

DangZero: Efficient Use-After-Free Detection via Direct Page Table Access

Floris Gorter Koen Koning Herbert Bos Cristiano Giuffrida

2022 年 7 月 11 日

摘要

释放后使用 (use-after-free) 漏洞仍然难以检测和缓解, 因此成为一种常见的漏洞利用来源。现有的解决方案会产生很高的性能或内存开销, 需要专用硬件, 且只能保证可以保护不被利用而不能保证能检测漏洞。

在本文中, 我们提出了一种新的解决方案 DangZero, 当释放后使用漏洞出现时可以检测它们。DangZero 基于传统的页面保护和别名方案, 在此方案中, 对象在释放后将无法访问, 而随后的访问会被立即检测到。与之前使用基于别名检测的解决方案不同, DangZero 依靠在 ring0 级别中直接访问页表来提供更高效的实现。关键思路是, 通过让程序的分配器直接访问页表, 我们就能有效地管理和失效易受攻击的对象。为了安全地实现这一点, 我们采用了类似 unikernel 的设计、虚拟化提供了 ring-0 (访客模式) 访问、隔离以及与现有 Linux 程序的兼容性。此外, 我们会证明直接页表访问对于垃圾回收式的别名回收而言是一个高效的组成部分。这样做可以安全地重复使用已释放区域的能力, 并解决了困扰最先进的基于别名的解决方案的可扩展性问题。我们的实验结果证实, DangZero 能提供准确的检测保证, 而开销却大大低于同类最先进的解决方案 (例如, 在 Nginx 网络服务器等长时间运行的程序上, 饱和吞吐量降低了 18%)。

关键词: 内存安全; 释放后使用检测; 页面权限

1 引言

时间性内存错误仍然是计算机系统防范漏洞和利用的一个重要问题。在 CWE 最常见和影响最大的 25 个软件问题中, 释放后使用 (UAF) 漏洞排名第 7 位。此外, 微软报告称, UAF 漏洞是第二大最常见的漏洞根源, 并将继续成为利用的首选目标。抵御此类威胁的方法可分为提供即时检测或仅提供保护以防被利用。提供漏洞检测对于离线 (如测试) 和在线部署方案以及在错误分流中都很重要。遗憾的是, 现有的解决方案都存在问题。

保证 UAF 保护通常比即时检测更有效, 现有的保护系统试图通过各种技术将其对性能的影响降至最低: 类型安全的内存重用 [2, 36] (然而, 这只能维护类型安全)、引用计数 [34] (但不适用于任意的 C/C++ 程序)、一次性分配 [39] (但不能约束内存使用) 和垃圾回收式 (GC) 解决方案 [1, 17, 19, 26]。虽然 GC 式解决方案因其报告的效率而获得了越来越多的关注, 但最近的研究表明, GC 式技术的基本成本并不高, 通常隐藏在并发性和内存/计算能力的供

应背后 [12]。现有方案的缺点是，许多解决方案无法防范不依赖内存重用的漏洞利用 [2]，而大多数基于编译器的解决方案 ([1, 2, 17, 39] 除外) 无法处理未修改的二进制文件。最重要的是，这类解决方案都无法提供强大的 UAF 检测保证。

大多数聚焦于 UAF 检测的系统都依赖于编译器工具来跟踪被释放对象的指针并使其失效 [25, 33, 37, 42]。尽管进行了专门的优化 [37]，但此类解决方案仍会产生不小的性能开销。成本较低的解决方案依赖于特殊的硬件支持 [18, 44] 或对象 ID [5, 11, 13, 18, 21, 29] 来检测 UAF。同样，大多数基于编译器的解决方案无法处理未修改的二进制文件。

不过，文献 [15, 16] 中也介绍了二进制兼容的 UAF 检测系统。这些解决方案为每个内存分配创建一个新的虚拟页（称为别名），并将其映射到与原始对象相同的物理页上。因此，每个对象都会收到一个唯一的（未使用的）指针，而且对象（及其指针）可以通过撤销页面映射轻松地在释放时失效。遗憾的是，这种基于别名的解决方案依赖于内核的页面保护和别名，而且由于额外的系统调用和内核管理成本，会产生很高的开销。此外，最先进的解决方案 [15] 仍然存在由于虚拟内存地址空间耗尽导致的可扩展性问题。

在本文中，我们将介绍一种高效、可扩展、二进制兼容的 UAF 检测系统 DangZero。其主要思想是依靠 ring0（即通常只运行操作系统内核的最高权限级别）的直接页表访问，以更高效的方式实现基于别名的传统方案。从现代类 unikernel 设计 [24] 中得到启发，DangZero 依靠虚拟化扩展和特权后端，如内核模式 Linux (KML) [28] 来直接访问页表。这种策略允许我们在 ring0 客户模式下透明地运行并隔离任意用户空间程序，同时安全地让它们直接访问它们自己的（访客）页表。

2 相关工作

2.1 检测 UAF

2.1.1 静态分析方法

由于面向对象和过程编程提倡内存使用和空闲模式的分离，静态源代码分析面临着一些挑战：（1）如何解决程序路径随程序大小呈指数增长的问题，即如何高效地推断程序中每个指针的内存申请位置、使用位置和释放位置。（2）如何高效地执行指针别名分析，即两个或多个不同的指针可能指向同一个内存对象。（3）如何有效确定 UAF 漏洞的所有路径可行性，即指向同一内存对象的指针具有从内存申请位置到释放位置，最后到使用位置的路径。

由于这些挑战，只有少数通用工具 [14, 23, 32] 和一些特定工具 [41]、[40] 可用于检测程序中的 UAF。

有界模型检查。CBMC [23] 是一种位精确的有界模型检查工具，它将模型检查 [6] 与可满足性求解结合在一起。CBMC 首先将每个循环展开成固定边界，然后将程序翻译成注有 UAF 断言的程序。然后，它将所有可能的程序路径视为约束，并使用 SMT 求解器进行求解，以确定程序是否包含违反 UAF 约束的情况。尽管 CBMC 的精度很高，但由于约束求解的限制，它只能适用于小型程序 [38]。

指针分析。Tac [40] 结合指针分析和机器学习来检测 UAF 漏洞。它首先使用指针分析工具 SVF [35] 获取程序中的指针信息，然后建立有限状态机来确定程序中是否存在 UAF。Tac 使用支持向量机 (SVM) 来提高指针别名分析的有效性，以减少检测结果中的误报。虽然 Tac

可以扩展到大型程序，但由于机器学习的缺陷，它无法保证在真实世界的程序中有效减少误报。CRed [41] 基于需求驱动的指针分析和时空上下文缩减，可有效检测 C 程序中的 UAF。它首先使用 SVF 获取程序中的候选 UAF 对，然后通过多阶段分析减少误报，包括调用上下文敏感减少和路径敏感减少。为了扩展到大型程序，该方法在调用上下文深度和可扩展性之间进行了权衡，以减少需要分析的程序路径数量。因此，由于要在可扩展性和呼叫上下文深度之间进行权衡，CRed 不可避免地会遗漏一些 UAF。此外，由于 CRed 对数组访问别名的建模不健全，而且缺乏对链表的处理，因此即使没有减少调用上下文深度，它仍有可能漏掉真正的 UAF。

2.1.2 动态分析方法

动态二进制翻译。Purify [22]、Valgrind [30] 和 Dr.Memory [10] 在执行时对二进制程序进行检测。为了实现这一目标，Purify [22] 采用了对象代码插入 (OCI) 技术，该技术在对象文件中加入了附加指令，而这些指令可能因系统而异。Valgrind [30] 使用反汇编-合成技术来完成类似的任务。反汇编-再合成技术首先将机器代码转换为一个或多个 IR 操作，插入额外的 IR 以实现内存检查，最后将这些 IR 转换回机器代码。Dr.Memory [10] 与 Valgrind 的不同之处在于，Dr.Memory 基于 DynamoRIO [9] 并执行多种内存检查优化。DynamoRIO 使用复制和注释功能逐字复制接收到的指令：每条指令都附有效果描述注释。Valgrind 和 Dr.Memory 都将影子内存拆分为类似的多级结构，即多级影子映射，以减少内存开销并加速查询。Purify、Valgrind 和 Dr.Memory 的开销很大，性能损失分别为 2x-25x、20x 和 10x。

插入库。DoubleTake [27] 使用插入库来检测二进制程序。它首先将程序执行划分为不同的时间段，每个时间段以一个不可撤销的系统调用结束。然后，在每个时间段开始和结束时，它会检查程序的状态，并确定在该时间段期间是否发生了任何内存错误（即 UAF、内存泄漏或缓冲区溢出）。如果发生，它会重新执行当前时间段，并进行额外的内存检查，以确定错误的确切位置。虽然 DoubleTake 的开销较低（5%），但它只能检测写入已释放内存的操作，而不能检测读取操作，因为它的检测能力基于已释放内存中存储的金丝雀值（随机整数），而读取操作无法破坏该值。

2.2 安全分配器

许多文献都关注通过安全分配器来缓解 UAF [1, 17, 39, 43]。早期的解决方案 [4, 31] 通过随机分配提供概率保护，使攻击者难以（而非不可能）锁定特定对象。此外，一些解决方案还能重新初始化空闲内存 [31]。

Cling [2] 和 TAT [36] 等解决方案本身并不能防止 UAF，但能确保所有重用都是类型安全的。UAF 仍有可能，但仅限于使用指向相同类型对象的悬空指针。

另一种常用技术建立在垃圾收集 (GC) 的基础上 [2, 7, 17, 19, 25]。例如，系统可能会对取消分配进行隔离，直到 GC 确定不再有对它们的引用为止。其缺点是，并发运行的 GC 并不便宜，尤其是强大的安全性要求定期进行“停止-世界”同步。作为一种替代方法，CRCount 会跟踪每个对象的引用次数，并在引用次数为零时释放释放的内存以供重用 [50]。在 FFmalloc [25] 中使用的一种极端解决方案是绝不重复使用和取消映射所有释放的内存。

3 本文方法

作者提出了一种新的基于别名分配与基于虚拟化的直接页表访问的检测 UAF 漏洞的新方法、一种新颖的别名回收解决方案以及一个使用 KML 作为特权后端的 DangZero 原型。

3.1 DangZero

图 1 介绍了 DangZero 的主要组成部分以及它们之间的交互。DangZero 的核心是覆盖分配器，它是默认分配器（如 glibc）的扩展，可以直接访问运行进程的页表。堆分配请求会被拦截，以便创建别名页面并返回给用户。反过来，也会拦截堆删除分配请求，使别名映射失效。特权后端可访问受限功能、特别是直接访问页表。DangZero 不需要对程序或底层分配器进行检测，因此可以在现成的二进制文件上运行。

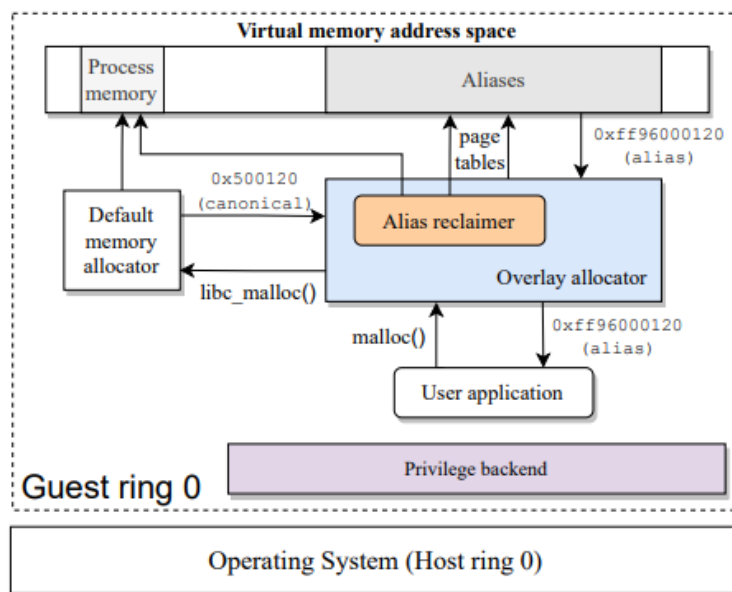


图 1. DangZero 组件概览 [20]

3.2 创建别名页及使其失效

虚拟地址映射保存在进程页表中，由操作系统内核管理。通常情况下，进程要更新映射，需要使用系统调用，这是一项昂贵的操作。Linux 等操作系统也不是为创建自定义映射而构建的，因此在设置方面受到限制。此外，由于为每个别名映射保存了元数据（如 VMA 数据结构），内核内存使用量往往会激增。

有了 DangZero, 我们可以直接从覆盖分配器中修改页表, 完全不受操作系统的控制。DangZero 占用了虚拟地址空间中一个未使用区域, 而内核从未接触过这一区域, 并且 DangZero 可直接写入与该区域对应的页表表项。为了方便访问受限资源, 我们的特权后台提供了安全访问。DangZero 的设计与所使用的具体机制无关, 但一般来说, 它必须支持对页表的直接读写访问, 并允许 TLB 刷新, 同时确保安全性和与系统其他部分的隔离。我们的主后端使用运行内核模式 Linux(KML), 将 Linux 转换为 libOS。

为了根据内存分配请求创建别名页，我们必须考虑需求分页的影响。Linux 在其内存系统中应用了需求分页，因此堆分配在实际使用之前没有物理支持。但是，如果不知道分配的

物理地址，我们就无法创建别名映射。分配器会触及典型页，以强制为内存对象提供物理支持，之后我们就可以设置页表，以确保别名页指向相同的物理页。在实践中，我们没有观察到强制填充页面的开销惩罚，因为页面通常由默认分配器使用，或者在分配后不久就被使用。如果这种方法会给某些工作负载带来困难，那么另一种设计方案是使用自定义页面故障处理程序，以便在堆分配完成后立即创建别名页面。这样，一旦堆分配获得物理支持，就会创建别名页。

3.3 别名页回收器

与传统的 GC 相似，别名页回收器由标记阶段和扫描阶段组成。在标记阶段，程序的所有内存和寄存器都会被扫描，以查找可能指向别名空间的指针。对于指向已释放的别名对象的任何指针，我们将该对象标记为仍被引用。之后，在扫描过程中，我们遍历别名空间中所有被释放的对象，并将所有未标记的对象返回到覆盖分配器的别名释放列表中。回收器会忽略所有未释放的对象，并在释放时将对象清零，以避免循环依赖 [17]。当一个物理页面在不同的分配中被重复使用时，清零对象还能防止未初始化读取造成的数据泄露。

目前，标记和扫描阶段需要暂停目标应用程序。这是垃圾回收中的一种常见模式，目的是获取程序内存的一致视图 [17]。这里还有更多可能的优化方法，如并发和并行标记和清扫。特别是，直接访问页表可以实现快速脏位扫描等优化 [3]。然而，我们选择了更简单的暂停应用方法，因为回收的开销足够低，因此我们并不认为要优先考虑更复杂的并发回收功能。

回收器必须跟踪某些元数据，以便正确执行标记和扫描操作。首先，它需要了解每个别名页面的状态：是否可用（可用于创建新的别名）、使用中（由活对象使用）或已失效（通过 free() 释放的对象）。此外，回收器还需要知道无效对象的边界在哪里，因为指向对象中任何位置的指针都排除了对其任何页面的重用。最后，在标记阶段，回收器需要记住哪些对象在清扫阶段被标记。

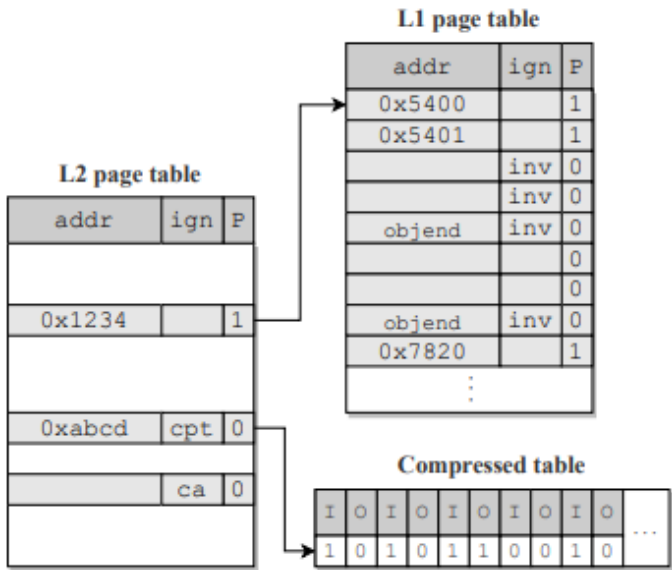


图 2. 页表中的别名回收器的元数据 [20]

回收器将所有这些信息存储在页表中，因此几乎没有内存开销。页表中的元数据如图 2 所示。我们在释放后获得这些存储空间，因为失效的别名页（即调用 free() 后）在各自的页表

项 (PTE) 中被设置为不存在, 这意味着 PTE 中的所有其他位都未被使用, 并被硬件忽略。因此, 我们使用位来区分可用和失效的页表项。此外, 我们还使用另一位来标记每个对象的结束。因此, 可以通过查找未设置该位的连续 PTE, 然后查找该位所在的 PTE 来重建对象边界。图 3 右侧显示的是包含多个 PTE 的 L1 页表其中, P 列表示存在位, 已失效位存储在一个忽略位中, 表示对象结束的位也是如此。图中显示的是正在使用的对象 (如前两个条目)、已失效对象 (第 3-5 条目和第 8 条目) 和可用条目 (条目 6-7) 的组合。

页表回收和压缩。如果页表的 512 个条目中仍然是有效映射 (即 present 位被设置), 则需要保留整个页表, 从而为我们的元数据位提供空闲的存储空间。然而, 当所有条目不存在 (即可用或失效) 时, 保留仅用于元数据的位, 会浪费大量空间。一个 4K 页表有 512 个 64 位条目 (每个条目描述一个页面), 但我们的元数据每个条目只需要 2 位。因此, 当所有条目都不存在时, 我们将页表压缩为 128 字节的数据, 并释放原始页表。我们可以将 32 个压缩页表放入一个 4K 页面, 并使用板块分配器管理这些内存块 [8]。然后, 我们将 L2 页表指向这个压缩条目, 并在页表条目中设置一个特殊的位来指示它的存在 (图 2 中的 cpt)。

在此基础上, 我们确定了页表可能处于的两种常见状态, 从而进一步压缩页表。通常情况下, 压缩后的页表中通常都是已失效的对象。此外, 页面表通常只包含每个页面都是单独对象或每个页面都属于同一个大对象的条目 (即所有或全部对象末端位被设置)。在所有这些情况下, 整个压缩页表 (128 字节) 被压缩成一个位, 并存储在上一级页表中 (图 2 中最后一个 L2 条目中的 ca)。

3.4 特权后端

DangZero 的关键组件之一是其特权后端, 负责从 (用户空间) 分配器直接访问内核特权功能。直接修改页表 (并刷新相应的 TLB) 的能力对我们系统的性能至关重要。

为此, DangZero 的主要后端建立在内核模式 Linux (KML) 之上, KML 是一种修改过的 Linux 内核, 可以使用户空间应用程序在内核模式 (ring0) 下运行。这有效地将 Linux 转变为高效的单内核。它还为应用程序提供了系统上的所有特权, 允许其调用内核代码 (如分配物理页的 alloc_page), 并写入任何内存。然而, KML 仍然由 Linux 支持, 因此受益于所有现有的 Linux 驱动程序, 并支持所有现有的 Linux 二进制文件。

然而, 为任意用户空间程序提供内核权限本质上是不安全的: 程序中的错误或漏洞会影响或破坏系统中的任何其他程序和用户。因此, 与其他单内核系统一样, 我们的系统运行在 (轻量级) 虚拟机 (VM) 中。将客户机与系统的其他部分隔离开来。通过将客户系统专用于单个进程, 我们只考虑一个安全域, 因此, 滥用正交漏洞的攻击者无法利用页表访问权限来入侵其他安全域 (如其他应用程序或主机)。此外, 与裸机基准相比, 这种设置可以提供更高的性能, 因为 KML 内核可以专门针对在虚拟化硬件上运行单个应用程序进行调整。DangZero 不包含此类优化、以公平评估其设计所带来的开销。

对于 DangZero, 我们使用了最新的 Linux 内核和可用的 KML 补丁, 即 v4.0。我们对 KML 内核打了两个补丁这样它就能在当时最新的 LTS 版本 (Ubuntu 的最新 LTS 版本 (20.04) 正常运行。KML 项目为 glibc 2.11 打了补丁, 将系统调用改为直接调用。我们将其移植到 glibc 2.31 (所用操作系统的默认版本)。现代版现代版本的 glibc 允许我们使用操作系统的默认 gcc 版本 (9.4.0)。对于某些功能, 我们需要访问内核数据结构 (如迭代 VMA 结构)。由于有了 KML, 我们可以通过普通函数调用直接从分配器调用该模块。

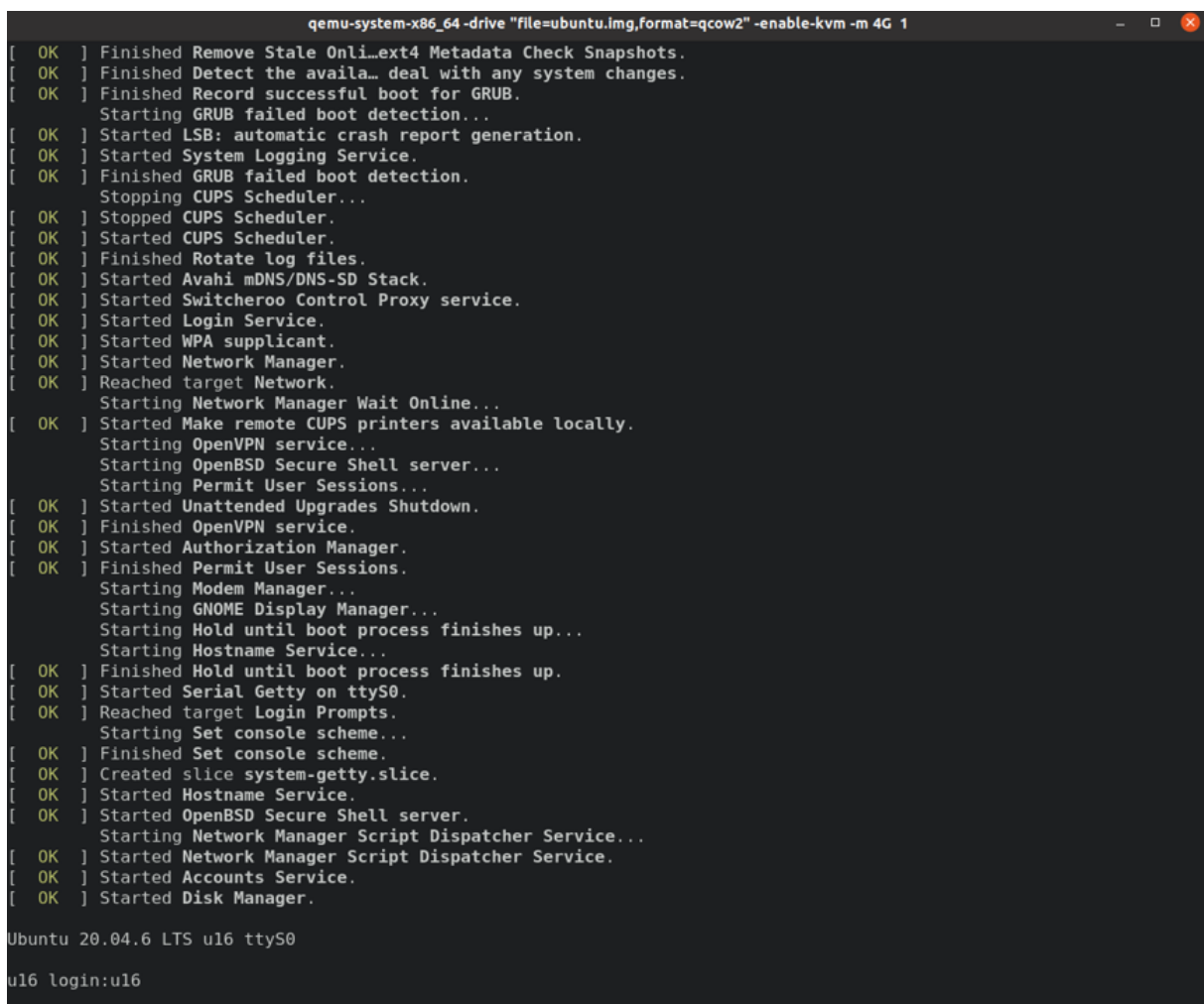
4 复现细节

4.1 与已有开源代码对比

DangZero 在 github 上有开源代码。我的此次任务就是根据 DangZero 的 gitHub 仓库的 README.md 文件复现最终的结果。

4.2 实验环境搭建

我是在 Ubuntu20.04 的虚拟机上搭建实验环境的。根据 README.md 文件中的环境构建操作，需要安装 qemu 和 docker。这两个软件的安装参考其官网。因为我的 Ubuntu 系统是在 VMware 中运行的，此时需要打开 VMware 中的允许虚拟化。然后获取 Ubuntu20.04 的镜像，并在 qemu 上安装，成功运行后如图 3。



```
qemu-system-x86_64 -drive "file=ubuntu.img,format=qcow2" -enable-kvm -m 4G 1
[ OK ] Finished Remove Stale Onli...ext4 Metadata Check Snapshots.
[ OK ] Finished Detect the availa... deal with any system changes.
[ OK ] Finished Record successful boot for GRUB.
Starting GRUB failed boot detection...
[ OK ] Started LSB: automatic crash report generation.
[ OK ] Started System Logging Service.
[ OK ] Finished GRUB failed boot detection.
Stopping CUPS Scheduler...
[ OK ] Stopped CUPS Scheduler.
[ OK ] Started CUPS Scheduler.
[ OK ] Finished Rotate log files.
[ OK ] Started Avahi mDNS/DNS-SD Stack.
[ OK ] Started Switcheroo Control Proxy service.
[ OK ] Started Login Service.
[ OK ] Started WPA supplicant.
[ OK ] Started Network Manager.
[ OK ] Reached target Network.
Starting Network Manager Wait Online...
[ OK ] Started Make remote CUPS printers available locally.
Starting OpenVPN service...
Starting OpenBSD Secure Shell server...
Starting Permit User Sessions...
[ OK ] Started Unattended Upgrades Shutdown.
[ OK ] Finished OpenVPN service.
[ OK ] Started Authorization Manager.
[ OK ] Finished Permit User Sessions.
Starting Modem Manager...
Starting GNOME Display Manager...
Starting Hold until boot process finishes up...
Starting Hostname Service...
[ OK ] Finished Hold until boot process finishes up.
[ OK ] Started Serial Getty on ttyS0.
[ OK ] Reached target Login Prompts.
Starting Set console scheme...
[ OK ] Finished Set console scheme.
[ OK ] Created slice system-getty.slice.
[ OK ] Started Hostname Service.
[ OK ] Started OpenBSD Secure Shell server.
Starting Network Manager Script Dispatcher Service...
[ OK ] Started Network Manager Script Dispatcher Service.
[ OK ] Started Accounts Service.
[ OK ] Started Disk Manager.

Ubuntu 20.04.6 LTS u16 ttyS0
u16 login:u16
```

图 3. Ubuntu20.04 启动

4.3 测试 DangZero

在基础环境搭建好后，接下来就需要在宿主机上编译作者提供的 KML 内核，并将编译好的 KML 内核传送到 qemu 上安装的 Ubuntu 上。KML 就对应着 DangZero 的特权后端部分。然后修改启动配置文件，是该 Ubuntu 在下次启动时使用 KML 内核。但我在这里遇到

了一个问题，在修改完后启动，这时是 KML 内核，在下次启动时就又回到了原来的 Ubuntu 内核。所以，我将启动配置文件修改为如图 4。

图 4. grub 配置文件

修改完成后更新配置并关机，然后在启动命令中加入显示图像的选项，并在启动过程中选择 KML 内核就可以解决问题。接着就可以测试 KML 作为特权后端是否能够提供特权。分别在 “/trusted” 和 “/” 目录下编译执行测试文件。执行结果如图 5。可见，在 trusted 目录下是在特权模式下执行的，在家目录下是在普通模式下执行的。说明特权后端能够提供特权。

图 5. 测试特权后端

紧接着，先获取 glibc-2.31 源码，然后将作者提供的补丁给源码打上，编译 glibc-2.31。编译完成后安装 gcc5，安装 gcc5 主要是为了兼容 dangmod 内核模块。完成后就可以编译安装

dangmod 模块了。安装后如图 6

```
u16@u16:~/dangzero/kmod$ lsmod | grep dangmod
dangmod                16384    0
u16@u16:~/dangzero/kmod$
```

图 6. dangmod 安装

最后执行 test 的 shell 脚本，测试 DangZero 是否能够检测 UAF 漏洞。我在执行后报段错误。根据 DangZero 的原理，我发现还缺少将自己编译完成的 glibc-2.31 替换掉系统的 libc 库，可以使用 LIBRARY_PATH 环境变量。在我尝试后，执行的程序没有报段错误，但不能得到预期结果。然后我看到一个 shell 文件中需要将作者提供的文件编译成动态链接库，这个动态链接库重写了内存分配函数。我就在该内存分配函数中加入了一条打印语句，看程序中的内存分配是否成功执行，最终能够打印出来，说明已经执行了内存分配。然后一直没找到问题的根源。

5 实验结果分析

本次实验是在 qemu 上搭建 Ubuntu 环境，属于 DangZero 的客户机部分。在 Ubuntu 上安装的 KML 是一种能在内核空间内执行普通用户空间程序的技术，属于 DangZero 的特权后端部分。dangmod 模块为属于 DangZero 的覆盖层分配器部分。在实验中，我能够在“/test”目录下以 ring0 权限执行程序，说明 KML 的安装没有问题，但最终程序中不该执行的语句（在发生 UAF 后的语句）被执行了，说明在覆盖层分配器的部分出现了问题，我检查过内存分配函数，它是可以被执行的，但无法检测出 UAF。

6 总结与展望

DangZero 是一种高效、可扩展、二进制兼容的 UAF 检测系统。但目前 DangZero 的实现模型存在两种限制。首先，由于 DangZero 是在 KML 的基础上开发的，其目前无法在版本低于 4.0（KML 的最新版本）的 Linux 内核上运行。这个可以通过将 KML 移植到更新的内核版本上来解决。其次，DangZero 目前的实现还不是线程安全的。线程安全可通过页表锁定来解决。隔离运行程序所带来的其他限制，例如不能跨进程共享内存，需要基于管理程序的解决方案，如跨虚拟机页面共享。

参考文献

- [1] Sam Ainsworth and Timothy M. Jones. Markus: Drop-in use-after-free prevention for low-level languages. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 578–591. IEEE, 2020.

- [2] Periklis Akritidis. Cling: A memory allocator to mitigate dangling pointers. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 177–192. USENIX Association, 2010.
- [3] Adam Belay, Andrea Bittau, Ali José Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis. Dune: Safe user-level access to privileged CPU features. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 335–348. USENIX Association, 2012.
- [4] Emery D Berger and Benjamin G Zorn. Diehard: Probabilistic memory safety for unsafe languages. *Acm sigplan notices*, 41(6):158–168, 2006.
- [5] Lukas Bernhard, Michael Rodler, Thorsten Holz, and Lucas Davi. xtag: Mitigating use-after-free vulnerabilities via software-based pointer tagging on intel x86-64. In *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*, pages 502–519. IEEE, 2022.
- [6] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Adv. Comput.*, 58:117–148, 2003.
- [7] Hans-J Boehm, Alan J Demers, and Scott Shenker. Mostly parallel garbage collection. *ACM SIGPLAN Notices*, 26(6):157–164, 1991.
- [8] Jeff Bonwick. The slab allocator: An object-caching kernel memory allocator. In *USENIX Summer 1994 Technical Conference, Boston, Massachusetts, USA, June 6-10, 1994, Conference Proceeding*, pages 87–98. USENIX Association, 1994.
- [9] Derek Bruening and Saman Amarasinghe. Efficient, transparent, and comprehensive run-time code manipulation. 2004.
- [10] Derek Bruening and Qin Zhao. Practical memory checking with dr. memory. In *International Symposium on Code Generation and Optimization (CGO 2011)*, pages 213–223. IEEE, 2011.
- [11] Nathan Burow, Derrick Paul McKee, Scott A. Carr, and Mathias Payer. CUP: comprehensive user-space protection for C/C++. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pages 381–392. ACM, 2018.
- [12] Zixian Cai, Stephen M. Blackburn, Michael D. Bond, and Martin Maas. Distilling the real cost of production garbage collectors. In *International IEEE Symposium on Performance Analysis of Systems and Software, ISPASS 2022, Singapore, May 22-24, 2022*, pages 46–57. IEEE, 2022.

- [13] Haehyun Cho, Jinbum Park, Adam Oest, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. Vik: practical mitigation of temporal memory safety violations through object ID inspection. In Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch, editors, *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, pages 271–284. ACM, 2022.
- [14] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-c - A software analysis perspective. In George Eleftherakis, Mike Hinchey, and Mike Holcombe, editors, *Software Engineering and Formal Methods - 10th International Conference, SEFM 2012, Thessaloniki, Greece, October 1-5, 2012. Proceedings*, volume 7504 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2012.
- [15] Thurston H. Y. Dang, Petros Maniatis, and David A. Wagner. Oscar: A practical page-permissions-based scheme for thwarting dangling pointers. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 815–832. USENIX Association, 2017.
- [16] Dinakar Dhurjati and Vikram S. Adve. Efficiently detecting all dangling pointer uses in production servers. In *2006 International Conference on Dependable Systems and Networks (DSN 2006), 25-28 June 2006, Philadelphia, Pennsylvania, USA, Proceedings*, pages 269–280. IEEE Computer Society, 2006.
- [17] Márton Erdos, Sam Ainsworth, and Timothy M. Jones. Minesweeper: a "clean sweep" for drop-in use-after-free prevention. In Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch, editors, *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, pages 212–225. ACM, 2022.
- [18] Reza Mirzazade Farkhani, Mansour Ahmadi, and Long Lu. Ptauth: Temporal memory safety via robust points-to authentication. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1037–1054. USENIX Association, 2021.
- [19] Nathaniel Wesley Filardo, Brett F. Gutstein, Jonathan Woodruff, Sam Ainsworth, Lucian Paul-Trifu, Brooks Davis, Hongyan Xia, Edward Tomasz Napierala, Alexander Richardson, John Baldwin, David Chisnall, Jessica Clarke, Khilan Gudka, Alexandre Joannou, A. Theodore Markettos, Alfredo Mazzinghi, Robert M. Norton, Michael Roe, Peter Sewell, Stacey D. Son, Timothy M. Jones, Simon W. Moore, Peter G. Neumann, and Robert N. M. Watson. Cornucopia: Temporal safety for CHERI heaps. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 608–625. IEEE, 2020.

- [20] Floris Gorter, Koen Koning, Herbert Bos, and Cristiano Giuffrida. Dangzero: Efficient use-after-free detection via direct page table access. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1307–1322, 2022.
- [21] Binfa Gui, Wei Song, and Jeff Huang. Uafsan: an object-identifier-based dynamic approach for detecting use-after-free vulnerabilities. In Cristian Cadar and Xiangyu Zhang, editors, *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, pages 309–321. ACM, 2021.
- [22] Reed Hastings. Purify: Fast detection of memory leaks and access errors. In *Proceedings of the USENIX Winter'92 Conference*, pages 125–136, 1992.
- [23] Daniel Kroening and Michael Tautschnig. CBMC - C bounded model checker - (competition contribution). In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 389–391. Springer, 2014.
- [24] Hsuan-Chi Kuo, Dan Williams, Ricardo Koller, and Sibin Mohan. A linux in unikernel clothing. In Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer, editors, *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 11:1–11:15. ACM, 2020.
- [25] Byoungyoung Lee, Chengyu Song, Yeongjin Jang, Tielei Wang, Taesoo Kim, Long Lu, and Wenke Lee. Preventing use-after-free with dangling pointers nullification. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.
- [26] Daiping Liu, Mingwei Zhang, and Haining Wang. A robust and efficient defense against use-after-free exploits via concurrent pointer sweeping. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1635–1648. ACM, 2018.
- [27] Tongping Liu, Charlie Curtsinger, and Emery D Berger. Doubletake: Fast and precise error detection via evidence-based dynamic analysis. In *Proceedings of the 38th International Conference on Software Engineering*, pages 911–922, 2016.
- [28] Toshiyuki Maeda and Akinori Yonezawa. Kernel mode linux: Toward an operating system protected by a type theory. In Vijay A. Saraswat, editor, *Advances in Computing Science - ASIAN 2003 Programming Languages and Distributed Computation, 8th Asian Computing Science Conference, Mumbai, India, December 10-14, 2003, Proceedings*, volume 2896 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2003.

- [29] Santosh Nagarakatte, Jianzhou Zhao, Milo M. K. Martin, and Steve Zdancewic. CETS: compiler enforced temporal safety for C. In Jan Vitek and Doug Lea, editors, *Proceedings of the 9th International Symposium on Memory Management, ISMM 2010, Toronto, Ontario, Canada, June 5-6, 2010*, pages 31–40. ACM, 2010.
- [30] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM Sigplan notices*, 42(6):89–100, 2007.
- [31] Gene Novark and Emery D Berger. Dieharder: securing the heap. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 573–584, 2010.
- [32] Mads Chr. Olesen, René Rydhof Hansen, Julia L. Lawall, and Nicolas Palix. Coccinelle: Tool support for automated CERT C secure coding standard certification. *Sci. Comput. Program.*, 91:141–160, 2014.
- [33] Zekun Shen and Brendan Dolan-Gavitt. Heapexpo: Pinpointing promoted pointers to prevent use-after-free vulnerabilities. In *ACSAC ’20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020*, pages 454–465. ACM, 2020.
- [34] Jangseop Shin, Donghyun Kwon, Jiwon Seo, Yeongpil Cho, and Yunheung Paek. Crcount: Pointer invalidation with reference counting to mitigate use-after-free in legacy C/C++. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.
- [35] Yulei Sui and Jingling Xue. SVF: interprocedural static value-flow analysis in LLVM. In Ayal Zaks and Manuel V. Hermenegildo, editors, *Proceedings of the 25th International Conference on Compiler Construction, CC 2016, Barcelona, Spain, March 12-18, 2016*, pages 265–266. ACM, 2016.
- [36] Erik van der Kouwe, Taddeus Kroes, Chris Ouwehand, Herbert Bos, and Cristiano Giuffrida. Type-after-type: Practical and complete type-safe memory reuse. In *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*, pages 17–27. ACM, 2018.
- [37] Erik van der Kouwe, Vinod Nigade, and Cristiano Giuffrida. Dangsang: Scalable use-after-free detection. In Gustavo Alonso, Ricardo Bianchini, and Marko Vukolic, editors, *Proceedings of the Twelfth European Conference on Computer Systems, EuroSys 2017, Belgrade, Serbia, April 23-26, 2017*, pages 405–419. ACM, 2017.
- [38] Kostyantyn Vorobyov and Padmanabhan Krishnan. Comparing model checking and static program analysis: A case study in error detection approaches. *Proc. SSV*, pages 1–7, 2010.
- [39] Brian Wickman, Hong Hu, Insu Yun, Daehee Jang, Jungwon Lim, Sanidhya Kashyap, and Taesoo Kim. Preventing use-after-free attacks with fast forward allocation. In Michael D.

Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2453–2470. USENIX Association, 2021.

- [40] Hua Yan, Yulei Sui, Shiping Chen, and Jingling Xue. Machine-learning-guided typestate analysis for static use-after-free detection. In *Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017*, pages 42–54. ACM, 2017.
- [41] Hua Yan, Yulei Sui, Shiping Chen, and Jingling Xue. Spatio-temporal context reduction: a pointer-analysis-based static approach for detecting use-after-free vulnerabilities. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 327–337. ACM, 2018.
- [42] Yves Younan. Freesentry: protecting against use-after-free vulnerabilities due to dangling pointers. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.
- [43] Insu Yun, Woosun Song, Seunggi Min, and Taesoo Kim. Hardsheap: a universal and extensible framework for evaluating secure allocators. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 379–392, 2021.
- [44] Tong Zhang, Dongyoon Lee, and Changhee Jung. BOGO: buy spatial memory safety, get temporal memory safety (almost) free. In Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck, editors, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, pages 631–644. ACM, 2019.