事务内存机制在系统安全中的应用: 现状与展望

李从午[1,2,3], 林璟锵[1,2,3], 蔡权伟[1,2], 罗 勃[4]

[1] 中国科学院信息工程研究所 信息安全国家重点实验室, 北京 中国 100093
[2] 中国科学院数据与通信保护研究教育中心, 北京 中国 100093
[3] 中国科学院大学网络空间安全学院, 北京 中国 100049
[4] 堪萨斯大学电子工程与计算机科学系, 劳伦斯 美国 KS 66045

摘要 为了提高并行程序中共享内存数据的读写访问性能, 事务内存机制于 1993 年被提出。因为事务内存机制直接涉及内存数据的读写控制, 所以也得到了系统安全研究人员的极大关注。2013 年, Intel 公司开始支持 TSX (Transactional Synchronization eXtension)特性, 第一次在广泛使用的计算机硬件中支持事务内存机制。利用事务内存机制的内存访问跟踪、内存访问信号触发和内存操作回滚, 以及 Intel TSX 特性的用户态事务回滚处理、在 Cache 中执行所有操作和硬件实现高效率, 研究人员完成了各种的系统安全研究成果, 包括: 授权策略实施、虚拟机自省、密钥安全、控制流完整性、错误恢复和侧信道攻防等。本文先介绍了各种基于事务内存机制的研究成果; 然后分析了现有各种系统安全研究成果与事务内存机制特性之间的关系, 主要涉及了 3 个角度: 内存访问的控制、事务回滚处理、和在 Cache 中执行所有操作。我们将已有的研究成果的技术方案从 3 个角度进行分解, 与原有的、不基于事务内存机制的解决方案比较, 解释了引入事务内存机制带来的技术优势。最后, 我们总结展望了将来的研究, 包括: 硬件事务内存机制的实现改进, 事务内存机制(尤其是硬件事务内存机制)在系统安全研究中的应用潜力。

关键词 系统安全; 事务内存; 网络空间安全
中图法分类号 TP309.2 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2019.11.04

Applying Transactional Memory in System Security: Present and Future

LI Congwu[1,2,3], LIN Jingqiang[1,2,3], CAI Quanwei[1,2], LUO Bo[4]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing 100093, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
[4] Department of Electrical Engineering and Computer Science, the University of Kansas, KS 66045, USA

Abstract Transactional memory was proposed in 1993 to improve the performance of shared memory access in parallel programs. Because transactional memory is related to the access control of memory data, lots of system security researchers also pay great attention to it. In 2013, Intel starts to support TSX (Transactional Synchronization eXtension), which is the first transaction memory mechanism in widely-used computer hardware. Utilizing memory access tracking, memory access signal triggering and memory operation rollback of transactional memory, as well as user-space rollback processing, in-cache execution of all operations and high efficiency of hardware implementations with Intel TSX, various system security schemes are finished, including authorization policy enforcement, virtual machine introspection, cryptographic key protection, control flow integrity, fault recovery and side channel attack/defense. This paper introduces the system security schemes based on transactional memory, and analyzes the relation between these schemes and the features of transactional memory, in terms of the control of memory access, the rollback processing and the in-cache execution of all operations. We deconstruct these schemes from these features, compare them with existing schemes not based on transactional memory, and then explain the advantages introduced by transactional memory. Finally, we discuss the future of applying transactional memory in system security, including the improvement of hardware transactional memory, and the potential applications of transactional memory, especially hardware transactional memory, in system security.

Key words system security; transactional memory; cyber security

通讯作者: 林璟锵, 博士, 研究员, Email: linjingqiang@iie.ac.cn。
本论文工作得到自然科学基金“通用计算平台的密钥保护技术研究”(No.61772518)和国家重点研发计划网络空间安全重点专项“基于国产密码算法的移动互联网密码服务支撑基础设施关键技术”(No.2017YFB0802100)资助。
收稿日期: 2017-12-13; 修改日期: 2018-03-07; 定稿日期: 2019-11-04

1 简介

事务内存(Transactional Memory)是M. Herlihy和J. Moss在1993年提出的技术概念[1], 其目的是为了提高并行程序中共享内存数据的读写访问性能、并同时简化编程人员的工作。事务内存机制的基本原理是, 在系统底层实现操作原语, 使得程序代码可以定义原子执行的事务操作; 相应的, 在事务操作中的内存数据访问也必须满足原子性要求: 事务执行中间的内存数据状态对其他线程/进程不可见、事务执行过程中访问的内存数据不应被其他线程/进程改变。事务内存的概念提出之后, 得到了非常广泛的关注; 有多种不同形式的事务内存机制被实现, 包括硬件[2-6]、软件[7-12]、软硬件混合[13-16]、分布式[17-21]等多种形式。事务内存机制在各种不同高性能计算场景中的使用, 也有不少相关的研究成果[22-25]。

因为事务内存机制直接涉及内存数据的读写控制, 所以也得到了系统安全研究人员的极大关注。2008 年, TMI 基于软件事务内存机制[12]完成了授权策略的实施方案[26]; 2013 年, Intel 公司开始在 Haswell CPU 上支持 TSX (Transactional Synchronization eXtension)特性[27], 第一次在广泛使用的计算机硬件中支持事务内存机制。Intel TSX 特性给系统安全研究人员提供了非常便利的研究基础, 随之出现了许多有影响力的、基于硬件事务内存机制的系统安全成果, 包括: 虚拟机自省 Virtual Machine Introspection (VMI)方案[28]、密钥安全方案[29]、控制流完整性方案[30]、攻击响应和错误恢复[29, 31]、侧信道攻击和防护方案[32-35]、地址随机化的攻击方案[36]等。上述系统安全研究成果, 有部分是利用事务内存机制的内存控制能力和事务回滚处理[26,28-29,31]、有部分是利用事务内存机制构建锁机制[30]、有部分则是利用了 Intel TSX 特性实现中的特定处理[32-36]。

本文总结了现有基于事务内存机制的系统安全研究, 分析了各种系统安全研究方案与事务内存机制之间的关系。然后, 我们进一步列出了现有硬件事务内存机制实现上的改进方向, 讨论了事务内存机制在系统安全研究中的潜在研究价值。

2 事务内存机制及其实现

下面, 我们先以 Intel TSX 特性[27]为例来介绍事务内存机制。然后, 大致介绍其它事务内存机制实现。

2.1 事务内存机制

事务内存机制的提出, 是为了解决传统共享数据多线程编程锁机制的性能问题。在传统多线程编程中, 为了解决共享数据的读写冲突, 通常采用锁机制: 在访问共享数据之前加锁、进入临界区, 访问共享数据之后、解锁退出临界区。加锁的临界区代码只能串行执行, 即使在实际的运行中并没有数据读写冲突。事务内存机制的解决思路是: 先尝试以不加锁的方式访问共享数据, 支持并行执行; 在确实有发生数据冲突的时候, 再进行额外的数据冲突处理、事务回滚。

事务内存机制通常提供如下的基本功能操作[1]: 在程序中定义事务原子执行的代码段, 以及获得原子执行的结果状态(成功或者失败)。运行在事务内存模式的任务, 数据的中间状态对其它线程/进程不可见、运行过程中访问的内存数据也不应被其它线程/进程改变; 所以, 一旦有其他任务同时访问了事务内存任务所访问的数据、且其中一方是写操作(即, 发生数据冲突), 就违反了事务内存机制的原则, 则事务内存任务被中止, 回滚到事务开始执行时刻的状态、所有更新的数据被清除。

在事务内存任务的执行过程中, 事务内存机制自动地缓存开始状态、跟踪执行过程中的数据访问、检测数据冲突、完成状态回滚、和提交事务等。例如, IBM System z 提供了 TBEGIN 和 TEND 指令, 分别表示事务内存任务的开始和结束[4]。如果事务正常开始、继续执行 TBEGIN 的下一条指令, 则 CC 状态寄存器被置为零; 如果发生数据冲突或者其他异常时, 则自动跳转到 TBEGIN 的下一条指令、且 CC 状态寄存器被置为非零值, 同时状态回滚到事务开始执行的时刻。

2.2 Intel TSX 特性

Intel TSX 特性在第四代酷睿 CPU(即 Haswell 架构)开始支持, 实现了硬件事务内存机制。在事务内存任务的执行过程中, Intel TSX 特性维护该任务的 Write-Set 和 Read-Set, 即有写入操作的内存区和有读取操作的内存区, 用于检测数据冲突和状态回滚。Write-Set 和 Read-Set 分别使用 L1 Cache 和 L3 Cache 的存储空间。

Intel TSX 特性引入了两类不同的事务内存编程指令[27], 分别称为 Hardware Lock Elision (HLE)和 Restricted Transactional Memory (RTM)。HLE 支持 2 条新的指令前缀, XACQUIRE 和 XRELEASE, 用于传统的加锁和解锁指令之前。HLE 支持历史遗留代码, CPU 先尝试以事务内存模式运行代码, 如果有数据冲突导致事务中止, 则事务回滚后、以传统的加锁和解锁指令方式运行。进入临界区时, HLE 向内存锁

写入数值(如果非事务内存模式运行, 则是加锁); 退出临界区时, HLE 再次访问内存锁、并检查写入的数值一致。如果 CPU 硬件不支持 TSX 特性, 则直接忽略该前缀。

RTM 提供了更为灵活的事务内存机制的使用方式; 在事务异常导致回滚时, 开发者可以有自定义的 Fallback 函数处理。XBEGIN 和 XEND 指令, 分别表示事务内存任务的开始和结束。XBEGIN 指令的操作数是 Fallback 函数地址。在 Fallback 函数中, 可以根据事务回滚的原因, 进行相应的不同处理。此外, 还有 XABORT 和 XTEST 指令, 分别用于自动地中止事务、检查是否处于事务内存模式。

2.3 其他事务内存机制实现

除了 Intel TSX 特性外, 还有多种硬件实现的事务内存机制设计[2-3], 还包括在 SUN SPARC [4]、IBM System z [5]、IBM Blue Gene/Q [6]等平台的实现。事务内存机制也可以使用软件方式实现[7-12], 或者是软硬件混合实现[13-16]: 部分功能由 CPU 硬件实现、部分功能由系统软件实现。对于分布式计算环境, 也有相应的分布式事务内存机制[17-21]用于多台计算机之间的内存读写控制。

3 基于事务内存机制的系统安全研究现状

事务内存机制直接涉及到内存数据的读写控制, 与此同时, 大量的系统安全都关系到内存数据的访问[37]; 所以, 基于事务内存机制的系统安全研究已有很长时间。早在 2008 年, ASeD [38]和 TMI [26]就考虑了利用事务内存机制来实现安全功能; TMI 基于软件事务内存机制[12]实施授权策略, 完成了详细的方案设计和原型系统。ASeD 仅仅提出了概念性的设想, 包括在可用性、安全和调试方面的应用, 但是没有给出详细的方案。

2013 年, Intel TSX 特性推出, 第一次在广泛使用的计算机硬件中支持事务内存机制。在现有基于 Intel TSX 特性的系统安全研究成果中, 有部分是利用了事务内存机制的自身性质、但基于硬件特性以获得更好的安全假设和性能[28-31], 有部分则利用 Intel TSX 特性实现硬件事务内存机制的特点[29-30,32-36]。

上述基于事务内存机制的系统安全研究成果, 涉及了数据机密性[29]、控制流完整性[30]、访问授权[26]、恶意代码监控[28]、错误恢复[31]、侧信道攻防[32-36]、应用软件安全[32-33]等, 几乎覆盖了系统安全研究的各个重要方面。

3.1 利用硬件锁机制实现控制流完整性

控制流完整性 Control Flow Integrity (CFI)于 2005 年提出[39], 用于解决代码重用攻击。CFI 的基本原理是: 在运行期间检查程序代码之间的跳转, 必须符合正常的跳转规则; 例如, 函数返回的目标地址, 必须是该函数的调用代码的下一行。细粒度的严格控制流完整性需要在编译期间给所有的代码跳转位置加标签, 在运行期间进行检查。严格的控制流完整性, 在实现上非常困难, 因为需要在编译期间事先确定所有可能的代码跳转。粗粒度的控制流完整性, 更具有可实现性: 用少量的规则来限定代码跳转, 而不要求对每一次代码跳转进行精确控制。例如, RET 指令的目标地址, 必须是 CALL 指令的下一条指令; 注意: 并不要求该CALL指令就是调用了相应的函数。

2011 年, Control Flow Locking [40]提出了基于锁的细粒度控制流完整性, 将代码跳转映射为加锁的临界区, 攻击者破坏控制流完整性就会导致加锁/解锁不匹配。Intel TSX 特性与锁机制具有相似性, 尤其是 HLE 方式。TSX-CFI 分别基于 RTM 和 HLE 实现了粗粒度的控制流完整性和细粒度的控制流完整性。

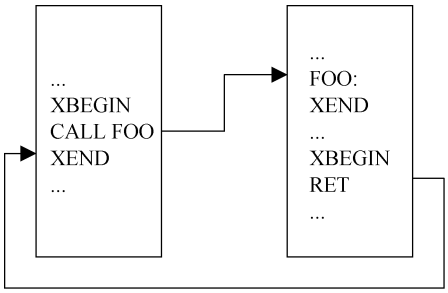

图 1 基于 RTM 方式的控制流完整性

Figure 1 RTM-based Control flow integrity

利用 RTM 实现粗粒度的控制流完整性[30], 基本原理如下:

- 在所有 CALL 指令前后, 分别增加 XBEGIN 和 XEND 指令;
- 在所有函数的开始位置, 增加 XEND 指令;
- 在所有 RET 指令之前, 增加 XBEGIN 指令。

从上图中, 可以看出, 经过编译期间的处理, 正常的函数调用和返回, 分别实现为 2 个事务内存任务。如果出现控制流劫持, 就会出现异常的事务中止; 例如, RET 指令的目标地址不是 CALL 指令的下一条指令(即 XEND 指令), 继续执行就会事务回滚(执行到另一条 XBEGIN 指令, 因 Intel TSX 不支持事务内存任务的嵌套而事务回滚, 或者在执行过程中, 因为数据冲突等其他原因而回滚); CALL 指令的目标地址, 如果不是某一个函数的开始, 也

会有类似的事务回滚。相比原有的控制流完整性方案, 基于 RTM 的 TSX-CFI 具有简洁有效、实现简单的特点。

细粒度的严格控制流完整性需要在代码跳转位置加标签, 在运行期间进行检查, 检查跳转指令与目标地址的标签一致。HLE 实现细粒度控制流完整性的基本原理与基于 RTM 的粗粒度控制流完整性类似, 而且由于 HLE 在事务内存任务结束时、检查写入数值的一致性, 所以可以将该数值作为细粒度控制流完整性所需要的检查标签。

相比 Control Flow Locking [40], 基于 HLE 的 TSX-CFI 同时还具有攻击恢复能力: 因为事务内存任务异常时的自动回滚, 当控制流完整性被破坏时, 标签不一致、HLE 检查发现写入的数值不一致, 事务回滚、程序恢复到攻击之前的状态。

3.2 利用事务内存机制的内存控制实现内存数据安全

不同于 TSX-CFI, 有更多系统安全方案利用事务内存机制的内存数据读写控制: 事务内存机制实现了事务内存任务与其他任务之间的、实时的、不可绕过的事件触发, 可以在此基础上完成授权实施程序与敏感资源访问、数据机密性程序与恶意读取操作、内核关键数据监控程序与 Rootkit 攻击等之间的必要交互。

2008 年, TMI[26]就开始了基于事务内存机制的系统安全研究尝试。利用软件事务内存 Software Transactional Memory (STM)机制实现中的数据冲突检查, TMI 进一步增加了授权策略检查、从而保证了授权策略的严格实施。如下图所示:

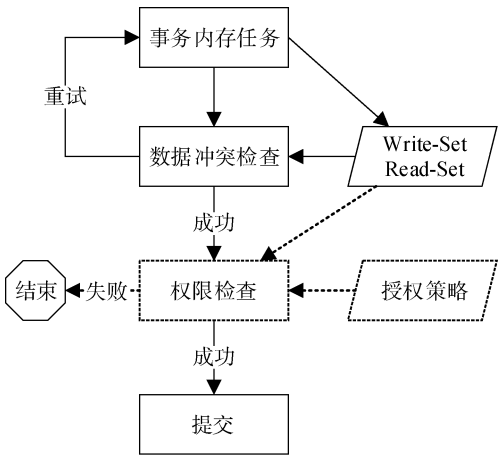

图 2 基于事务内存机制的授权策略实施

Figure 2 Authorization policy enforcement based on transactional memory

图中的虚线部分是在软件事务内存机制实现的基础上新增的授权策略实施步骤; 借助于软件事务内存机制实现的 Write-Set 和 Read-Set 维护, TMI 确保了授权策略的实施、包括在各种竞争状态情况下。TMI 利用软件事务内存机制 DSTM2 [12]实现了原型系统。通常计算机系统中的授权策略实施, 只能依赖于专业人员的编程实现和大规模的测试; 而 TMI 的授权策略实施在设计上具有更可靠的理论基础, 能够确保覆盖各种极端边界情况: 如果在某些情况下, 授权策略没有被实施, 则意味着事务内存机制有问题。

TxIntro [28]和 Mimosa [29]都是基于 Intel TSX 特性: TxIntro 在事务内存任务中监控 Read-Set 检测数据的异常变化, Mimosa 在事务内存任务中监控 Write-Set 检测非授权的读取操作。TxIntro 在 Xen 平台上实现了高效的虚拟机自省方案: 主监控任务以事务内存模式运行, 各虚拟机操作系统的内核关键数据都在主监控任务的 Read-Set 中, 当内核关键数据被改动时、事务回滚, 主监控任务检查到数据冲突来源后, 开始对相应虚拟机的自省检查; 在虚拟机的自省检查中, 以事务内存模式运行虚拟机内核关键数据的读取、并检查数据的一致性, 读取得到一致性的可用数据之后, 结束事务内存任务, 然后运行常规的虚拟机自省程序代码。相比已有的虚拟机自省系统[41-43], TxIntro 能够更实时地监控关键数据的改动, 能够实现更细粒度的关键数据监控(Intel TSX 特性的数据冲突检测粒度小, 以 64 字节的 Cache Line 为单位)。

Mimosa 基于 Intel TSX 特性, 构建了安全的 RSA 密码计算环境, 防御各种内存信息泄露攻击 Memory Disclosure Attack。在执行 RSA 签名或解密时, 以事务内存模式运行: 先将 RSA 私钥的明文解密到内存中(即, 进入 Write-Set), 然后执行 RSA 私钥计算, 最后清除残余的敏感数据、结束事务内存任务。在 RSA 私钥计算期间, 恶意的内存信息泄露攻击如果读取 RSA 私钥, 就会导致事务回滚、RSA 私钥被自动清除, 攻击者不能获得任何信息。同时, Mimosa 结合了基于寄存器的 AES 算法实现[44]: 没有 RSA 计算任务时, RSA 私钥使用 AES 算法加密、以密文形式存储在内存中, 且 AES 密钥只存储在寄存器中。由于 Intel TSX 事务内存任务非常易于受到各种系统事件影响而回滚, Mimosa 还给出了详细的总结: 如何在 Intel TSX 事务内存模式下执行长时间、耗内存的密码计算任务, 例如 2048/3072/4096 bit 的 RSA 算法。

进一步, 由于Intel TSX事务内存任务执行的中间状态数据存储在Cache中, Mimosa保证敏感数据不会出现在内存芯片上, 可以同时防御 Cold-Boot攻击[45]和远程 Cache-based 时序攻击[46-48]。相比已有的 RSA 密钥安全方案[49-52], Mimosa 对操作系统基础服务的安全假设更少, 而且有明显的性能优势、与没有安全增强的普通 RSA 密码软件实现性能差异很小。

3.3 利用事务回滚处理的错误恢复

利用事务内存机制的回滚处理[53], 完成了函数级的容错方案: 使用 2 个程序实例(分别称为 Leader 和 Trailer)冗余地执行函数功能; Leader 实例以正常模式运行, 在执行结束之后, 结果不正式提交、暂存在系统中; 然后Trailer实例以事务内存模式运行, 在事务内存任务结束之前、比较 2 个实例的运行结果, 如果不一致就主动触发事务回滚处理、清除 Trailer 的执行效果。由于 Leader 实例的函数功能已经执行结束, 如果结果不一致就只能杀死进程来清除执行效果。注意, 如果 Leader 实例也以事务内存模式运行, 则比较 2 个事务内存任务的运行结果、必定会有数据冲突导致事务回滚。

HAFT [31]利用事务内存机制的回滚处理, 完成了指令级冗余 Instruction-Level Redundancy (ILR)的容错系统。如图所示, HAFT 采取了指令级冗余来检测临时硬件错误, 然后进一步主动触发事务回滚处理来完成错误恢复, 能够同时清除全部冗余执行的所有执行结果。

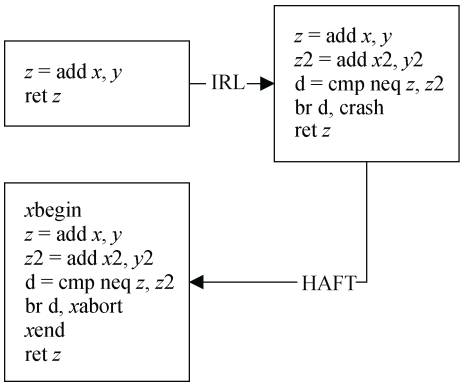

图 3 指令级冗余的错误恢复

Figure 3 Fault recovery of instruction-level redundancy

基于 Intel TSX 特性提供的 XBEGIN、XEND 和 XABORT 指令, HAFT 完成了相应的编译器工具, 自动地增加支持错误检测和错误恢复的代码。性能试验表明, HAFT 的性能代价是 2 倍处理时间; 由于利用 Intel TSX 特性的、硬件支持的事务回滚, HAFT 实现了高效的错误恢复能力。此外, 其他利用事务回滚处理的错误恢复解决方案还有[54]。

3.4 利用 Intel TSX 特性异常处理的应用层隔离

在 Intel TSX 特性的事务内存机制实现中, 除了数据冲突, 还会有很多其他情况导致事务中止, 包括操作系统的任务调度、各种中断和异常[27]。在计算机系统中, 上述情况(任务调度、中断和异常等)通常都是在操作系统内核中处理, 用户态的应用程序是无感知的, 即: 应用程序并不知道任务调度、中断和异常的存在。但是, 在 Intel TSX 特性的 RTM 方式中, 提供了开发者自定义的 Fallback 函数, 在事务内存任务因各种原因而回滚时, CPU 硬件直接跳转到用户态的 Fallback 函数、且该跳转不涉及任何操作系统内核处理。基于该特征, 应用程序可以排除或检测任务调度、中断和异常等事件的干扰。

DrK 攻击[36]利用了上述特征来攻击内核地址空间布局随机化 Kernel Address Space Layout Randomization (KASLR): 即使用户态应用程序发生任意次数的内存访问异常, 也不会陷入操作系统的异常处理; 而且 Intel CPU 硬件在判断 Unmapped、Mapped、Executable 和 Non-executable 内存的过程中, 有很明显的时序差异。DrK 攻击以 RTM 方式、任意次数地探测不同地址内存的 Unmapped/Mapped 和 Executable/Non-executable 属性, 在访问异常跳转的 Fallback 函数中计时, 根据计时结果、就可以快速去除内核地址空间布局随机化。由于运行在 RTM 方式的事务内存模式, 操作系统并不会感知到 DrK 攻击程序的内存访问异常; 而通常情况下, 进程发生内存访问异常, 就会被操作系统杀死。

同样利用上述特征, T-SGX [32]实现了防范恶意操作系统的防御机制, 消除了恶意操作系统对 SGX Enclave 的 Page-fault 侧信道攻击[55-56]。Intel SGX 特性[57]是 Intel 公司近年推出的硬件安全特性, 为用户态的应用程序提供不受操作系统和特权任务影响的机密性、完整性和远程证明。恶意操作系统虽然不能直接读取 SGX Enclave 的数据, 但是可以有意识地使 SGX Enclave 不断地触发缺页错误, 从而跟踪 SGX Enclave 的执行路径、推导出正在处理的敏感数据。T-SGX 将 SGX Enclave 处理敏感数据的程序代码以 RTM 方式运行, 使得: 在运行过程中, 发生缺页错误, 事务回滚、直接跳转到用户态的 Fallback 函数, 并不会陷入操作系统的缺页错误处理, 所以操作系统无法跟踪 SGX Enclave 的执行路径。为了能够

在 Intel TSX 事务内存模式下完成各种处理, T-SGX 实现了自动化的编译期工具, 估算代码运行时的内存使用、将代码拆分为多个事务内存任务, 使得: 各个事务内存任务所读写的内存空间不会超过 CPU 硬件容量的限制, 确保能够顺利成功执行。

类似的, 为了防御恶意操作系统对于 SGX Enclave 的 Page-fault 侧信道攻击, Déjà vu [33]采取了不同的思路: SGX Enclave 程序测量自己的运行时间, 与预期的时间值比较; 如果超出预期, 则认为有可能恶意操作系统通过缺页错误等方式在控制 SGX Enclave 程序的运行。与此同时, 要求操作系统不会调度或者中断 SGX Enclave 程序和计时器的执行; 即, 提出了对操作系统的额外服务质量要求。在假设操作系统不可信的前提下, Déjà vu 以 RTM 方式运行循环代码, 作为独立的 SGX Enclave 线程, 将循环次数作为计时结果; 类似的, 在计时器执行期间, 如果恶意操作系统试图暂停计时器的运行、影响计时结果(例如, 通过任务调度或者缺页错误), 也会跳转到用户态的 Fallback 函数而被检测到。

3.5 利用 Intel TSX 特性 Cache 使用的侧信道攻击和防御

Intel TSX 特性维护事务内存任务的 Write-Set 和 Read-Set, 分别使用 L1 Cache 和 L3 Cache 的存储空间[28, 32, 34]。当 Cache 存储空间不足时, 就会发生事务回滚。由于在 Intel CPU 上, 所有核共享使用 L3 Cache; 所以, 基于该特征, 可以探测其他进程的 Cache 使用、实施侧信道攻击。

Prime+Abort 攻击[34]的大致步骤与其他利用共享 L3 Cache 的侧信道攻击[58-59]类似: 对于要探测的内存地址, 先生成相应的访问地址集合: 当探测地址被访问时, 会导致访问地址集合中的某一个地址被逐出 Cache。然后, 攻击进程以事务内存模式运行, 并读取访问地址集合中的所有地址, 当被攻击进程访问了相应的内存地址时, 就会因 L3 Cache 存储空间不足、Read-Set 中数据被逐出而事务回滚。相比其他利用共享 L3 Cache 的类似攻击[58-59], Prime+Abort 攻击的显著优势是不需要准确的定时操作; 因为 Read-Set 中数据被逐出的事务回滚是 Intel TSX 特性自动触发的, 所以攻击进程不需要定时地检查访问地址集合的 Cache 状态。此外, 该工作还同时讨论了利用 Write-Set 数据的攻击可行性。

从另一方面看, 因为 Intel TSX 事务内存任务执行过程中相关的数据都会载入到 Cache 中, 所以也可以防御 Cache-based 侧信道攻击对敏感数据的访问分析[29]。Cloak [35]全面地论证了在云计算、本地计算和 SGX Enclave 等执行环境中, 如何利用 Intel TSX 特性、以事务内存模式执行敏感数据计算, 将代码和数据都保持在 Cache 中, 从而抵抗各种类型的 Cache-based 侧信道攻击。相比已有方案, Cloak 具有更好的通用性。

4 事务内存机制的特性分析与系统安全研究展望

以上我们总结了目前基于事务内存机制的系统安全研究成果, 下面我们对其进行技术总结, 并分析事务内存机制与各种系统安全方案之间的关系, 讨论将来基于事务内存机制的系统安全研究。

4.1 事务内存机制与系统安全方案的关系

表 1 中总结了上述系统安全研究成果与事务内存机制的关系。

以上研究成果涉及了系统安全研究中的各个重要方面。综合以上, 事务内存机制以及 Intel TSX 特性在以下方面提供了系统安全研究的重要基础条件, 可以组合得到各种防御或攻击方案。

- 软件或硬件支持的事务执行: 能够提供多线程/多任务的高效率执行, 是实用安全方案的条件之一。
- 硬件锁机制: Intel TSX 特性的 HLE 方式, 同时也实现了硬件锁机制。各种基于锁机制的安全方案, 都可以考虑基于 Intel TSX 特性的改进实现。
- 内存访问触发的、进程/线程/任务之间的信号: 某种程度而言, 在进程/线程/任务之间, Intel TSX 特性提供了硬件实现的通信机制。
- 内存访问的跟踪: 实时跟踪任务执行过程中的所有内存数据访问, 是内存相关攻击和防御中的有用条件。
- 自动的操作回滚: 事务内存任务发生数据冲突或者其他异常时, 回滚到事务开始执行时刻的状态、所有更新的数据被清除; 可用于清除攻击/错误的效果或痕迹。
- 针对操作系统的应用层隔离: Intel TSX 特性的 RTM 方式的事务回滚处理, 使得操作系统不能处理某些应用层相关的内存访问异常。在下文, 我们对此还有更进一步的讨论。
- 在 Cache 中执行所有操作: 现有的硬件事务内存机制都是基于 Cache 或者是其他 CPU 片上存储来实现。对于 Cold-Boot 攻击和 Cache-based 时序攻击, 有直接的防御效果。
- 探测其他的进程/线程/任务的 Cache 使用:

当 Write-Set 和 Read-Set 中数据因 Cache 存储空间不足或其他原因、被逐出时, 内存事务任务回滚。对于共享 Cache 场景, 可以构造 Cache-based 攻击或者防御方案。

表 1 系统安全研究方案与事务内存机制的关系

Table 1 The relationship between system security and transactional memory

系统安全方案	与事务内存机制的关系	备注
TSX-CFI	1. 利用 RTM 方式的事务内存任务不可嵌套, 实现基于锁机制的粗粒度控制流完整性; 2. 利用 HLE 方式在进入和退出临界区的数值检查, 实现代码跳转的细粒度控制流完整性; 利用 Intel TSX 特性的事务回滚处理, 清除攻击效果。	
TMI	利用软件事务内存机制实现中的 Write-Set 和 Read-Set 的内存检查, 确保授权策略的实施。	TMI 需要扩展事务内存机制的提交和异常处理的逻辑, 只能直接应用在软件事务内存机制或者是将来的、功能更为灵活的硬件事务内存机制; 对于 Intel TSX 特性等硬件事务内存机制, 需要有更深入的设计。
TxIntro	利用事务内存机制的数据冲突检测, 实现高效实时的虚拟机自省监控触发。利用 Intel TSX 特性的数据冲突检测粒度(以 Cache Line 为单位, 64 字节), 实现更细粒度的、虚拟机自省数据监控。	
Mimosa	1. 利用事务内存机制的数据冲突检测和事务回滚处理, 防御各种内存信息泄露攻击; 2. 利用 Intel TSX 特性事务内存任务在 Cache 中执行的特征, 防御 Cold-Boot 攻击。	攻击进程(或线程)和防御进程(或线程)分别是引发事务内存任务的数据冲突的 2 方。
HAFT	利用事务内存机制的事务回滚处理, 自动地完成错误恢复。利用 Intel TSX 特性提供的事务回滚指令, 基于硬件特性、完成了高效率的解决方案。	
T-SGX	利用 Intel TSX 特性的事务回滚处理, 使得恶意操作系统 Page-fault 侧信道攻击无效。	利用了 Intel TSX 特性的相同特征: 在 RTM 方式中, 当事务回滚时, CPU 硬件直接跳转到用户态的 Fallback 函数、且该跳转不涉及任何操作系统内核处理(即使事务回滚的原因是缺页错误和内存访问异常等)。
Déjà vu	利用 Intel TSX 特性的事务回滚处理, 在用户态空间构建不受恶意操作系统影响的计时器。	
DrK	利用 Intel TSX 特性的事务回滚处理, 任意次数地探测不同地址内存的属性; 即使内存访问异常, 也不会触发操作系统的异常处理。	
Prime+Abort	利用 Intel TSX 事务内存任务执行过程中的 L3 Cache 使用, 使得被攻击进程的 L3 Cache 访问会导致攻击进程的事务回滚。	
Cloak	利用 Intel TSX 特性事务内存任务在 Cache 中执行的特征, 防御各种 Cache-based 侧信道攻击。	

4.2 讨论与展望

根据当前的研究成果, 我们认为在如下方面值得继续深入研究。首先, 对于 Intel TSX 特性的硬件事务内存实现:

- 硬件事务内存机制的设计和实现。Intel TSX 特性的回滚处理, 尤其是对于缺页错误、内存访问异常等情况, CPU 硬件直接跳转到用户态的 Fallback 函数、且该跳转不涉及任何操作系统内核处理。上述的回滚处理, 违反了现有计算机体系结构和操作系统的设计原则。事务内存任务的回滚处理, 应该区分不同的原因: 某些原因的异常(例如, 缺页错误、内存访问异常等), 应该先陷入操作系统的错误处理、然后再次跳转到用户态的 Fallback 函数; 某些原因的异常(例如, 数据冲突), 则可以直接由用户态处理。

- 在 Cache 中实现操作任务。Intel TSX 特性和 SGX 特性, 都是在 Cache 中实现操作任务。首先, 由于可用 Cache 存储资源有限, 上述特性都提升了共享 Cache 场景的 Cache-based 攻击效果、或者是降低了攻击难度, 对于 Intel TSX 特性和 SGX 特性都是如此[34,60-61]。其次, 对基于 Intel TSX 特性的方案, 留下了拒绝服务攻击的隐患, 尤其对于较大内存访问的任务。而且, 不同于 SGX Enclave 程序可以将运行中间状态加密暂存到内存芯片, 由于事务内存任务的原子性, 当 Cache 存储空间不足时就应该事务回滚、清除已完成的操作, 所以拒绝服务攻击的效果更加明显。现有的 Cache 一致性协议, 为硬件事务内存的实现提供了内存访问跟踪、数据冲突检测和状态回滚等所需的基础条件; 但是, 在 Cache 管理策略上应该考虑事务内存任务有更大的 Cache 使用优先权。

除了在高性能计算领域取代传统的锁机制、提高性能, 事务内存机制也为系统安全研究提供了新的机会[62]。在系统安全中, 应用事务内存机制(尤其

是 Intel TSX 特性)仍然还有很大的研究空间:

- 利用事务内存机制的内存访问实时跟踪, 确保授权策略和操作审计的实施。在现有 Intel TSX 特性等硬件事务内存机制的基础上, 深入设计, 将内存访问的实时跟踪能力呈现为用户态应用程序可使用的系统服务。
- 利用事务内存机制内存访问的信号触发和操作回滚, 可用于软件防调试/防篡改、恶意软件动态检测、软件攻击恢复和错误恢复等。事务内存机制通过对内存的读写操作, 相当于动态地创建了运行沙箱, 而且对沙箱之外的内存读写操作非常敏感。在已有的、破坏数据完整性和数据机密性的攻击解决方案基础上[28-29], 可以进一步考虑更多类型的攻击, 尤其是底层的内存攻击, 例如 Rootkit、SMRAM 的 Cache Poisoning 攻击等。
- Intel TSX 特性的 RTM 方式的事务回滚处理, 提供了操作系统无感知的内存访问异常操作。在此基础上, 可以考虑构造更多的恶意攻击, 分析被攻击系统的各种内存使用情况。或者, 从防御的角度, 防范恶意操作系统的攻击; 但是, 从安全模型角度而言, 应该与其他的、防范恶意操作系统的应用软件安全方案配合(目前只有 Intel SGX Enclave), 实现完整的应用软件安全。
- Intel TSX 特性提供了更细粒度(64 字节, 而不是 4k 字节的内存页)、实时的进程/线程/任务之间的通信触发机制, 可直接用于改进已有系统安全方案的性能和数据一致性, 包括入侵检测、虚拟机监控、关键资源监控等。同时, 以硬件特性为系统安全方案的技术基础, 通常具有更坚实的信任起点, 也是对现有方案的改进。
- 由于Intel TSX特性在Cache中执行事务内存任务, 可以研究更多 Cache 相关的攻击和防御技术。最为典型的场景是各种 Cache-based 侧信道攻击, 而且 Intel TSX 特性还因此导致了明显的拒绝服务攻击隐患[29,34]: 当攻击程序大量地访问内存时, 会因 Cache 存储空间竞争而导致 Write-Set/Read-Set 数据被逐出, 事务内存任务回滚。

5 总结

事务内存机制涉及内存数据访问的跟踪、数据冲突的检测、内存访问的操作回滚等与内存数据操作密切相关的步骤。2013 年, Intel 公司推出 TSX 特性, 实现了硬件事务内存机制。之后, 出现了大量基于 Intel TSX 特性的系统安全研究成果, 包括恶意软件检测、数据机密性、控制流完整性、攻击恢复和错误恢复、侧信道攻防等。

在已有研究成果的基础上, 本文分析了事务内存机制以及 Intel TSX 特性与系统安全研究之间的关系, 然后分别对事务内存机制的硬件实现和基于事务内存机制的系统安全研究提出展望。Intel TSX 特性的回滚处理和 Cache 管理策略, 值得更进一步的讨论和完善。事务内存机制的内存访问跟踪、内存访问信号触发和内存操作回滚, 以及 Intel TSX 特性的事务回滚处理、在 Cache 中执行所有操作和硬件实现高效率, 都给系统安全留下了很大的研究空间。

致 谢 感谢自然科学基金“通用计算平台的密钥保护技术研究”和国家重点研发计划网络空间安全重点专项“基于国产密码算法的移动互联网密码服务支撑基础设施关键技术”对本文的资助。

参考文献

[1] M. Herlihy and J. Moss, “Transactional memory: Architectural support for lock-free data structures,” in *20th International Symposium on Computer Architecture* (ISCA), pp. 289-300,1993.

[2] L. Hammond, V. Wong, M. Chen, B. Carlstrom, J. Davis, B. Hertzberg, M. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, “Transactional memory coherence and consistency,” in *31st International Symposium on Computer Architecture* (ISCA), pp. 102-113, 2004.

[3] S. Ananian, K. Asanovic, B. Kuszmaul, C. Leiserson, and S. Lie, “Unbounded transactional memory,” in *11th IEEE Symposim on High-Performance Computer Architecture* (HPCA), pp. 316-327, 2005.

[4] D. Dice, Y. Lev, M. Moir, , D. Nussbaum, and M. Olszewski, “Early experience with a commercial hardware transactional memory implementation,” in *14th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), pp. 157-168, 2009.

[5] C. Jacobi, T. Slegel, and D. Greiner, “Transactional memory architecture and implementation for IBM System z,” in *45th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO), pp. 25-36, 2012.

[6] A. Wang, M. Gaudet, P. Wu, J.-N. Amaral, M. Ohmacht, C. Barton, R. Silvera, and M. Michael, “Evaluation of Blue Gene/Q hardware support for transactional memories,” in *21st International Conference on Parallel Architectures and Compilation Techniques* (PACT), pp. 127-136, 2012.

[7] B. Carlstrom, A. McDonald, H. Chafi, J.-W. Chung, C.-C. Minh, C. Kozyrakis, and K. Olukotun, “The Atomos transactional programming language,” in *ACM SIGPLAN Conference on Programming*

Language Design and Implementation (PLDI), pp. 1-13, 2006.

[8] T. Harris, S. Marlow, S. Peyton-Jones, and M. Herlihy, "Composable memory transactions," in *10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP), pp. 48-60, 2005.

[9] Y. Ni, A. Welc, A.-R. Adl-Tabatabai, M. Bach, S. Berkowits, J. Cownie, R. Geva, S. Kozhukow, R. Narayanaswamy, J. Olivier, S. Preis, B. Saha, A. Tal, and X. Tian, "Design and implementation of transactional constructs for C/C++," in *23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications* (OOPSLA), pp. 195-212, 2008.

[10] N. Shavit and D. Touitou, "Software transactional memory," *Distributed Computing*, vol.10, no. 2, pp. 99-116, 1997.

[11] M. Schindewolf, A. Cohen, W. Karl, A. Marongiu, and L. Benini, "Towards Transactional Memory Support for GCC," in *GCC Research Opportunities Workshop* (GROW), 2009.

[12] M. Herlihy, V. Luchango, and M. Moir, "A flexible framework for implementing software transactional memory," in *21st ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications* (OOPSLA), pp. 253-262, 2006.

[13] K. Moore, J. Bobba, M. Moravan, M. Hill, and D.Wood, "LogTM: Logbased transactional memory," in *12th IEEE Symposium on High Performance Computer Architecture* (HPCA), pp. 254-265, 2006.

[14] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum, "Hybrid transactional memory," in *12th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), pp. 336-346, 2006.

[15] S. Kumar, M. Chu, C. Hughes, P. Kundu, and A. Nguyen, "Hybrid transactional memory," in *11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP), pp. 209-220, 2006.

[16] R. Rajwar, M. Herlihy, and K. Lai, "Virtualizing transactional memory," in *32nd International Symposium on Computer Architecture* (ISCA), pp. 494-505, 2005.

[17] R. L. Bocchino, V. S. Adve, and B. L. Chamberlain, "Software transactional memory for large scale clusters," in *13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP), pp. 247-258, 2008.

[18] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues, "D2STM: Dependable distributed software transactional memory," in *15th IEEE Pacific Rim International Symposium on Dependable Computing* (PRDC), pp. 307-313, 2009.

[19] C. Kotselidis, M. Ansari, K. Jarvis, M. Lujan, C. Kirkham, and I.Watson, "DiSTM: A software transactional memory framework for clusters," in *37th IEEE International Conference on Parallel Processing* (ICPP), pp. 51-58, 2008.

[20] P. Romano, L. Rodrigues, N. Carvalho, and J. Cachopo. "Cloud-TM: Harnessing the cloud with distributed transactional memories," *Operating Systems Review*, vol. 44, no. 2, pp. 1-6, 2010.

[21] M. Saad, and B. Ravindran, "Snake: Control flow distributed software transactional memory," in *13th International Symposium on Stabilization, Safety, and Security of Distributed Systems* (SSS), pp. 238-252, 2011.

[22] T. Karnagel, R. Dementiev, R. Rajwar, K. Lai, T. Legler, B. Schlegel, and W. Lehner, "Improving in-memory database index performance with Intel transactional synchronization extensions," in *20th IEEE Symposium on High Performance Computer Architecture* (HPCA), pp. 476-487, 2014.

[23] D. Lupei, B. Simion, D. Pinto, M. Misler, M. Burcea, W. Krick, and C. Amza, "Transactional memory support for scalable and transparent parallelization of multiplayer games," in *5th European Conference on Computer Systems* (EuroSys), pp. 41-54, 2010.

[24] F. Zyulkyarov, V. Gajinov, O. Unsal, A. Cristal, E. Ayguad´e, T. Harris, and M. Valero, "Atomic Quake: Using transactional memory in an interactive multiplayer game server," in *14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoPP), pp. 25-34, 2009.

[25] J.-W. Chung, M. Dalton, H. Kannan, and C. Kozyrakis, "Thread-safe dynamic binary translation using transactional memory," in *14th IEEE Symposium on High-Performance Computer Architecture* (HPCA), pp. 279-289, 2008.

[26] A. Birgisson, M. Dhawan, U. Erlingsson, V. Ganapathy, and L. Iftode, "Enforcing authorization policies using transactional memory introspection," in *15th ACM Conference on Computer and Communications Security* (CCS), pp. 223-234, 2008.

[27] Intel R, "Chapter 8: Intel transactional memory synchronization extensions," *Intel Architecture Instruction Set Extensions Programming Reference*, 2012.

[28] Y. Liu, Y. Xia, H. Guan, B. Zang, and H. Chen, "Concurrent and consistent virtual machine introspection with hardware transactional memory," in *20th IEEE Symposium on High Performance Computer Architecture* (HPCA), pp. 416-427, 2014.

[29] L. Guan, J. Lin, B. Luo, J. Jing, and J. Wang, "Protecting private keys against memory disclosure attacks using hardware transactional memory," in *36th IEEE Symposium on Security and Privacy* (S&P), pp. 3-19, 2015.

[30] M. Muench, F. Pagani, Y. Shoshitaishvili, C. Kruegel, G. Vigna, and D. Balzarotti, "Taming transactions: Towards hardware-assisted control flow integrity using transactional memory," in *19th International Symposium on Research in Attacks, Intrusions, and Defenses* (RAID), pp. 24-48, 2016.

[31] Kuvaiskii, D., Faqeh, R., Bhatotia, P., Felber, P., and Fetzer, C,

"HAFT: Hardware-assisted fault tolerance," in *11th European Conference on Computer Systems* (EuroSys), pp. 25-25, 2016.

[32] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating controlled-channel attacks against enclave programs," in *24th ISOC Network and Distributed System Security Symposium* (NDSS), 2017.

[33] S. Chen, X. Zhang, M. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with Déjà Vu," in *12th ACM Asia Conference on Information, Computer and Communications Security* (AsiaCCS), pp. 7-18, 2017.

[34] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+Abort: A timer-free high-precision L3 cache attack using Intel TSX," in *26th USENIX Security Symposium*(USENIX Security), pp. 51-67, 2017.

[35] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, "Strong and efficient cache side-channel protection using hardware transactional memory," in *26th USENIX Security Symposium*(USENIX Security), pp. 217-233, 2017.

[36] Y. Jang, S. Lee, and T. Kim, "Breaking kernel address space layout randomization with Intel TSX," in *23rd ACM Conference on Computer and Communications Security* (CCS), pp. 380-392, 2016.

[37] L. Szekeres, M. Payer, T. Wei, and D. Song, "Eternal war in memory", in *34th IEEE Symposium on Security and Privacy* (S&P), pp. 48-62, 2015.

[38] J.-W. Chung, W. Baek, N. G. Bronson, J. Seo, C. Kozyrakis, and K. Olukotun, "ASeD: Availability, security, and debugging support using transactional memory," in *20th Annual ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA), pp. 366-366, 2008.

[39] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity," in *12th ACM Conference on Computer and Communications Security* (CCS), pp. 340-353, 2005.

[40] T. Bletsch, X. Jiang, and V. Freeh, "Mitigating code-reuse attacks with control-flow locking," in *27th Annual Computer Security Applications Conference* (ACSAC), pp. 353-362, 2011.

[41] T. Garfinkel, and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection." in *10th ISOC Network and Distributed System Security Symposium* (NDSS), pp. 191-206, 2003.

[42] B. D. Payne, M. de Carbone, and W. Lee, "Secure and flexible monitoring of virtual machines," in *23rd Annual Computer Security Applications Conference* (ACSAC), pp. 385-397, 2007.

[43] B. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An architecture for secure active monitoring using virtualization," in *29th IEEE Symposium on Security and Privacy* (S&P), pp. 233-247, 2008.

[44] T. Muller, F. Freiling, and A. Dewald, "TRESOR runs encryption securely outside RAM," in *20th USENIX Security Symposium*(USENIX Security), 2011.

[45] J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten, "Lest we remember: Cold boot attacks on encryption keys," in *17th USENIX Security Symposium*(USENIX Security), pp. 91-98, 2009.

[46] D. Bernstein. "Cache-timing attacks on AES", 2005.

[47] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *8th Workshop on Cryptographic Hardware and Embedded Systems* (CHES), pp. 201-215, 2006.

[48] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701-716, 2005.

[49] B. Garmany and T. Muller, "PRIME: Private RSA infrastructure for memory-less encryption," in *29th Annual Computer Security Applications Conference* (ACSAC), pp. 149-158, 2013.

[50] Y. Zhao, J. Lin, W. Pan, C. Xue, F. Zheng, and Z. Ma, "RegRSA: Using registers as buffers to resist memory disclosure attacks," in *31st International Conference on Systems Security and Privacy Protection* (IFIP SEC), pp. 293-307, 2016.

[51] T. Parker and S. Xu, "A method for safekeeping cryptographic keys from memory disclosure attacks," in *1st International Conference on Trusted Systems* (INTRUST), pp. 39-59, 2010.

[52] L. Guan, J. Lin, B. Luo, and J. Jing, "Copker: Computing with private keys without RAM," In *21st ISOC Network and Distributed System Security Symposium* (NDSS), pp.23-26, 2014.

[53] F. Haas, S.Weis, S.Metzlaff, and T. Ungerer, "Exploiting Intel TSX for fault-tolerant execution in safety-critical systems," in *29th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems* (DFT), pp. 197-202, 2014.

[54] O. Oleksenko, D. Kuvaiskii, P. Bhatotia, C. Fetzer, and P. Felber, "Efficient fault tolerance using Intel MPX and TSX," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*(DSN), 2016.

[55] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *36th IEEE Symposium on Security and Privacy* (S&P), pp. 640-656, 2015.

[56] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, "Preventing your faults from telling your secrets," in *11th ACM Symposium on Information, Computer and Communications Security* (AsiaCCS), pp. 317-328, 2016.

[57] Intel R, "Intel software guard extensions programming reference," 2014.

[58] Irazoqui, G., Eisenbarth, T., and Sunar, B, "S$A: A shared cache attack that works across cores and defies VM sandboxing and its application to AES," in *36th IEEE Symposium on Security and Privacy*(S&P), pp. 591-604, 2015.

[59] Lui, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B, “Last-level cache side-channel attacks are practical,” in *36th IEEE Symposium on Security and Privacy*(S&P), pp. 605-622, 2015.

[60] F. Brasser, U. Muller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, “Software grand exposure: SGX cache attacks are practical,” in *11th USENIX Workshop on Offensive Technologies* (WOOT), 2017.

[61] J. Gotzfried, M. Eckert, S. Schinzel, and T. Muller, “Cache attacks on Intel SGX,” in *10th European Workshop on Systems Security* (EuroSec), 2017.

[62] I. Muttik, A. Nayshtut, and R. Dementiev, “Creating a spider goat: Using transactional memory support for security,” BlackHat, 2014.

李从午 于 2012 年在北京交通大学信息安全专业获得工学学士学位。现在中国科学院信息工程研究所信息安全专业攻读工学博士学位。研究领域为数据安全与隐私。研究兴趣为基于硬件特性的数据安全保护。Email: licongwu@iie.ac.cn

林璟锵 于 2009 年在中国科学院研究生院信息安全专业获得博士学位。现任中国科学院信息工程研究所研究员。研究领域为数据安全与隐私、应用密码学, 尤其是密码技术在计算机网络系统的应用。Email: linjingqiang@iie.ac.cn

蔡权伟 于 2015 年在中国科学院信息工程研究所信息安全专业获得工学博士学位。现任中国科学院信息工程研究所助理研究员。研究领域为网络与系统安全。研究兴趣包括: 应用密码学、分布式容错系统、网络身份管理等。Email: caiquanwei@iie.ac.cn

罗勃 于 2008 年在宾州州立大学获得工学博士学位。现任中国科学院研究生院信息安全专业获得博士学位。现任堪萨斯大学信息和通信技术中心信息保护实验室副教授。研究领域为信息安全与隐私, 数据库和信息检索。Email: bluo@ku.edu