论文总结

背景综述

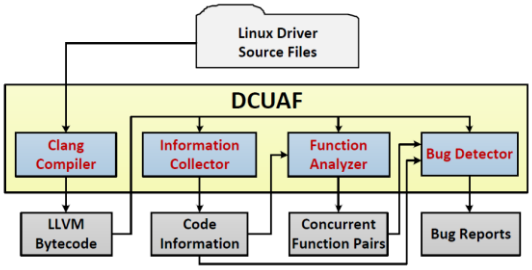
本篇论文是 2019 年 USENIX ATC 会议发表的一篇论文。会议的组织者是 VMware Research 的首席研究员 Dahlia Malkhi 和以色列理工学院的 Dan Tsafrir。我校的余华老师也参加了此次会议。项目主要由 Facebook， NSF（美国国家科学基金会）资助。IBM，ORACLE 公司也给予相应的支持，主要用于研究开发新的互联网技术，推动互联网的发展。过去几次主要会议探讨存储和文件系统、云计算和边缘计算。涵盖边缘计算的新范例、新挑战、技术和商业影响，以安全、可持续的方式实现端到端（边缘-云端-边缘）的人工智能系统，如何利用人工智能技术构建大规模分布式系统、云存储、量子计算、区块链、硬件加速网络系统的趋势与未来等议题。

作者简介

白家驹是清华大学的博士后，研究方向为系统软件可靠性、操作系统和程序分析，重点通过程序分析针对设备驱动程序进行故障检测。他还是 2019 年中国 Linux 内核开发者大会最佳演讲者。

Julia Lawall 是 Inria/LIP6 的 Wisper 组的 Inria 高级研究科学家,同时也参加了 IRILL。目前的研究重点是领域特定语言的设计和实现，主要针对操作系统中的问题。她还研究了部分计算、lambda 演算的最优约简和连续。

论文主旨



本文主要致力于高效率检测 Linux 设备驱动程序中的并发性 use-after-free bugs，并为此提出了一种名为 DCUAF 的静态检测方法。

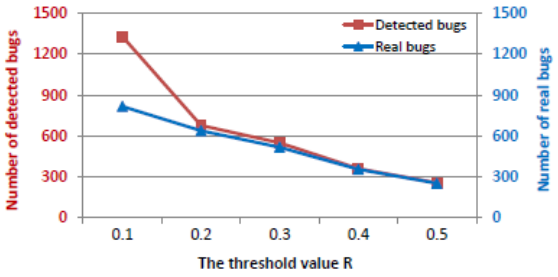
设备驱动中的 use-after-free bugs 可能会造成被黑客攻击或系统崩溃等严重后果，需要尽可能在出现错误之前检测出来。本方法的主要思路是先找出可能会并发执行的驱动函数对，然后对每一组函数对进行代码分析，从而检测出其中的 use-after-free bugs。作者首先引入了 Linux 驱动接口模块。Linux 驱动接口是 Linux 设备驱动程序与内核其他部分联系的枢纽，驱动程序中的其他驱动函数均由驱动接口调用。所以，对并发性驱动函数对的检测可以转换为对并发性驱动接口对的检测，这会大大减少工作量。而后，作者调查研究了 Linux 内核修复记录。他发现在 2016-2019 年的所有 use-after-free bug 记录中，由 42%是并发性的，并且只有很少部分是通过工具检测出来的。

Time	Commits	Drivers	Concurrency	Tool
2016 (Jan-Dec)	186	111	42 (38%)	26
2017 (Jan-Dec)	478	205	87 (42%)	49
2018 (Jan-Dec)	285	145	66 (46%)	52
Total	949	461	195 (42%)	127

本文的基本思想是首先提取并发函数对，即可以并发执行的驱动程序接口函数对，然后对这些函数对的源码进行比较分析，以检测并发性 use-after-free bugs。实现这一想法需要应对两个主要挑战: 1）提取并发函数对 2）代码分析的准确性和效率。为此，本文提出 Local-Global Strategy 来完成并发函数对的提取，提出 Summary-Based Lockset Analysis 来完成源代码的分析。

Local-Global Strategy 分为 Local 部分和 Global 部分。其中 Local 部分通过查找特定类型的函数接口对来缩小查找范围，这部分分为三步： 1）由于并发执行的函数对中双方大部分都会申请相同的锁，所以通过寻找所有申请了相同锁的驱动接口对来缩小并发函数对范围。2）第一步中得到的函数接口对中，如果是由同一祖先函数所调用的，那它们必不可能并发执行。所以第二步删去第一步得到的函数接口对中被相同祖先调用的函数接口对，进一步缩小范围。 3）删去重复的接口对。Global 部分通过 Local 部分得到的函数接口对得到分别包含每一对接口对的驱动函数对。对于每一对得到的函数接口对，统计包含他们的所有驱动函数对并分析这些驱动函数对的并发性。只有当其中并发性的函数对超过一定比例（驱动函数申请的锁不一定是为了防止该驱动接口并发，只有比例达到一定大小，才确定是为了该接口对申请的锁）是才认定该接口对是并发性驱动接口对，相应的驱动函数对是并发性驱动函数对。再进一步处理得到最终想要得到的并发性驱动函数对。

Summary-based lockset analysis 对前面得到的并发性驱动函数对进行分析，从而检测出 bug。首先，收集并发函数对中的函数名、源文件名和对变量的操作等信息，并存在数据库中。然后，分析数据库中的数据，如果函数对双方对相同变量进行操作，其中一个操作为释放内存操作，并且之前未统计过的话，就将其放入并发性 use-after-free bug 列表中。最终返回所有检测到的并发性 use-after-free bugs。



Description		3.14	4.19
Bug detection	Filter repeated	348	390
	Final detected (real / all)	526 / 559	640 / 679
	Double free (real / all)	82 / 89	117 / 132
	Interrupt handler (real / all)	25 / 25	23 / 23
Time usage		8m43s	10m15s

结果：

最终将 R 值定为 0.2。最终在 Linux 3.14 内核中检测到了 559 个 use-after-free bugs，其中 526 个是正确识别的，正确率约为 94%。在 Linux 4.19 内核中检测到了 679 个 use-after-free bugs，起中 640 个是正确识别的，正确率约为 94%。

相关工作

为了检测设备驱动程序中的并发问题，现有的许多方法都是基于动态分析或静态分析的:

动态分析。相关的动态分析方法是基于采样的或基于锁集的。 Data-Collider 是一种有效的基于采样的检测 Windows 内核中数据竞争的方法。它在运行时随机抽样内存访问。为了增加捕获对相同内存地址的并发访问的可能性，它将当前正在运行的线程延迟一小段时间，并使用硬件断点在延迟期间捕获任何二次访问。如果发生了第二次访问，并且至少有一次是写访问，那么就会检测到真正的数据竞争。 Eraser 是第一个基于锁集的检测数据竞争的方法。它利用二进制代码来执行对每个运行线程的共享变量访问的运行时监控，并通过在执行期间维护和检查共享变量的锁集来检测数据竞争。动态方法需要相关的硬件设备来实际运行测试的驱动程序，这在实践中可能很难获得。此外，由于并发执行的非确定性，它们可能会遗漏许多真正的并发 bug。

静态分析。大多数相关的静态分析方法是基于静态锁集分析。 RacerX 是一种著名的基

于静态锁集的方法，用于检测 OS 内核代码中的数据竞争和死锁。它使用过程间、流敏感和上下文敏感的分析来维护和检查代码路径中的锁集，并检测数据竞争和死锁。它还对报告的 bug 进行了排序。WHOOP 是一种有效的基于静态锁集的设备驱动程序数据竞争检测方法。它使用一个象征性的双锁分析，试图证明一个驱动 race-free。它还使用了一个健全的部分秩序减少，以加快 CORRAL（一个现有的并发 bug 检测器）。这些静态方法针对的是常见的并发性 bug，如数据竞争和原子性违规，它们通常有许多误报(例如， RacerX 上的工作报告的 false-positive rate 接近 50%)。它们不关注并发性 use-after-free bug。此外，他们假设所有的驱动接口功能都可以并发执行，或者依赖于手动引导来确定，这可能会带来很多误报或者需要大量的手动工作。与这些方法不同， DCUAF 针对的是并发性 use-after-free bugs，并使用本地-全局策略来精确和自动地从驱动程序源代码中提取并发函数对。