计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240400161

基于配置引导的实时 Linux 内核靶向模糊测试

施鹤远[1,3]　陈世俊[2,3]　张　强[4]　沈煜恒[5]　姜　宇[5]　施荣华[1]
1 中南大学电子信息学院　长沙 410004
2 中南大学计算机学院　长沙 410012
3 网络空间安全态势感知与评估安徽省重点实验室　合肥 230037
4 湖南大学信息科学与工程学院　长沙 410082
5 清华大学软件学院　北京 100084
(hey.shi@foxmail.com)

摘　要　实时 Linux 在各类实时性需求业务场景中得到广泛应用,保障实时 Linux 内核安全稳定至关重要。当前主流的覆盖率导向内核模糊测试技术对于特定待测代码定位存在局限,导致内核模糊测试对实时特性相关代码的针对性测试能力不足。针对此问题,文中提出了一种基于配置引导的实时 Linux 内核靶向模糊测试方法。该方法首先结合内核配置选项构建内核文件树,识别实时特性相关内核代码;然后基于实时 Linux 内核函数间调用关系和基本块地址,构建实时特性相关待测靶点;最后利用基于权重的种子调度策略提升内核模糊测试针对性测试效果。该方法在 4 个版本的实时 Linux 内核的测试任务中,发现了 58 个实时特性相关的内核缺陷。与通用覆盖率导向内核模糊测试工具 Syzkaller 相比,该方法对于实时特性相关代码的基本块覆盖数量提升 17.06%,发现实时特性相关漏洞数量提升 65.39%。实验结果表明,该方法能显著提高内核模糊测试的实时特性相关代码覆盖能力与针对性测试能力。

关键词:实时 Linux;模糊测试;异常检测;内核配置
中图分类号　TP311

Configuration-guided Directed Kernel Fuzzing for Real-time Linux

SHI Heyuan[1,3],CHEN Shijun[2,3],ZHANG Qiang[4],SHEN Yuheng[5],JIANG Yu[5] and SHI Ronghua[1]
1 School of Electronic Information,Central South University,Changsha 410004,China
2 School of Computer Science and Engineering,Central South University,Changsha 410012,China
3 Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation,Hefei 230037,China
4 College of Computer Science and Electronic Engineering,Hunan University,Changsha 410082,China
5 School of Software,Tsinghua University,Beijing 100084,China

Abstract　The real-time Linux,due to its real-time characteristics,has been widely applied in various high-precision scenarios,which underscores the importance of its own security and reliability. However,the current methods for locating code sections related to real-time are limited,resulting in coverage-oriented kernel fuzzing tools,such as Syzkaller,lacking the ability to test this code comprehensively and thoroughly. To address this issue,this paper proposes a configuration-guided targeted fuzzing approach for the real-time Linux kernel. Our approach first constructs a kernel file tree by combining kernel configuration options,identifying real-time feature code,and building test targets. Next,it leverages the inter-function call relationships and basic block addresses within the real-time Linux kernel to define specific testing targets for real-time features. Finally,it utilizes a weight-based seed scheduling strategy to enhance the efficiency of directed testing in kernel fuzzing. In testing tasks across four versions of real-time Linux kernels,the proposed method identifies 58 kernel defects related to real-time features. Compared to general coverage-guided kernel fuzz testing method Syzkaller,our approach achieves a 17.06% increase in the basic block coverage of real-time feature code and a 65.39% improvement in the detection of vulnerabilities related to real-time features. Experimental results demonstrate that this method significantly enhances the capabilities of kernel fuzz testing tools in terms of coverage of real-time feature related code and directed testing ability.

Keywords　Real-time Linux,Fuzz testing,Anomaly detect,Kernel configuration

---

基金项目:国家自然科学基金(62202500);国家重点研发计划(2022YFB3104003);湖南省自然科学基金(2023JJ40772);网络空间安全态势感知与评估安徽省重点实验室开放课题(CSSAE-2023-010);中南大学高性能计算中心
This work was supported by the National Natural Science Foundation of China(62202500),National Key Research and Development Program of China(2022YFB3104003),Hunan Provincial Natural Science Foundation(2023JJ40772),Fund of Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation(CSSAE-2023-010) and High Performance Computing Center of Central South University.
通信作者:沈煜恒(shenyh20@mails.tsinghua.edu.cn)

1 引言

随着工业 4.0 的推进以及物联网技术的普及和不断发展，现在工业生产中对实时操作系统的需求持续增长[1]。实时 Linux(Real-time Linux，简称 RT-Linux)是 Linux 社区维护的一个提供实时能力的完全抢占式操作系统。实时 Linux 通过修改主线 Linux 的内核编译配置，可以将传统 Linux 使能成具备实时抢占等特性的 RT-Linux，为上层的用户态程序提供实时功能。目前，RT-Linux 凭借其高效的任务处理和调度能力，被广泛应用于各类对任务执行时间和响应时间有着极其严格要求的应用场景[2]，如航空航天、自动驾驶系统、工厂自动化等领域。由于这类领域对时间精度有着严格要求，RT-Linux 的实时特性需要有效手段保障代码的质量与安全。

针对 RT-Linux 安全性的测试方法有许多，模糊测试是质量保障的有效手段之一[3]。模糊测试运用遗传算法的思想，生成大量随机的测试用例输入至待测的程序中，通过观察待测程序各类行为以检测潜在的异常情况进而发现缺陷。模糊测试领域诞生了许多有价值的工作，适用于各个程序场景，如网络协议[4]、数据库[5]、云原生组件[6]和大语言模型[7]等。其中，Syzkaller 是操作系统内核安全方向最著名的工具[8]。Syzkaller 是一款基于覆盖率引导的内核模糊测试工具，它利用自定义的结构化语言，生成大量符合内核参数规范的系统调用序列。Syzkaller 通过自动化构建并执行大量随机生成的测试用例，可以监控内核的各类异常的内存操作和并发缺陷。一旦检测到新的异常行为或新的代码覆盖，Syzkaller 便将相应的测试用例保存到语料库中进行更多的突变和执行，进而提高后续生成测试用例的质量，发现操作系统内核中的潜在漏洞。

目前，许多研究基于 Syzkaller 并做出一定改进，以提升对 Linux 内核的测试效率。例如，有研究提出为模糊测试提供高质量的初始语料库。Moonshine[9]提出了蒸馏算法，通过精化大量真实的内核系统调用序列，生成高质量的测试用例；Rtkaller[10]为了对 RT-Linux 进行高效测试，提出使用任务作为测试输入，使用并行执行策略触发内核调度行为；KSG[11]对多个子模块做静态分析，提出一种系统调用规范自动生成方法。Tardis[12]设计跨框架的覆盖率收集机制，实现不同嵌入式操作系统的内核测试。然而，当前的模糊测试工具在面对 RT-Linux 内核针对性测试需求时面临以下挑战。一方面，现有工作以全局内核代码覆盖率为导向，将最大化内核全局代码覆盖率作为测试目标，无法满足针对性模糊测试需求，需要提出面向实时特性的针对性内核模糊测试机制。另一方面，准确定位实时特性代码相关位置作为测试靶点并非易事。由于内核实时特性通过内核配置使能，配置对内核代码的编译行为产生直接影响。然而，内核代码中存在上千个配置选项，导致研究人员无法通过变更内核配置文件精准激活测试目标。若未正确启用测试目标的配置选项，测试结果可能不完整；而盲目启用大量配置选项则会造成测试资源的浪费。因此，现有测试需要一种能准确定位实时特性相关代码的方法。

综上所述，为了保障 RT-Linux 实时特性质量和代码的安全性，本文提出了一种基于配置文件引导的 RT-Linux 靶向模糊测试方法。首先，为了精准定位 RT-Linux 实时特性的影响范围，本文通过提取内核配置文件完整的配置选项，构建内核文件树，遍历文件树选取所有与实时特性相关的配置选项所在代码，从而达到精准构建测试靶点的效果。进一步地，为了聚焦测试实时特性靶点，本文结合静态分析等技术抽取程序控制流，提出一种基于权重的种子调度策略，以改进 Syzkaller 测试，实现靶向模糊测试的效果。通过本文测试方法，在 4 个不同版本 RT-Linux 上总共发现实时特性相关缺陷 58 个，实时漏洞相关性比 Syzkaller 提升 65.39%，实时模块覆盖数量提升 17.06%。实验结果表明，本文方法能显著提高内核模糊测试的实时特性相关代码覆盖能力与针对性测试能力。

2 研究背景与知识

2.1 实时 Linux

操作系统是计算机中最庞大、最重要也是最复杂的软件之一，其安全性的保障一直是热点研究问题。内核是操作系统中最基础也最复杂的核心组件，承担着硬件资源管理、进程调度、文件系统管理、安全控制和外部设备交互等多项任务，其关键性不言而喻。随着操作系统的不断发展，实时操作系统成为研究和应用领域中备受瞩目的一部分。实时操作系统的最显著特征是强调实时性，专注于处理实时数据和任务。这对于需要及时响应和高度可靠性的应用程序至关重要，尤其是在高精度场景下，如嵌入式系统、通信系统、工业自动化和航空航天领域。

主流的实时操作系统[13]包括 RT-Linux，RT-Thread，FreeRTOS 和 ThreadX 等。其中，RT-Linux 是 Linux 的衍生版本，通过重新实现一个小的实时内核，在保留 Linux 操作系统基本功能的同时实现实时特性集成。Preempt-RT[14]是一个 Linux 内核实时化的协作项目，该项目实现了包括内核实时调度、实时调度器、中断处理机制针对性优化以及高精度计时器等实时特性在内的实时抢占代码补丁集合 RT-Preempt 于 Linux 中。RT-Linux 具有专业的研发团队和社区维护，被广泛应用于各个领域。当前，RT-Linux 的部分功能已经合入到 Linux 主线代码中。本文聚焦 Preempt-RT 相关实时特性展开针对性模糊测试。

2.2 模糊测试

近年来针对程序安全研究涌现了许多测试方法，有一项非常具代表性且取得了实际成效的技术是模糊测试[15]。模糊测试是一种有效的漏洞挖掘技术，其核心思想是通过自动或半自动方式生成测试用例，然后将这些测试用例输入到目标程序中，监测程序是否出现异常行为[16]。研究人员通过分析异常行为，可以发现程序中存在的潜在缺陷。模糊测试可以发现多种类型的漏洞，包括内存错误和逻辑错误等。AFL[17]是一款基于覆盖率引导的开源模糊测试工具，被广泛运用于各种应用系统的安全性评估，在安全领域具有里程碑意义。

靶向模糊测试是一种衍生的模糊测试方法，首次被 AFLGo[18]提出。与传统的基于代码覆盖率引导的模糊测试不同，靶向模糊测试将更多的时间花销在如何让测试用例覆盖特定测试模块，强调不在不相关的部分浪费资源。因此靶向模糊测试的使用场景非常广泛，例如代码补丁。开发人员会通过提交代码补丁来修补代码部分缺陷，但可能该补丁修补漏洞并不完善，甚至可能因为这部分补丁产生其他漏洞。靶向模糊测试则可以深度测试该补丁以及该补丁相关联的代码模块，保障其安全性。针对补丁场景的靶向测试具体代表工作有 DeltaFuzz[19]，AFLChurn[20]和 SemFuzz[21]等。然而，

由于内核庞大的代码空间且内核程序间存在复杂的交互，目前的靶向测试研究工作均是灰盒的应用程序测试场景，在内核测试领域并不适用。

内核漏洞模糊测试与传统模糊测试不同，面临着复杂的系统结构、多样的模块功能以及不便于程序检测和调试等多种挑战[22]。近年来谷歌公司发布的一款开源模糊测试工具 Syzkaller 在内核安全领域具有卓越的代表性和适用性。Syzkaller 的工作流程如图 1 所示。Syzkaller 采用其特有的描述性语言生成规范的系统调用，随之将这些系统调用作为初始种子执行，并监控目标操作系统内核的行为，以检测潜在的漏洞和缺陷。开发人员在不断维护发展 Syzkaller 的同时，还创建了相关的开源社区 Syz-group 和漏洞报告网站 Syzbot[23]。Syzkaller 完善的开源生态为内核安全领域的研究和漏洞修复提供了强大的工具和平台，推动了内核软件的安全性不断提升。Syzkaller 已经被广泛应用于 Linux，Android 和 Windows 等操作系统内核[24]，目前已成功协助研究人员发现许多重要的内核漏洞，对提升操作系统的安全性产生了积极的影响。

然而，Syzkaller 测试 RT-Linux 的实时特性代码仍存在一定缺陷。RT-Linux 实时特性与其他模块功能代码有深度交互，很难从单独的文件或代码中准确定位实时功能代码。另一方面，随着 RT-Linux 不同版本的发布，实时特性所对应的程序代码经历了快速的更新迭代。不同 RT-Linux 版本间的代码差异相当显著，这导致测试目标难以确定。从测试工具自身来说，Syzkaller 以全局内核代码覆盖率为导向，将最大化内核全局代码覆盖率作为测试目标，使得测试过程会耗费大量时间和开销在其他功能模块，很难触发到特定模块的代码路径，最终导致测试效率低、对特定内核功能模块的针对性缺陷检测能力不足等问题。因此，准确定位 RT-Linux 实时特性相关靶点并对其靶向进行测试，是一项具有挑战性的任务。

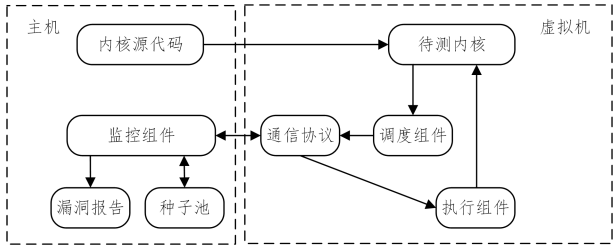


图 1　Syzkaller 工作流程

Fig. 1 Workflow of Syzkaller

3　基于配置引导的实时 Linux 内核靶向模糊测试

本章将详细描述所提出的基于内核配置文件分析的 RT-Linux 靶向模糊测试方法的设计流程。图 2 给出了测试总体框架。该方法主要分为 3 个部分：1）RT-Linux 实时特性的靶点构建；2）地址-权重表构建；3）靶向模糊测试。

首先，针对实时特性的靶点构建，在 RT-Linux 内核源代码使用 Clang 插桩编译之后，会对内核配置文件进行分析。配置文件分析主要是针对配置文件完整地提取所有配置选项，筛选所有实时特性选项的有关文件和代码，全面识别实时特性影响域，构建 RT-Linux 实时特性的靶点。其次，通过内核源代码静态分析，抽取内核函数调用图及其基本块地址，输入权重计算模型获取基本块的权重表征，构建基本块地址-权重表。最后，根据地址-权重表提出一种新的种子调度策略，用于引导 Syzkaller 实现针对实时特性的靶向模糊测试。

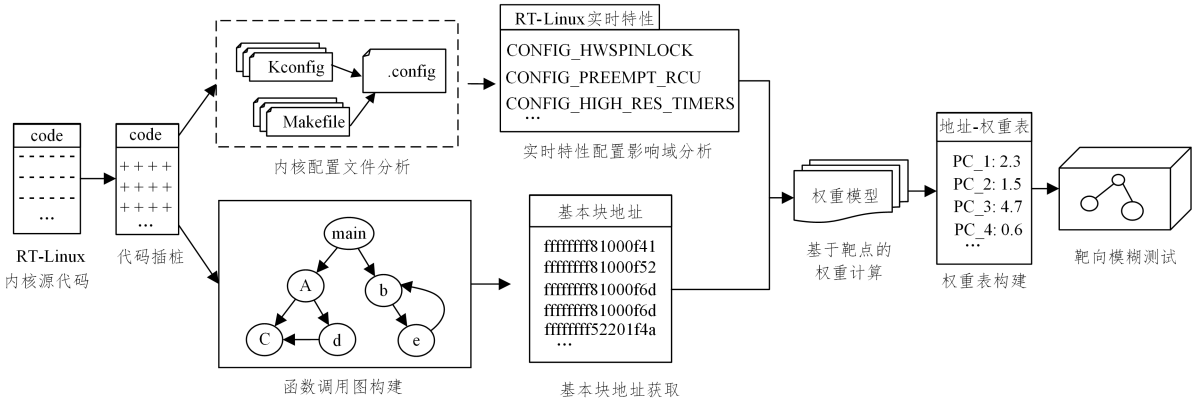


图 2　基于内核配置文件分析的 RT-Linux 靶向模糊测试框架

Fig. 2　RT-Linux directed fuzzing framework based on kernel configuration file analysis

3.1　RT-Linux 实时特性的靶点构建

靶点构建的目的是明确 RT-Linux 实时特性相关代码，作为针对性模糊测试过程的测试靶点输入。在靶点构建过程中，不仅要确保没有过多实时特性无关代码位置，还需确保不遗漏实时性代码相关位置，以平衡测试针对性和全面性。图 3 给出了实时特性靶点构建的完整流程。

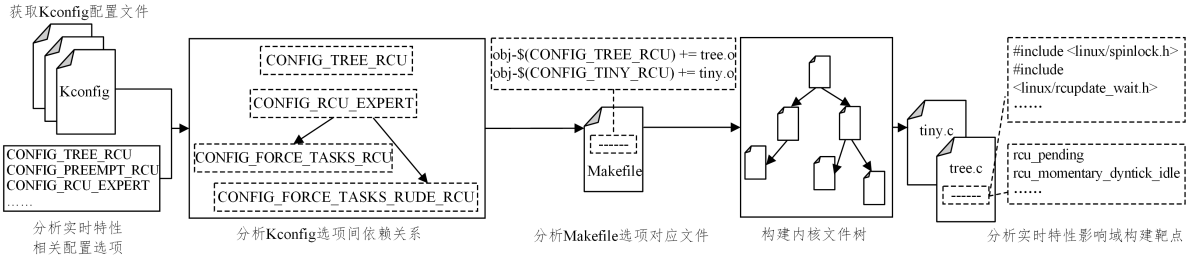


图 3　RT-Linux 实时特性靶点构建流程

Fig. 3　Workflow of constructing real-time feature targets in RT-Linux

3.1.1 内核配置文件分析

内核配置相关的文件包括 Kconfig, Makefile 和. config,这三者在 Linux 开发中具有密切关系,构成了内核的配置、构建和编译过程的基础。内核各级目录结构下一般会存 Kconfig 和 Makefile 两个文件。其中,Kconfig 是内核配置文件的核心,用于定义内核配置选项,以指定内核的行为、功能、驱动程序支持和其他特性,分布在各级目录下的 Kconfig 配置文件相当于一个分布式的内核配置数据库;Makefile 文件是内核的编译构建脚本,基于 Kconfig 的指导调用生成各个模块文件,构建整个内核。. config 文件是用于编译内核时的参考文件,它包含了所有 Kconfig 实际编译至内核的配置选项设置,存在于内核源代码的顶层目录。其中,. config 配置文件中的配置选项设置和内核功能存在密切关系,用户主要通过查看或修改. config 中的配置选项实现与内核交互。

Kconfig 中配置选项绝大多数为布尔类型和三态类型,并且相互间有依赖关系。选项设置为'y'时,代表该选项关联的文件函数和代码将会编译进内核;选项设置为'm'时,代表相应的文件函数和代码会编译为模块并在需要时加载;选项设置为'n'时,代表该选项相关代码将不会被编译至内核。然而,. config 文件中展示的配置选项并不完整。这是因为当 Kconfig 的某个父项被设置为'n'时,其关联的子项会被关闭,导致内核. config 文件缺少部分子配置选项。如图 4 所示,Kconfig 将 CONFIG_RCU_EXPERT 选项默认设置为'*n*',在. config 中显示为"# CONFIG_RCU_EXPERT is not set",而与其关联的子选项并未在. config 中展现。

为了抽取内核配置文件的完整配置选项,基于内核源代码目录建立各层子目录之间的连接,获取所有的 Kconfig 配置文件。其次,通过解析 Kconfig 配置文件中的关系规则,分析配置选项之间的依赖关系,识别出所有的配置选项。同时,通过分析 Makefile 文件获取到每个配置选项对应的配置相关文件,结合选项之间的依赖关系构建内核文件树。最后,结合. config 文件统计搜集到的配置选项,获取内核全部的配置选项。

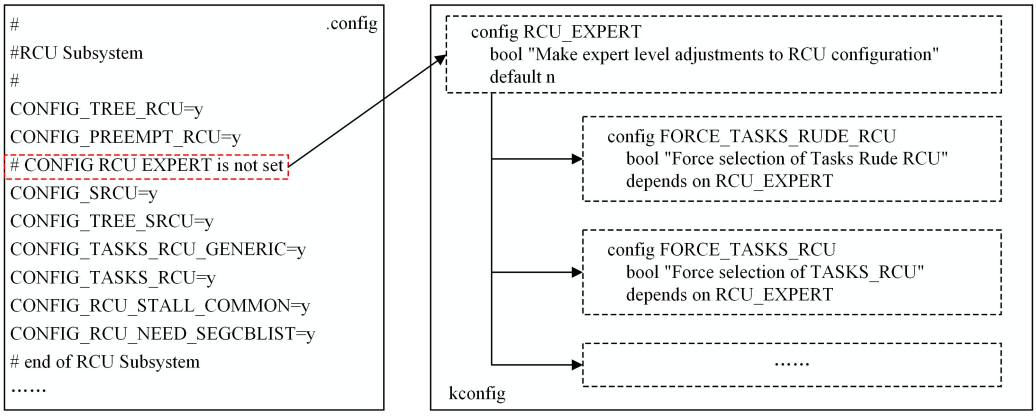


图 4 config 文件未展示 CONFIG_RCU_EXPERT 选项

Fig. 4 config file does not display the suboption of CONFIG_RCU_EXPERT

3.1.2 实时特性影响域分析

RT-Linux 内核代码会引入新的特性、修复漏洞或者改变原有的行为,各个子模块间相互作用实时特性本身的文件和代码行难以完全表征实时特性对于内核其他功能特性的影响,单纯使用实时特性相关文件作为测试靶点会导致测试针对性和完备性不足。因此,需要明确实时特性在内核代码中的影响范围。

为确保全面且准确地构建实时特性的靶点,实时特性影响域分析主要基于 RT-Linux 官方社区发布的实时模块,包括高精度计数器、RCU 同步机制、实时互斥、本地锁等特性。这些实时特性通过开启. config 配置文件中相应的配置选项,编译至内核中发挥作用。首先,根据 RT-Linux 官方发布的实时特性模块初步获取. config 配置文件中的部分实时特性配置选项。其次,在对 Kconfig 文件语法分析时着重关注实时特性的依赖关系,通过 Makefile 标记部分实时特性功能文件。在基于配置文件分析获取完整的内核配置选项后,过滤获取所有与实时特性有关的配置子选项。随后采用广度优先搜索算法遍历内核文件树,标记所有与实时特性有关文件与其子功能文件。这样获取到的实时配置选项和相关文件是无缺漏的,保证了实时特性靶点构筑的完备性。最后,对所有文件进行代码静态分析,标记所有与实时特性相关的函数、变量、代码行作为靶点。通过以上方法可以保证标记代码为实时特性功能或与之交互的子模块,保证靶点构建的有效性和完整性。

3.2 权重-地址表构建

3.2.1 内核代码静态分析

静态分析是一种在不运行程序代码,而使用语义、语法和控制流分析等技术对代码进行审查和评估的方法。通过静态分析技术,可以构建函数调用图和获取基本块地址,帮助研究人员深入了解代码结构和行为。然而,内核作为一个庞大且复杂的代码库,其代码的规格和复杂性远远超过了人工分析的范围。因此,构建其函数调用图以及获取基本块地址通常需要借助特定的工具和技术。

在内核编译阶段,使用 LLVM 项目中的 Clang 编译器,通过文件目录结构来提取所有内核文件的编译命令,并为每个文件生成中间表示形式。中间表示形式包含了每个文件的控制流信息,以及表示该文件的所有函数调用关系。接着,利用 LLVM-link 工具以自上而下的方式逐步链接所有的中间表示形式,从而构建出完整而统一的内核控制流。最后,通过 LLVM-opt 工具对这一控制流进行处理,生成函数调用图。通过函数调用图,可以分析实时特性代码内部和其他模块的调用关系,保证测试效率和全面性。

在内核模糊测试中，内核代码覆盖率是通过覆盖的基本块地址数量作为量化指标。objdump 和 addr2line 是静态分析中重要的工具，通常用于分析可执行文件和调试信息。使用 objdump 解析内核镜像文件，获取内核的所有基本块地址，再使用 addr2line 解析所有基本块地址映射的代码、所在文件行数和函数。基于基本块地址和关联代码信息，可以筛选出与实时特性靶点相关的基本块地址和函数信息。

3.2.2 基本块地址权重

为保证模糊测试有效聚焦于实时特性靶点，基于内核函数调用图和基本块地址关联信息构建了权重计算模型。计算模型根据关联信息权重大小建立模糊测试种子调度队列，优先调度权重大的种子序列，帮助和指导模糊测试种子调度，提高测试的针对性和测试效率。

本文定义内核内任意基本块地址为 m，该基本块所在函数为 f，目标靶点所有基本块相关联的函数集合为 T_f，可以计算基本块权重 $W_{\text{block}}(m)$。其计算式如下：

$$W_{\text{block}}(m)=\begin{cases}0, & \text{if } R(f,T_f)=\varnothing\\ C*\exp\left\{-\dfrac{\sum\limits_{t_f\in R(f,t_f)} d(f,t_f)}{Num[R(f,F_f)]}\right\}, & \\ \text{otherwise} & \end{cases} \tag{1}$$

其中，C 为常数，$R(f,T_f)$ 表示函数 f 可直接或间接调用目标靶点函数的集合，$Num[R(f,T_f)]$ 表示函数 f 可直接或间接调用的目标靶点函数数量，t_f 表示任意靶点函数。$d(f,t_f)$ 基于函数调用图计算可得，表示函数 f 调用目标靶点函数 t_f 所需的次数。通过该计算式，获取所有基本块地址对应的权重，构建地址-权重表来指导测试。

3.3 靶向模糊测试

在 Syzkaller 测试过程中，每个测试用例执行后都会作为新的种子送回种子池，并统计其覆盖信息。而本文提出的测试方法将在种子调度策略中进行改进，达到靶向模糊测试的效果。

Syzkaller 会根据测试工具自定义的模板生成初始种子。靶向模糊测试的完整流程如图 5 所示。在模糊测试初始化过程中，初始种子随机输入种子队列中，随后队列按照先进先出的逻辑输出种子用于种子变异，生成可执行的测试用例。测试用例执行内核程序时，系统实时监测内核程序，判断程序内是否产生异常，记录内核漏洞并统计内核覆盖信息。测试用例执行完成后将会作为新种子输入权重计算模型，结合地址-权重表计算种子的权重。在新一轮测试开始时，所有种子则不再是随机挑选，而是依据种子能量优先级依次进入种子队列。测试中的种子筛选阶段，除开第一轮为随机选择，其他轮次均为依据种子能量优先级进行筛选。

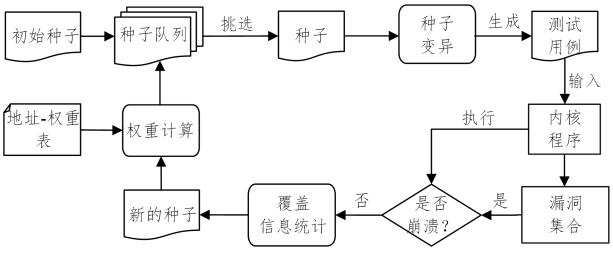


图 5 内核靶向模糊测试流程

Fig. 5 Workflow of directed kernel fuzzing

3.3.1 种子权重

测试用例执行完成作为新种子时，接下来将计算种子的权重。内核模糊测试的种子形式是系统调用序列，而每条系统调用均会覆盖大量基本块地址。系统调用覆盖的所有基本块地址权重累加，构成该系统调用的权重。系统调用的权重计算式如下：

$$W_{\text{syscall}}(s)=\sum_{m\in cor(s)} W_{\text{block}}(m) \tag{2}$$

其中，S 表示任意系统调用，$W_{\text{syscall}}(S)$ 表示系统调用的权重，$cor(S)$ 表示系统调用 S 覆盖所有基本块的集合。得到所有系统调用的权重后，可用于计算各个种子的权重。种子权重的计算式如下：

$$W_{\text{seed}}(t)=\sum_{s\in t} W_{\text{syscall}}(s) \tag{3}$$

其中，t 表示任意种子，$W_{\text{seed}}(t)$ 表示种子的权重。

3.3.2 种子调度

种子调度决定了测试用例参与模糊测试的优先级、次数与变异策略。根据种子权重和执行次数来判断当前种子的能量，选取测试用例。种子的权重越高，该种子有更大概率参与模糊测试变异执行过程。

定义种子 t 的能量为 $E(t)$，种子的选取次数为 $N(t)$，则有如下计算式：

$$E(t)=\frac{W_{\text{seed}}(t)}{N(t)+1} \tag{4}$$

种子按照能量高低依次排列在队列 Q 中。按照队列先进先出的逻辑，每次测试选取能量最高的种子 T：

$$T=\arg\max_{(t,E(t))\in Q} E(t) \tag{5}$$

种子的选取次数也会更新，$N(t)=N(t)+1$。通过此调度策略，种子选取充分考虑了种子的权重和选取次数，基于权重能够保证测试更聚焦于待测目标的覆盖，深度测试内核实时特性。同时，选取次数的限制能够避免测试陷入局部最优解，让其他种子有机会挖掘到不同类型的漏洞。

4 实验结果

4.1 实验平台设置以及数据描述

本文测试方法通过 Golang 和 Python 两种编程语言实现，在原有内核模糊测试工具 Syzkaller 上作出改进。相比其他相关拓展工作，Syzkaller 得益于谷歌公司开发人员不断发展和维护，是内核模糊测试领域最具影响力的测试工具，具有很强的时效性和适用性。同时，Syzkaller 现有 filter 功能支持靶向测试内核代码。因此，本次实验选择对比基线工具为最新版 Syzkaller。

本次实验目标是验证本文方法聚焦测试实时特性的有效性，因此实验选择测试目标必须为实时特性相关模块。选择 io_uring 模块作为模糊测试的靶点，因为 io_uring 在实时操作中具有重要意义。io_uring 模块是 Linux 内核于 2019 年引入的高性能异步 I/O 框架，通过共享内存和环形队列与用户态进程高效交互。io_uring 设计用于高效的多操作处理，主要的系统调用为 io_uring_submit()，使其在实时操作中具有紧密联系。定义与 io_uring 实时特性相关的漏洞为靶点相关漏洞。如图 6 所示，当挖掘到的漏洞与 io_uring 关联子函数相关，则归类为靶点相关漏洞。将靶点相关漏洞和其他无关漏洞统称为内核全局漏洞，即将所有内核相关缺陷均纳入内

核全局漏洞范畴。

本次实验选择 4 个不同版本的 RT-Linux 作为测试对象，包括 6.1.1，6.2.1，6.3.3，6.4.0。采用 Clang 针对目标内核 io_uring 模块代码进行插桩。启用内核 KASAN（Kernel Address Sanitizer）和 KCSAN（Kernel Concurrency Sanitizer）组件进行错误检测。实验评估指标主要从漏洞挖掘能力和实时特性覆盖信息两方面进行，比较本文方法与基线方法在实时特性相关代码的基本块覆盖数量、实时特性相关漏洞发现数量等方面的测试结果。本文所有的实验都在相同硬件上运行 24 小时，每个实验重复两次。在每个 RT-Linux 版本上，设置两组对照实验。每组实验开启两个虚拟机，每个虚拟机配置 2 个测试线程与 4 GB 内存。测试采用 KCOV 收集覆盖信息。实验运行环境使用 AMD Ryzen 9 7945HX 处理器，该处理器配备 16 核心和 32 线程，搭载 32 GB 运行内存，运行在 Ubuntu 22.04 操作系统下。

```
Sample crash report:
......
Call Trace:
 __dump_stack lib/dump_stack.c:79 [inline]
 dump_stack+0xfa/0x151 lib/dump_stack.c:120
 __kasan_report mm/kasan/report.c:403 [inline]
 kasan_report.cold+0x5f/0xd8 mm/kasan/report.c:416
 check_region_inline mm/kasan/generic.c:180 [inline]
 kasan_check_range+0x13d/0x180 mm/kasan/generic.c:186
 instrument_atomic_read_write include/linux/instrumented.h:101 [inline]
 atomic_inc include/asm-generic/atomic-instrumented.h:240 [inline]
 io_uring_cancel_sqpoll+0x2c7/0x450 fs/io_uring.c:8871
 io_sq_thread+0x1109/0x1ae0 fs/io_uring.c:6782
 ret_from_fork+0x1f/0x30 arch/x86/entry/entry_64.S:294
......
```

图 6 实时特性相关漏洞示例

Fig. 6 Example of vulnerability related to real-time features

4.2 实时特性覆盖信息

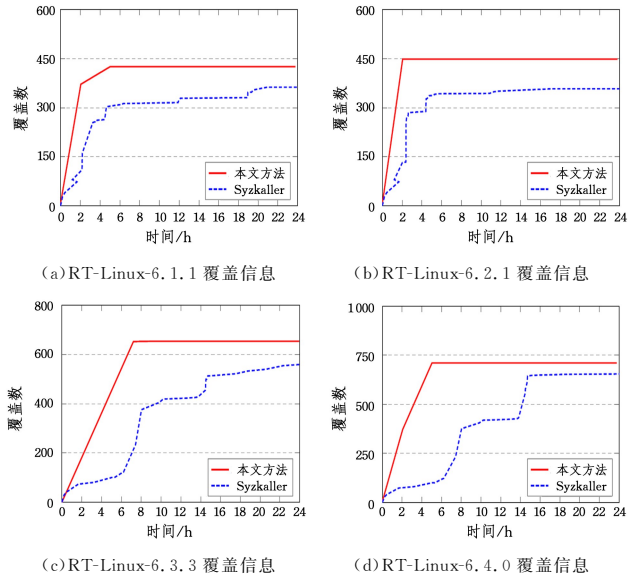
为了确定本文测试方法的有效性，记录 24 h 内本文测试方法和 Syzkaller 所覆盖的 io_uring 实时特性基本块数量。测试结果如表 1 所列。Syzkaller 平均覆盖实时特性基本块数量为 484.25，本文方法平均覆盖实时特性基本块数量为 560.25。对比分析两者覆盖信息，本文方法的平均基本块覆盖数提升 17.06%。

表 1 实时特性基本块覆盖数对比

Table 1 Comparison of basic block coverage numbers of real-time feature

RT-Linux 版本	6.1.1	6.2.1	6.3.3	6.4.0	平均
Syzkaller	363.00	358.00	560.00	656.00	484.25
本文方法	426.00	449.50	654.50	711.00	560.25
指标提升	17.36	25.56	16.88	8.38	17.06

为了进一步说明本文的覆盖率增长情况，绘制了覆盖率的增长曲线。由图 7 表明，本文方法覆盖速率在 2h 之前快速增长，在 4～8h 达到饱和。如此快速的增长是因为只针对实时特性统计基本块覆盖数量。聚焦靶点测试 2h 后覆盖率很容易达到饱和，反映了本文测试方法优先覆盖实时特性相关基本块。本文测试方法基于内核实时配置入手，利用静态分析方法抽取靶点目标基本块，全方位构建内核实时特性靶点，优先调度直接或间接调用目标基本块的测试用例，聚焦覆盖靶点基本块，与传统覆盖率引导的测试工具有很大不同。而由于 Syzkaller 是基于覆盖率引导的测试工具，不能全方面测试实时特性，曲线增长则相对缓慢。因此，本文测试方法在针对实时特性的基本块覆盖速率和基本块覆盖数量上更具有效性。



(a) RT-Linux-6.1.1 覆盖信息 (b) RT-Linux-6.2.1 覆盖信息

(c) RT-Linux-6.3.3 覆盖信息 (d) RT-Linux-6.4.0 覆盖信息

图 7 本文方法与 Syzkaller 覆盖信息对比

Fig. 7 Comparison of coverage information between the proposed method and syzkaller

4.3 漏洞分析示例

本文测试方法基于 Syzkaller 增量实现，其缺陷分析继承 Syzkaller 原生机制。当 Syzkaller 触发漏洞时，会记录下触发漏洞的对应测试用例和对应的内核输出。同时，测试工具根据这些信息进行复现，基于其内核输出信息，分析具体类型和原因。

图 8 给出了本文测试工具所发现的漏洞示例，该示例指出 io_uring 子系统中的数据竞争问题主要发生在 io_uring_try_cancel_requests 函数中。该函数在执行请求取消操作时未能有效地进行同步保护，导致多个工作线程并发访问时出现竞争条件。

```
BUG: KCSAN: data-race in io_sq_thread / io_uring_try_cancel_requests

Call Trace:
io_uring_try_cancel_requests+0x1e8/0x570 io_uring/io_uring.c:3308
io_ring_exit_work+0x126/0x500 io_uring/io_uring.c:3069
process_one_work kernel/workqueue.c:3267 [inline]
process_scheduled_works+0x465/0x990 kernel/workqueue.c:3348
worker_thread+0x526/0x730 kernel/workqueue.c:3429
kthread+0x1d1/0x210 kernel/kthread.c:388 ret_from_fork+0x4b/0x60
arch/x86/kernel/process.c:147 ret_from_fork_asm+0x1a/0x30
arch/x86/entry/entry_64.S:244
......
```

图 8 漏洞分析示例

Fig. 8 Example of vulnerability

4.4 漏洞挖掘能力

为了确定本文测试方法的适用性，分别收集了本文测试方法和 Syzkaller 24 小时内挖掘到的所有漏洞。其中，Syzkaller 挖掘漏洞数量如表 2 所列，Syzkaller 可以平均发现 33.5 个内核漏洞，但是平均只有 4 个漏洞与实时特性相关，发现漏洞的实时特性关联比例只有 11.94%。本文测试

方法漏洞挖掘数量如表 3 所列，本文测试方法虽然平均只能发现 18.75 个内核漏洞，但是其中平均有 14.5 个内核漏洞与 io_uring 实时特性相关。对漏洞挖掘信息进行分析，发现 Syzkaller 在广泛挖掘内核漏洞中能做到优秀表现，但是很难挖掘到与实时特性靶点相关的漏洞。这是因为 Syzkaller 基于覆盖率引导广泛搜寻内核相关漏洞，而不是重点关注实时特性。而本文测试方法专注于实时特性缺陷的挖掘，与传统覆盖率引导测试不同。本文测试方法从配置文件入手，通过使能实时特性配置，基于配置引导全面构建实时特性靶点。其次，测试通过构建内核函数调用图，赋予实时特性相关基本块更高权重，帮助测试有效聚焦实时特性相关靶点。最后，测试着重调度与实时特性覆盖相关种子，生成测试用例执行测试，实现内核靶向模糊测试。最终，本文测试方法的漏洞挖掘相关性达到 77.33%，相比 Syzkaller 提升了 65.39%，有效聚焦测试实时特性 io_uring。

表 2　Syzkaller 漏洞挖掘数量

Table 2　Number of vulnerabilities discovered by Syzkaller

RT-Linux 版本	6.1.1	6.2.1	6.3.3	6.4.0	平均
靶点相关漏洞	4	6	2	4	4
内核全局漏洞	42	33	27	32	33.5

表 3　本文方法漏洞挖掘数量

Table 3　Number of vulnerabilities discovered by the proposed method

RT-Linux 版本	6.1.1	6.2.1	6.3.3	6.4.0	平均
靶点相关漏洞	12	21	14	11	14.5
内核全局漏洞	15	24	19	17	18.75

结束语　针对实时操作系统 RT-Linux 实时特性相关的代码段的代码逻辑越来越复杂，难以保障代码安全性的问题，本文提出了一种基于内核配置文件引导的 RT-Linux 靶向模糊测试方法。本文方法通过内核配置文件准确分析实时特性影响域，构建内核函数调用图，建立内核地址-权重表，指导种子筛选调度，实现了针对实时特性的靶向模糊测试，能有效保障实时特性的可信与安全。本文通过对比 Syzkaller 测试工具在 4 个不同版本的 RT-Linux 实验结果，发现该方法显著提高靶点覆盖速率，增加靶点覆盖数量 17.06%。此外，本文方法总共挖掘到 58 个实时特性相关缺陷，挖掘漏洞相关性提升 65.39%。

参 考 文 献

[1] ZIKRIA Y B, KIM S W, HAHM O, et al. Internet of Things (IoT) operating systems management: Opportunities, challenges, and solution[J]. Sensors, 2019, 19(8): 1793.

[2] WANG C, YANG F, WANG H, et al. Improving real time performance of Linux System using RT-Linux[J]. Journal of Physics: Conference Series, 2019, 1237(5): 052017.

[3] ZHU X, WEN S, CAMTEPE S, et al. Fuzzing: a survey for roadmap[J]. ACM Computing Surveys (CSUR), 2022, 54(11s): 1-36.

[4] LUO Z, ZUO F, SHEN Y, et al. ICS protocol fuzzing: Coverage guided packet crack and generation[C]// 2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE, 2020: 1-6.

[5] FU J, LIANG J, WU Z, et al. Griffin: Grammar-free DBMS fuzzing[C]// Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. 2022: 1-12.

[6] WANG R, WANG Q, HU Y, et al. Industry practice of configuration auto-tuning for cloud applications and services[C]// Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2022: 1555-1565.

[7] ZHANG C, BAI M, ZHENG Y, et al. Understanding Large Language Model Based Fuzz Driver Generation[J]. arXiv: 2307.12469, 2023.

[8] DMITRYV Y, ANDREY K. Syzkaller is an unsupervised coverage-guided kernel fuzzer[OL]. https://github.com/google/syzkaller. 2015

[9] PAILOORS, ADAYA, JANAS. {MoonShine}: Optimizing {OS} fuzzer seed selection with trace distillation[C]// 27th USENIX Security Symposium (USENIX Security 18). 2018: 729-743.

[10] SHEN Y, SUN H, JIANG Y, et al. Rtkaller: State-aware task generation for RTOS fuzzing[J]. ACM Transactions on Embedded Computing Systems (TECS), 2021, 20(5s): 1-22.

[11] SHEN Y, XU Y, SUN H, et al. Tardis: Coverage-Guided Embedded Operating System Fuzzing[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 41(11): 4563-4574.

[12] SUN H, SHEN Y, LIU J, et al. {KSG}: Augmenting Kernel Fuzzing with System Call Specification Generation[C]// 2022 USENIX Annual Technical Conference (USENIX ATC 22). 2022: 351-366.

[13] HAMBARDE P, VARMA R, JHA S. The survey of real time operating system: RTOS[C]// 2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies. IEEE, 2014: 34-39.

[14] REGHENZANI F, MASSARI G, FORNACIARIW. The real-time linux kernel: A survey on preempt_rt[J]. ACM Computing Surveys (CSUR), 2019, 52(1): 1-36.

[15] SU W, FEI H. Survey of Coverage-guided Grey-box Fuzzing[J]. Journal of Information Security Research, 2022, 8(7): 643.

[16] BÖHME M, CADAR C, ROYCHOUDHURYA. Fuzzing: Challenges and reflections[J]. IEEE Software, 2020, 38(3): 79-86.

[17] Lcamtuf: American fuzzy lop[OL]. https://lcamtuf.coredump.cx/afl. 2013

[18] 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. 2017: 2329-2344.

[19] ZHANG J M, CUI Z Q, CHEN X, et al. DeltaFuzz: historical version information guided fuzz testing[J]. Journal of Computer Science and Technology, 2022, 37(1): 29-49.

[20] ZHU X, BÖHME M. Regression greybox fuzzing[C]// Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021: 2169-2182.

[21] YOU W, ZONG P, CHEN K, et al. Semfuzz: Semantics-based

automatic generation of proof-of-concept exploits[C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017:2139-2154.

[22] LI H, ZHANG C, YANG X, et al. Survey of OS Kernel Fuzzing [J]. Journal of Chinese Computer Systems, 2019, 40(9): 1994-1999.

[23] SHI H, WANG R, FU Y, et al. Industry practice of coverage-guided enterprise linux kernel fuzzing[C]// Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019: 986-995.

[24] SHI H, WANG G, FU Y, et al. Abaci-finder: Linux kernel crash classification through stack trace similarity learning [J]. Journal of Parallel and Distributed Computing, 2022, 168: 70-79.

SHI Heyuan, born in 1993, associate professor, is a member of CCF (No. T2400M). His main research interests include software safety and testing, and so on.

SHEN Yuheng, born in 1998, doctoral student. His main research interests include verification of software and operating system kernel fuzzing.