, CHESSER M, RANASINGHE D C. Ember-IO: effective firmware fuzzing with model-free memory mapped IO [C]// Proceedings of the 2023 ACM Asia Conference on Com-

puter and Communications Security. New York: Association for Computing Machinery, 2023: 401-414.

[60] CHESSER M, NEPAL S, RANASINGHE D C. MULTIFUZZ: A Multi-Stream Fuzzer For Testing Monolithic Firmware [C]// 33rd USENIX Security Symposium. Berkeley: USENIX Association, 2024: 5359-5376.

[61] ZHOU W, ZHANG L, GUAN L, et al. What your firmware tells you is not how you should emulate it: A specification-guided approach for firmware emulation [C]// Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2022: 3269-3283.

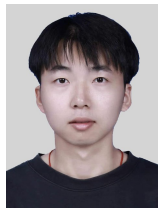
[62] GUI Z, SHU H, KANG F, et al. Firmcorn: vulnerability-oriented fuzzing of IOT firmware via optimized virtual execution [J]. IEEE Access, 2020, 8: 29826-29841.

[63] KIM H, OZMEN M O, BIANCHI A, et al. PGFUZZ: policy-guided fuzzing vehicles [C]// Network and Distributed Systems Security Symposium. San Diego: ISOC, 2021: 1-15.

[64] KIM J, YU J, KIM H, et al. FIRM-COV: high-coverage greybox fuzzing for IoT firmware via optimized process emulation [J]. IEEE Access, 2021, 9: 101627-101642.

[65] HERNANDEZ G, MUENCH M, MAIER D, et al. FIRMWIRE: Transparent dynamic analysis for cellular baseband firmware [C]// Network and Distributed Systems Security Symposium. San Diego: ISOC, 2022: 1117-1134.

[66] ZHENG Y, LI Y, ZHANG C, et al. Efficient greybox fuzzing of applications in Linux-based IoT devices via enhanced user-mode emulation [C]// Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: Association for Computing Machinery, 2022: 417-428.



SUN Qiming, born in 2000, postgraduate. His main research interest is fuzzing test.



HOU Gang, born in 1982, Ph. D, associate professor, is a member of CCF (No. 33349M). His main research interests include fuzzy testing and trusted software.

(责任编辑:喻藜)