Isolation and Beyond: Challenges for System Security

Tyler Hunt
The University of Texas at Austin

Zhipeng Jia
The University of Texas at Austin

Vance Miller
The University of Texas at Austin

Christopher J. Rossbach
The University of Texas at Austin and
VMware Research Group

Emmett Witchel
The University of Texas at Austin

Abstract

System security has historically relied on hardware-provided isolation primitives. However, Meltdown [36] and Spectre [30] demonstrate that basic user/kernel isolation could be bypassed in every widely deployed ISA for decades; they are a caution to system designers who accept hardware isolation guarantees as an article of faith. Hardware isolation is fallible and should be considered fallible by software systems.

We argue that future systems should broaden their view to adopt techniques that compensate for weaknesses in hardware isolation and should secure and optimize the communication among isolated components. Changing algorithms to be data oblivious, so that their externally observable behavior is independent of their (secret) input data is one such technique. Securing communication requires that the timing and size of messages be independent of secret data, but how best to achieve that independence so as to limit performance and energy overheads will vary from application to application.

* CCS Concepts• **Security and privacy → Systems security**; **Operating systems security**;

ACM Reference Format:
Tyler Hunt, Zhipeng Jia, Vance Miller, Christopher J. Rossbach, and Emmett Witchel. 2019. Isolation and Beyond: Challenges for System Security. In *Workshop on Hot Topics in Operating Systems (HotOS '19), May 13–15, 2019, Bertinoro, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3317550.3321427

1 Introduction

Historically, software system designers have leveraged hardware isolation mechanisms as the basis for secure systems. The user/kernel processor mode bit, page tables on CPUs and GPUs, and more recently hardware-supported trusted execution environments (TEEs) like Intel's software guard extensions (SGX) are all examples of hardware-enforced isolation mechanisms. These primitives have found favor for decades because they are efficiently implemented in hardware, and have clear semantics that system software can use as the basis for security.

While isolation is and will remain an important building block for secure systems, the focus on isolation mechanisms obscures two difficult, recent lessons. First, modern hardware is highly optimized for performance which makes isolation difficult. Second, even with fully effective isolation mechanisms, secure systems will likely consist of multiple isolated but communicating environments, where the communication creates new side channels. This paper advocates redundant protections for isolated computation using a combination of software and hardware techniques. It also advocates securing and optimizing the communication among multiple, distinct isolated computations, often within the same machine.

Modern CPUs and GPUs achieve their impressive performance using numerous hardware structures shared across protection boundaries such as memory caches, memory prefetchers, and prediction structures like branch and target buffers. The number, complexity, and shared nature of these structures make isolation difficult. Isolation is a non-interference property: computational entity A's execution history (including timing) should be independent of entity B's, even if both are concurrently executed on the same physically-shared hardware. Recent security failures (e.g., Meltdown [36] and Spectre [30]) have shown how difficult it is for hardware to provide non-interference. The timing of computations using shared hardware structures leaks private data across protection boundaries. As another example, Graviton [65], a recent design for GPU-based trusted execution environments, trusts GPU memory because it cannot be snooped. However, recent side-channel attacks on GPUs [46] have shown practical methods to fingerprint websites using performance counters observed during GPU rendering in the browser, rather than by monitoring memory accesses.

While improving isolation is itself important, recent failures of isolation mechanisms should motivate systems designers to examine how computations can remain secure despite imperfect isolation. Hardware isolation is fallible and should be considered fallible by software systems. Some

HotOS '19, May 13–15, 2019, Bertinoro, Italy

ACM ISBN 978-1-4503-6727-1/19/05…$15.00
https://doi.org/10.1145/3317550.3321427

algorithms are easier to secure than others. For example, machine learning algorithms (e.g., support vector machines, matrix factorization, neural networks, decision trees, and k-means clustering) can be data oblivious: constructed so their externally visible behavior (e.g., memory references, system calls, branch behavior) does not disclose anything about their secret input data [47]. Limiting information leakage via hardware usage patterns reduces the burden on hardware to provide isolation by eliminating unintended communication. Forcing the programmer to modify their algorithms to strengthen security is burdensome but necessary to reduce reliance on side-channel-ridden isolation primitives.

We believe that secure, complex computations will be broken up across multiple TEEs on CPUs and eventually on GPUs. The principle of least privilege [56] argues for splitting monolithic systems into a number of smaller components, with deliberately simple ones responsible for security-critical functionality [16, 52]. The evolution of modern hardware support for security, from trusted platform modules (TPMs) that can verify boot to enclaves that can verify the start state of a smaller, user-level computation reflects the principle that minimizing and simplifying the trusted computing base (TCB) is valuable for security. Also, as hardware TEEs have limited resources (such as hardware thread contexts and physical memory), larger computations must communicate across TEEs: decomposition is inevitable.

Unfortunately, dividing computation among many communicating TEEs creates further side channels. An attacker with control of the operating system can transparently interpose on most communication mechanisms within a single computer (e.g., enclaves cannot share memory) and collect fine-grained, high-precision timing information. The communication of these isolated components must be secured, so an adversary cannot learn anything about the data being processed by observing the communication (which includes sizes and timings).

We survey recent attempts at providing isolation for CPUs (§3) and for GPUs (§4) and find that unintended side-channel leaks have undermined the isolation of both commercial and research systems. We then consider communication between isolated components and how to secure it efficiently (§5). Secure communication is straightforward in theory–communication patterns should be independent of secret data–but it is difficult to achieve in practice at a reasonable cost. We then discuss ways for future systems to be robust in the face of compromised isolation (§6), including changes to the programming model for secure software. Specialized programming models hold promise for achieving robust, practical system security.

2 Background and related work

System security is a broad area and this paper is not a comprehensive review. We focus on issues relevant to operating system designers for cloud services, given their popularity, requirement for multi-tenancy, and need for security. Cloud services maximize hardware utilization by sharing hardware resources among many mutually distrustful cloud tenants. Cloud tenants are isolated from each other by the hypervisor (e.g., virtual machines) or the operating system (e.g., containers). We assume cloud applications are processing private, personal data, e.g., genome information, financial information, or health records.

Side channels. Side channels are communication channels based on mechanisms that were not intended for communication. Data communication over side channels is typically unintentional and can be exploited by an attacker to extract secrets. Mechanisms for side channels can exploit any state that depends on secrets and is visible to the attacker, e.g., the timing of accesses to the CPU's cache [27, 39], power draw [31], or temperature [42]. We focus on side channels available to the observer without physical access, primarily channels based on timing.

Threat Model. We are concerned with software attacks launched by a locally-resident adversary. For cloud services, attackers can run malicious code on the same physical device as a target cloud application [55]). The attacker's privileges could be as limited as another cloud tenant's or as far-reaching as a disgruntled data center employee's. Denial-of-service attacks are out of scope. Physical attacks such as bus snooping are also out of scope.

Isolation using TEEs. Researchers and industry continue the search for strong isolation mechanisms. For example, Sanctum [9] and Keystone [34] are successive designs for TEEs for RISC-V processors. Keystone corrects some of the security problems with SGX (e.g., securing page faults) and some of the performance issues (e.g., a limited pool of BIOS-segregated enclave memory) while maintaining the SGX programming model. Ongoing work expands the threat model to some side-channel attacks, specifically cache timing attacks.

Several recently proposed systems aim to protect applications from an untrusted platform [4, 7, 13, 62] by leveraging hardware supported TEEs [19, 57, 75].

GPU Security and Protection. GPU security properties, vulnerabilities of GPUs [79], and techniques to exfiltrate data from the GPU [49] are increasingly well-explored. PixelVault [64] exploits physical isolation between CPUs and GPUs to implement secure key storage for keys, although it was shown to be insecure [79]. Attacks leveraging GPU memory reuse without re-initialization [17, 35, 78] are a common theme. Techniques to isolate malicious device drivers [8], protect the system from malicious accelerators [48], or provide trusted I/O paths for accelerators [29, 71, 77] are applicable to securing GPUs as well.

Graviton [65] provides TEEs on GPUs using cryptographically secured communication, and relying on the GPU command processor to protect memory from other concurrently active contexts. HIX [23] extends an SGX-like design with support for secure MMIO to GPUs to enable enclave access to GPUs. Neither HIX or Graviton make communication patterns with the GPU data oblivious.

Microarchitectural side-channel attacks. CPU side-channel attacks [12, 18, 27, 39, 43, 68, 74] are well-studied, leading to defenses based on static or dynamic partitioning of caches [11, 21, 22, 53, 69], OS- and compiler-supported cache line locking [33, 69], randomization of replacement [70] and fill [38] policies, timing noise injection [41], or managing traffic at the memory controller [59, 67, 76]. Covert Channels using shared GPU hardware [45] are a nascent area, including AES Key extraction using shared GPU hardware [24, 25]. Sub-warp randomization techniques to obscure timing relationships between execution and memory accesses [24] have been proposed to alleviate correlation-based timing attacks [26].

3 Isolation on the CPU

CPUs provide distinct virtual address spaces managed by the operating system (and the hypervisor) to provide isolation. Relying on address space separation requires trusting the operating system and hardware. Remote users can verify that the operating system provided by the cloud is not malicious by using a TPM for trusted boot (e.g., Google developed Titan, a custom chip that provides trusted boot [61]). But the initial state of an operating system is not a strong indicator of the system's security. Operating systems are complex, constantly changing, and constantly under attack. The national vulnerability database lists 207 critical vulnerabilities (9 or 10 on CVSS V3 scale) in 2016 and 127 in 2017.

ARM's TrustZone [3] creates two worlds–a secure world and a normal world–enforced by privilege levels of the CPU. Software running in the secure world can access memory that is not accessible by the normal world, and can use trusted devices (which are beyond the capabilities of TPMs) over a secure bus. TrustZone has advantages over trusted boot and TPMs, but it shares many of their limitations by subsuming the entire OS into the TCB. An OS is too large a software unit for secure systems.

As a response to the relatively weak guarantee of a whole trusted operating system, recent hardware supports protection for small, user-level components. Intel's SGX and RISC-V's Keystone enclaves are examples of this "trusted execution environment"(TEE) abstraction. The hardware generates a secure certificate that identifies the initial code and data of an enclave so a remote user can be convinced the expected code is running on a legitimate hardware platform[1]. Remote attestation for the initial state of enclaves is similar to trusted boot for operating systems. Enclaves can run on any CPU core at any time. They can make system calls, as long as they copy the arguments out of the enclave and the untrusted result back into the enclave. Enclaves run unprivileged code, so the OS and hypervisor have control over their resource usage, including the number of physical memory frames they occupy.

Both address space separation and TEEs fail to provide true isolation since an attacker may still observe the isolated execution's effect on shared hardware resources via timing [30, 36]. Addressing side channels in Keystone is ongoing work [34].

Violating SGX isolation with page faults. Providing hardware isolation for enclave data while allowing privileged software free reign to manage memory has created a security problem for enclaves [73]. The operating system can mark pages as not present, creating hardware page faults that the OS must handle, allowing it to receive a page-granularity trace of accesses to enclave code and data. This coarse-granularity trace is sufficient to recover the fine-grained content of many data structures (e.g., hash tables) given that application code is public (or can be reverse engineered).

The attack's designers do not offer a comprehensive fix and Intel has not included one in the specification for version two of the SGX hardware. The attack's designers note that allowing the enclave to pin all code pages in memory would defeat all of their attacks, but they do not know if it is sufficient to defeat all attacks. Allowing unprivileged code to pin memory pages in the hardware undercuts the functionality of privileged software and can be the basis for denial of service attacks. Sanctum and Keystone enclaves route page faults to the enclave itself, which increases security, but both systems reduce the flexibility of the OS/hypervisor to manage memory [9, 34].

Making application memory references data oblivious thwarts the page fault monitoring attack while also closing cache side channels. There are algorithm-independent methods for making memory references data oblivious (e.g., oblivious RAM (ORAM) [15]), but they tend to have high overhead; PHANTOM [40], which uses an optimized FPGA implementation of ORAM, still incurs a slowdown of 14.7×. GhostRider [37] uses a compiler to optimize application code for ORAM, reducing the slowdown to 10.68×. Hyperflow [14] eliminates side channels using hardware-supported information flow. Some workloads are slowed down significantly, some are not, but the performance implications are difficult to extrapolate to a high-performance processor. Data-oblivious algorithms [47, 66] and data structures (e.g., priority queues [20]) can be more efficient than these alternatives, but require programmer effort.

[1] The part of Intel's remote attestation protocol that validates the hardware platform additionally requires a software service.

4 Isolation on the GPU

Discrete GPUs are the *de facto* accelerator of choice for important workloads like neural networks. While the hardware organization of GPUs isolates them from the CPU, they remain under control of driver software running on the host CPU. Leveraging their physical isolation for security is tricky.

Whole GPU. Early attempts to use a GPU as an isolated secure coprocessor [64] were unsuccessful [79]. GPUs lack a clear software/hardware boundary *by design*, to provide vendors with flexibility and compatibility over successive generations. The resulting porous boundary creates the security problem that a non-bypassable hardware feature in one version of a GPU can become a bypassable software feature in another version. Also, GPUs are less isolated from CPUs then they appear: PCIe-attached GPUs rely heavily on MMIO to expose command queues and registers and the IOMMU enforces their isolation. The IOMMU can be reconfigured at any time by untrusted privileged software, so the IOMMU is not a reliable isolation mechanism against privileged software.

Enclaves. It is natural to wonder if the protected enclaves on CPUs can be implemented for GPUs. HIX [23] extends an SGX design with a duplicate copy of all the memory protection hardware to enable the hardware to guarantee that a single enclave has exclusive access to MMIO regions exported by a GPU. This, in principle, defeats a malicious OS that wants to interpose or create its own mappings to them. While this design provides stronger isolation than what current enclaves can achieve for the GPU, it remains vulnerable to the same set of side-channel attacks described in §3, because communication is not data oblivious.

Graviton [65] postulates changes in GPU firmware (and some GPU hardware) to create enclaves called "secure contexts." Secure contexts are protected by a Graviton-enabled GPU, which enforces rules when manipulating GPU page tables to enforce its isolation guarantees.

An application creates a secure context by establishing an encrypted and authenticated communication channel with the GPU. The kernel driver is not trusted. The GPU requires authentication from the application to approve all changes to memory mappings and only accepts commands and returns results for a secure context over the associated secure channel.

A fundamental challenge for Graviton is that it supports multiple, distrustful principals concurrently using the GPU, creating the possibility of attacks which leverage shared resource contention. Contention on shared GPU resources (caches, functional units, and memory) has been used to create covert channels between otherwise isolated GPU contexts [45]. Side channels based on shared resource contention were also used to fingerprint websites, and infer parameters about neural networks [46]. On GPUs too, shared hardware resources make isolation difficult.

5 Securing communication

Finer-grained protection granularity from hardware (e.g., SGX) and impossible-to-ignore performance gains from accelerators (e.g., GPUs) are driving application design toward a system of communicating components. Independent of the strength of isolation achievable on CPUs and GPUs, isolated components currently must communicate over an untrusted communication medium.

Also, as a practical consideration, current enclave users may wish to split their applications across multiple enclaves for performance. Current SGX enclaves can only use 128MB of physical memory per CPU. This is a limitation of the implementation due to the size of on-chip data structures that track enclave physical memory pages. While this limitation should be reduced in future hardware revisions, it has been a limitation since SGX became available in 2015.

Using encrypted channels to communicate protects the secrecy and integrity of every message, but the size and the timing of those messages (i.e., the communication pattern) are still available to untrusted code. Communication patterns can reveal secrets. For a network communication example, Schuster et al. were able to infer the movie being streamed over an encrypted connection [58] based on sizes and timing of encrypted packets.

CPU/Enclave and GPUs. Software running on the CPU communicates with the GPU over the PCIe bus, mostly moving data or controlling the execution of GPU programs (kernels). The PCIe bus is generally under the control of the hypervisor and/or host operating system, and routes packets to multiple devices connected to the PCIe root complex in a tree topology, so packets in transit to/from the GPU may be visible to other devices. In fact, the host software may change the routing topology dynamically and can install pseudo-devices that allow it to sniff traffic.

Even simple observations of encrypted traffic to and from the GPU can leak critical pieces of information like kernel execution time. For instance, if the application sends a large batch of data to the GPU and no other messages are exchanged until the GPU has finished processing the batch, the attacker can trivially compute the execution time for that batch. Using execution time, Jiang et al. were able to extract AES-128 encryption keys from encryption kernels running on the GPU by reasoning about the number of unique cache line fetches [24] and the number of memory bank stalls [25].

Oblivious communication. There is really only a single technique to secure communication—communication patterns must be made *oblivious* of any secret data. A simple way to make communication oblivious to secret data is to establish a schedule for sending and receiving messages that is independent of secret data. As an example, consider a fixed-rate schedule (shown in Figure 1) in which an application sends an encrypted message of a particular size after a constant amount of time.

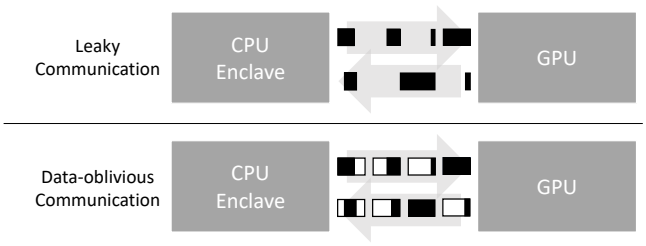

Figure 1. Leaky communication is made secure by splitting/padding messages into a fixed size (black data with white padding) and sending them at fixed rate. Messages and padding are indistinguishable to the attacker because all communication is encrypted.

Using a fixed-rate schedule is secure and enables a range of interesting system engineering tradeoffs. One example tradeoff is how to optimize for latency. For example, application A could adopt a fixed-rate schedule of sending an 8KB message every 5 milliseconds; actual data is divided between messages when it is larger than 8KB, and messages are padded to 8KB when there is not enough data. Such a schedule consumes 1.6MB/s of bandwidth. However, if application B has control messages that benefit from sub-5-ms transmission time, a schedule of 4KB every 2.5ms would consume identical bandwidth but would be preferable.

Note that tuning the fixed-rate communication schedule to a computation does leak information about that computation. In our example, both fixed-rate schedules consume identical bandwidth, but the timing leaks whether application A or B is running. This level of information leakage may or may not be acceptable for a given use.

Another challenge with securing communication by fixing rates is minimizing cost (e.g., in performance or energy consumption). Our example fixed-rate schedule will consume energy to encrypt and transport unused parts (or whole) messages that will only be discarded. New hardware mechanisms could reduce the energy costs of reserving bandwidth. For example, hardware could hide information about how many CPU enclaves are communicating with the GPU by providing a fixed bandwidth to the GPU to each enclave. Any such hardware would need to make difficult tradeoffs of safety versus utilization.

Another way to minimize the cost of securing communication is to weaken security guarantees. Hermetic is an enclave-based analytics system that is differentially private in the face of important side channels including timing [72]. Askarov et al. provide a timing channel mitigation scheme that trades information leakage rather than resource utilization for performance [5].

6 Securing future systems

Faced with frequent and high-profile failures of hardware-supported isolation, we ask whether it is possible to build systems whose security properties depend less on isolation, or that can still provide meaningful, perhaps degraded guarantees when hardware isolation is compromised. The more pervasively secret-oblivious techniques are used at all stack layers, the less likely it is that observable behaviors can enable unintended communication that exposes secrets.

Future work in secure systems will surely include hardware isolation mechanisms, but it will broaden to include communication mechanisms and also software/programming model changes.

Improving enclaves. Improved enclave support that lifts restrictions (e.g., limited enclave memory size) in current implementations will make it easier to build secure systems. For example, Keystone enclaves can be built from system memory without requiring a BIOS-sequestered enclave page cache like SGX does.

While both Sanctum [9] and Keystone [34] address side channel information leakage as an explicit design goal (at least for cache timing side channels), it is notable that after years of effort, side channel mitigation remains an ongoing effort for Keystone.

Hardware to make isolation easier. Given the end of Dennard scaling and the limited range of options modern architects have to optimize performance, hardware manufacturers are likely to continue to prioritize performance over security. To increase security, hardware could expand to include redundant but isolated structures, such as multiple memory controllers, isolated or partitioned caches, or multiple branch predictors.

Additionally, hardware might expose a "slow but secure" mode that includes features like fixed-latency floating point operations, no cache prefetching, and no branch prediction. Use of such a hardware mode would likely be detectable by an adversary and the performance penalty would likely be severe enough to motivate developers to minimize its use. However, bear in mind that there are significant performance costs to current software remediation for side channel attacks (e.g., KPTI [28]).

Restricting access to time. To extract information from timing side channels the attacker must learn the time at which events occur, or the time between events. Intuitively, denying the attacker exact information about when an event occurred could prevent or mitigate leaks. The problem is that modern computer platforms have a large variety of resources that can help reconstruct timing: high-precision, user-level CPU counters (e.g., `rdtsc` on Intel processors), explicit system-level resources like OS timekeeping APIs, and implicit resources like file-system timestamps and networked communication. For example, to make it harder for Javascript code to exploit Spectre and Meltdown, Chrome and other web browsers have addressed explicit and implicit timing sources. They decreased

the resolution of `performance.now()` and they have disabled `SharedArrayBuffer` which allows a dedicated worker thread to increment a counter regularly enough to act as a high-precision time source for another thread [60].

Enforcing principled fuzzing of all time sources can significantly reduce the channel bandwidth [32]. But such an approach must be completely comprehensive. Any overlooked time source negates the value of fuzzing. Adding noise is entirely insufficient. As measurements across a network show, noise is highly vulnerable to filtering, providing little security value [10]. Unfortunately, restricting access to accurate time sources is a brittle defense.

Software-visible remediation. Security is notorious as a goal that everyone wants but no one wants to pay for. One of the heaviest costs for software systems is software-visible changes. These could be changes to the toolchain such as compilers or even programmer-visible changes including programming languages or programming models.

Retpoline [63] is an example of a software-visible change that prevents Spectre variant 2 (which can be used by user-level code to steal kernel-level secrets). It performs better than disabling indirect branch speculation in hardware. However, it relies on specific microarchitectural details of speculation, so it is only secure on AMD processors and Intel Broadwell (and earlier) processors [44].

Cryptographic codes have made algorithmic changes to eliminate side channels. Code that branches or accesses memory based on the secret key can end up leaking the key itself [6]. Even widely deployed hardware functionality like floating point operations can be too dangerous to use because of the timing channels they create [1], although there is work on a constant time standard [2].

We anticipate this trend will continue, with security-sensitive code using specialized libraries and possibly toolchains. Widely deployed cryptography [51] and other security-sensitive [50] libraries have the explicit goal of being constant time (execution time is independent of input data). While this strategy can be effective and reasonably efficient, it only applies to certain sensitive codes, like cryptographic codes, that justify the extensive programmer effort.

Algorithms whose memory access patterns and whose communication patterns are independent of their processing requirements are more robust to isolation violations [47]. Oblivious algorithms also require toolchain support to make sure the compiler does not undermine the programmer's intent, and they rely on assumptions about the hardware, e.g., fixed-time floating point operations. Raccoon [54] is an example of toolchain support, where a compiler transforms normal program source to equivalent source that appears to execute all possible program paths. While the cost of security is still significant, Raccoon reduces slowdowns of the ORAM-based GhostRider [37] from 195× to 21.8×.

7 Conclusion

Computer use continues to permeate society, raising the importance of secure systems. While some recent security problems have been shocking in their severity, there has also been significant progress in understanding and efficiently addressing information leakage via side channels. As we progress to a world of communicating, protected computational environments, we should use this understanding to build systems that are secure, efficient, and robust to partial failures in security assumptions.

Acknowledgments

We thank our shepherd Brad Karp and we thank Hovav Shacham for help with references. This research was supported in part by NSF grant CNS-1618563.

References

[1] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 623–639, Washington, DC, USA, 2015. IEEE Computer Society.

[2] Marc Andrysco, Andres Nötzli, Fraser Brown, Ranjit Jhala, and Deian Stefan. Towards verified, constant-time floating point operations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1369–1382, New York, NY, USA, 2018. ACM.

[3] Arm Limited. Introducing Arm TrustZone. https://developer.arm.com/technologies/trustzone. (Accessed: January 2019).

[4] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, David Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. Scone: Secure Linux containers with Intel SGX. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 689–703, Berkeley, CA, USA, 2016. USENIX Association.

[5] Aslan Askarov, Danfeng Zhang, and Andrew C. Myers. Predictive black-box mitigation of timing channels. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 297–307, New York, NY, USA, 2010. ACM.

[6] Gilles Barthe, Gustavo Betarte, Juan Campo, Carlos Luna, and David Pichardie. System-level non-interference for constant-time cryptography. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1267–1279, New York, NY, USA, 2014. ACM.

[7] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding applications from an untrusted cloud with haven. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 267–283, Berkeley, CA, USA, 2014. USENIX Association.

[8] Silas Boyd-Wickizer and Nickolai Zeldovich. Tolerating malicious device drivers in Linux. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 9–9, Berkeley, CA, USA, 2010. USENIX Association.

[9] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *USENIX Security Symposium*, 2016.

[10] Scott A. Crosby, Dan S. Wallach, and Rudolf H. Riedi. Opportunities and limits of remote timing attacks. *ACM Transactions on Information and System Security*, 12(3):17:1–17:29, January 2009.

[11] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Trans. Archit. Code Optim.*, 8(4):35:1–35:21, January 2012.

[12] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Understanding and mitigating covert channels through branch predictors. *ACM Trans. Archit. Code Optim.*, 13(1):10:1–10:23, March 2016.

[13] Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 287–305, New York, NY, USA, 2017. ACM.

[14] Andrew Ferraiuolo, Mark Zhao, Andrew C. Myers, and G. Edward Suh. HyperFlow: A processor architecture for nonmalleable, timing-safe information flow security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1583–1600, New York, NY, USA, 2018. ACM.

[15] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 182–194, New York, NY, USA, 1987. ACM.

[16] Munawar Hafiz, Ralph Johnson, and Raja Afandi. The security architecture of qmail. In *Proceedings of the 11th Conference on Patterns Language of Programming (PLoP'04)*. Citeseer, 2004.

[17] Ari B. Hayes, Lingda Li, Mohammad Hedayati, Jiahuan He, Eddy Z. Zhang, and Kai Shen. GPU taint tracking. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '17, pages 209–220, Berkeley, CA, USA, 2017. USENIX Association.

[18] Casen Hunger, Mikhail Kazdagli, Ankit Singh Rawat, Alexandros G. Dimakis, Sriram Vishwanath, and Mohit Tiwari. Understanding contention-based channels and using them for defense. In *HPCA*, pages 639–650. IEEE Computer Society, 2015.

[19] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 533–549, Berkeley, CA, USA, 2016. USENIX Association.

[20] Zahra Jafargholi, Kasper Green Larsen, and Mark Simkin. Optimal oblivious priority queues and offline oblivious RAM. Cryptology ePrint Archive, Report 2017/452, 2019. https://eprint.iacr.org/2019/237.pdf.

[21] Aamer Jaleel, Eric Borch, Malini Bhandaru, Simon C. Steely Jr., and Joel Emer. Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (TLA) cache management policies. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, pages 151–162, Washington, DC, USA, 2010. IEEE Computer Society.

[22] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. High performance cache replacement using re-reference interval prediction (rrip). In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 60–71, New York, NY, USA, 2010. ACM.

[23] Insu Jang, Adrian Tang, Taehoo Kim, Simha Sethumadhavan, and Jaehyuk Huh. Heterogeneous Isolated Execution for Commodity GPUs. In *ASPLOS*, 2019.

[24] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. A complete key recovery timing attack on a GPU. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2016.

[25] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. A novel side-channel timing attack on GPUs. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI '17, pages 167–172, New York, NY, USA, 2017. ACM.

[26] Gurunath Kadam, Danfeng Zhang, and Adwait Jog. Rcoal: Mitigating GPU timing attack via subwarp-based randomized coalescing techniques. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*, pages 156–167, 2018.

[27] Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. A high-resolution side-channel attack on last-level cache. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 72:1–72:6, New York, NY, USA, 2016. ACM.

[28] Kernel page-table isolation. https://en.wikipedia.org/wiki/Kernel_page-table_isolation. (Accessed: January 2019).

[29] Yonggon Kim, Ohmin Kwon, Jinsoo Jang, Seongwook Jin, Hyeongboo Baek, Brent Byunghoon Kang, and Hyunsoo Yoon. On-demand bootstrapping mechanism for isolated cryptographic operations on commodity accelerators. 62, 07 2016.

[30] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *CoRR*, January 2018.

[31] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, Berlin, Heidelberg, 1999. Springer-Verlag.

[32] David Kohlbrenner and Hovav Shacham. Trusted browsers for uncertain times. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 463–480, Berkeley, CA, USA, 2016. USENIX Association.

[33] Jingfei Kong, Onur Aciiçmez, Jean-Pierre Seifert, and Huiyang Zhou. Hardware-software integrated approaches to defend against software cache-based side channel attacks. In *HPCA*, pages 393–404. IEEE Computer Society, 2009.

[34] Dayeol Lee, David Kohlbrenner, Kevin Cheang, Cameron Rasmussen, Kevin Laeufer, Ian Fang, Akash Khosla an Chia-Che Tsai, Sanjit Seshia, Dawn Song, and Krste Asanovic. Keystone enclave: An open-source secure enclave for RISC-V. https://keystone-enclave.org/files/keystone-risc-v-summit.pdf, 2018. (Accessed: April 2019).

[35] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. Stealing webpages rendered on your browser by exploiting GPU vulnerabilities. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 19–33, Washington, DC, USA, 2014. IEEE Computer Society.

[36] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Dkaniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *CoRR*, January 2018.

[37] Chang Liu, Austin Harris, Martin Maas, Michael Hicks, Mohit Tiwari, and Elaine Shi. Ghostrider: A hardware-software system for memory trace oblivious computation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 87–101, New York, NY, USA, 2015. ACM.

[38] Fangfei Liu and Ruby B. Lee. Random fill cache architecture. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 203–215, Washington, DC, USA, 2014. IEEE Computer Society.

[39] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 605–622, Washington, DC, USA, 2015. IEEE Computer Society.

[40] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. PHANTOM: Practical oblivious computation in a secure processor. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 311–324, New York, NY, USA, 2013. ACM.

[41] Robert Martin, John Demme, and Simha Sethumadhavan. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *ISCA*, pages 118–129. IEEE Computer Society, 2012.

[42] Ramya Jayaram Masti, Devendra Rai, Aanjhan Ranganathan, Christian Müller, Lothar Thiele, and Srdjan Capkun. Thermal covert channels on multi-core platforms. In *USENIX Security Symposium*, 2015.

[43] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. Hello from the other side: SSH over robust cache covert channels in the cloud. In *NDSS*. The Internet Society, 2017.

[44] Mehmet_lyigun. Mitigating spectre variant 2 with retpoline on windows. https://techcommunity.microsoft.com/t5/Windows-Kernel-Internals/Mitigating-Spectre-variant-2-with-Retpoline-on-Windows/ba-p/295618, 2018. (Accessed: April 2019).

[45] Hoda Naghibijouybari, Khaled N. Khasawneh, and Nael Abu-Ghazaleh. Constructing and characterizing covert channels on GPGPUs. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[46] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: GPU side channel attacks are practical. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[47] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Sebastian Nowozin Aastha Mehta, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, 2016.

[48] Lena E. Olson, Jason Power, Mark D. Hill, and David A. Wood. Border control: Sandboxing accelerators. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 470–481, New York, NY, USA, 2015. ACM.

[49] Roberto Di Pietro, Flavio Lombardi, and Antonio Villani. CUDA leaks: A detailed hack for CUDA and a (partial) fix. *ACM Trans. Embed. Comput. Syst.*, 15(1):15:1–15:25, January 2016.

[50] Thomas Pornin. Constant-time toolkit. https://github.com/pornin/CTTK. (Accessed: January 17, 2019).

[51] Thomas Pornin. Why constant-time crypto? https://www.bearssl.org/constanttime.html. (Accessed: January 17, 2019).

[52] Niels Provos, Markus Friedl, and Peter Honeyman. Preventing privilege escalation. In *USENIX Security Symposium*, 2003.

[53] Moinuddin K. Qureshi and Yale N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO*, pages 423–432. IEEE Computer Society, 2006.

[54] Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: Closing digital side-channels through obfuscated execution. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 431–446, Berkeley, CA, USA, 2015. USENIX Association.

[55] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[56] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, volume 63, 1975.

[57] Felix Schuster, Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy data analytics in the cloud using SGX. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2015.

[58] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *USENIX Security Symposium*, 2017.

[59] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. Avoiding information leakage in the memory controller with fixed service policies. In *MICRO*, pages 89–101. ACM, 2015.

[60] Surma. Meltdown/spectre. https://developers.google.com/web/updates/2018/02/meltdown-spectre. (Accessed: April 2019).

[61] Titan in depth: Security in plaintext. https://cloud.google.com/blog/products/gcp/titan-in-depth-security-in-plaintext. (Accessed: April 2019).

[62] Chia-Che Tsai, Donald E. Porter, and Mona Vij. Graphene-sgx: A practical library os for unmodified applications on sgx. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '17, pages 645–658, Berkeley, CA, USA, 2017. USENIX Association.

[63] Paul Turner. Retpoline: a software construct for preventing branch-target-injection. https://support.google.com/faqs/answer/7625886. (Accessed: April 2019).

[64] Giorgos Vasiliadis, Elias Athanasopoulos, Michalis Polychronakis, and Sotiris Ioannidis. PixelVault: Using GPUs for Securing Cryptographic Operations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1131–1142, New York, NY, USA, 2014. ACM.

[65] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on GPUs. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[66] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In *ACM Conference on Computer and Communications Security (CCS)*, 2014.

[67] Yao Wang, Andrew Ferraiuolo, and G. Edward Suh. Timing channel protection for a shared memory controller. In *HPCA*, pages 225–236. IEEE Computer Society, 2014.

[68] Zhenghong Wang and Ruby B. Lee. Covert and side channels due to processor architecture. In *22nd Annual Computer Security Applications Conference (ACSAC 2006), 11-15 December 2006, Miami Beach, Florida, USA*, pages 473–482, 2006.

[69] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 494–505, New York, NY, USA, 2007. ACM.

[70] Zhenghong Wang and Ruby B. Lee. A novel cache architecture with enhanced performance and security. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 83–93, Washington, DC, USA, 2008. IEEE Computer Society.

[71] Samuel Weiser and Mario Werner. SGXIO: Generic Trusted I/O Path for Intel SGX. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 261–268, New York, NY, USA, 2017. ACM.

[72] Min Xu, Antonis Papadimitriou, Ariel Feldman, and Andreas Haeberlen. Using differential privacy to efficiently mitigate side channels in distributed analytics. In *ACM European Conference in Computer Systems (EuroSys)*, 2018.

[73] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2015.

[74] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. Are coherence protocol states vulnerable to information leakage? In *HPCA*, pages 168–179. IEEE Computer Society, 2018.

[75] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[76] Yanqi Zhou, Sameer Wagh, Prateek Mittal, and David Wentzlaff. Camouflage: Memory traffic shaping to mitigate timing attacks. In *HPCA*, pages 337–348. IEEE Computer Society, 2017.

[77] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune. Building verifiable trusted path on commodity x86 computers. In *2012 IEEE Symposium on Security and Privacy*, pages 616–630, May 2012.

[78] Zhe Zhou, Wenrui Diao, Xiangyu Liu, Zhou Li, Kehuan Zhang, and Rui Liu. Vulnerable GPU memory management: Towards recovering raw data from GPU. *PoPETs*, 2017(2):57–73, 2017.

[79] Zhiting Zhu, Sangman Kim, Yuri Rozhanski, Yige Hu, Emmett Witchel, and Mark Silberstein. Understanding the security of discrete GPUs. In *Proceedings of the General Purpose GPUs*, GPGPU-10, pages 1–11, New York, NY, USA, 2017. ACM.