Sharing is leaking: blocking transient-execution attacks with core-gapped confidential VMs

Charly Castes
EPFL & Google
Lausanne, Switzerland

Andrew Baumann
Google
Zurich, Switzerland

Abstract

Confidential VMs on platforms such as Intel TDX, AMD SEV and Arm CCA promise greater security for cloud users against even a hypervisor-level attacker, but this promise has been shattered by repeated transient-execution vulnerabilities and CPU bugs. At the root of this problem lies the need to multiplex CPU cores with all their complex microarchitectural state among distrusting entities, with an untrusted hypervisor in control of the multiplexing.

We propose *core-gapped confidential VMs*, a set of software-only modifications that ensure that no distrusting code shares a core, thus removing all same-core side-channels and transient-execution vulnerabilities from the guest's TCB. We present an Arm-based prototype along with a performance evaluation showing that, not only does core-gapping offer performance competitive with non-confidential VMs, the greater locality achieved by avoiding shared cores can even improve performance for CPU-intensive workloads.

ACM Reference Format:
Charly Castes and Andrew Baumann. 2024. Sharing is leaking: blocking transient-execution attacks with core-gapped confidential VMs. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3622781.3674190

1 Introduction

Since Spectre [37] and Meltdown [40] in 2018, CPU security isolation has been repeatedly broken by a series of transient-execution vulnerabilities [3, 10, 16, 22, 26–29, 36, 42, 44, 48, 53–55, 57, 61, 62, 65, 67, 69–72, 75, 76, 78] and CPU bugs that break isolation [14, 32, 51, 52, 66, 79] and the flood of vulnerabilities shows no sign of stopping (see fig. 3 in §2.2). For the worst problems, mitigations have been developed in the form of microcode fixes and software patches, but these often induce substantial performance overheads. Because these vulnerabilities stem from microarchitectural designs that seek to maximise single-threaded performance through aggressive speculation and out-of-order execution [59], we have no reason to doubt that they will continue.

The same period has seen increasing adoption by cloud providers and customers of *confidential VMs* (CVMs) on platforms like AMD SEV-SNP, Intel TDX, and Arm CCA. These architectures encrypt guest memory and enable the removal of the hypervisor and host stack from a cloud guest's trusted computing base (TCB). In principle, the guest TCB is reduced solely to the CPU hardware and remotely-attested platform firmware. However, in practice, transient-execution vulnerabilities and CPU bugs are a significant threat to confidentiality; somewhat ironically, the features offered to *increase* security of guest VMs place them at the greatest risk from this new class of attacks. Because the *untrusted* hypervisor controls resource allocation and scheduling of CVMs, any leaks that do exist can be effectively amplified and exploited. For example, a malicious hypervisor may interrupt guest execution at inopportune moments to attempt to leak microarchitectural state, or it may co-schedule attacker and victim vCPUs on different hardware threads of a shared core.

Rather than the status quo of waiting for the development of ad-hoc mitigations to new transient-execution vulnerabilities as they arise, we seek a principled and robust fix. In this regard, the most promising approach is to isolate distinct security domains on distinct cores, a technique we refer to as *core gapping*. Core gapping avoids the vast majority of shared microarchitectural state and (as we'll discuss shortly in §2.2) mitigates all but one of the processor vulnerabilities known to have broken cloud VM security isolation. For a typical VM isolation scenario, core gapping requires scheduling vCPUs of guest VMs on distinct *physical* cores, thus preventing leaks to other VMs through either time-slicing context switches or sibling hardware threads. This is necessary but unfortunately not sufficient for VM isolation; to prevent leaks across the trust boundary with privileged code (i.e., hypervisor and host), we must ensure that they too execute on distinct physical cores [25]. However, the naive implementation of this approach for CVMs runs into an immediate and fundamental limitation: because the hypervisor is untrusted, the guest cannot rely on it to implement core isolation.

Our goal is to address this limitation, enabling core gapping of confidential VMs *without* a trusted hypervisor and

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0391-1/24/04.
https://doi.org/10.1145/3622781.3674190

Table 1. Confidential VM terms in different ISAs.

	Arm CCA	Intel TDX	CoVE [58]
Confidential VM	realm VM	TD VM	TVM
Security monitor	RMM	TDX module	TSM
Privileged mode	realm	SEAM	confidential

without further hardware changes. We leverage the convergence of hardware support for confidential VMs to a model where firmware that is attested and trusted by both host and guest executes on all transitions to/from the execution of confidential vCPUs. We modify this firmware to constrain the physical CPUs on which a confidential vCPU is dispatched and limit the causes and frequency of interrupts. Our modified firmware also delegates all VM exit events to a remote core; we thus prevent the possibility of leaks across the trust boundary, by avoiding the context switch between a confidential VM and untrusted hypervisor.

In this paper, we present our design for core-gapped confidential VMs (§3) on platforms such as Arm CCA, Intel TDX, and RISC-V CoVE [58]. We implement a prototype for Arm CCA (§4) by modifying Arm's trusted firmware, showing that it is practical on a simulator implementing the forthcoming architecture extensions for CVMs, and evaluate its performance on a current Arm platform (without hardware CVM support). Our prototype uses novel techniques to temporarily dedicate host cores to confidential VM execution (§4.2), support fast cross-core RPCs without relying on constant busy-waiting (§4.3), and minimise VM exits by delegating interrupt management to dedicated cores (§4.4). A performance evaluation (§5) shows that thanks to these optimisations, core-gapped CVMs offer performance competitive with shared-core VMs. Finally, we discuss in §6 how the same approach could be applied to competing CVM platforms such as Intel TDX.

2 Background

In this section we cover background and further motivation for our work. We present the common principles underlying emerging ISA extensions for confidential VMs, then discuss the impact and limitations of transient-execution attacks and how recent proposals address such vulnerabilities before concluding with our threat model.

2.1 Confidential VMs

Confidential VMs have gained traction in recent years to the point where all major architectures now include ISA extensions to support them [1, 4, 30]. CVMs offer the ability to "lift and shift" existing workloads to an untrusted cloud environment while providing the capabilities and guarantees typical of more restricted trusted execution environments such as remote attestation, confidentiality, and integrity. In

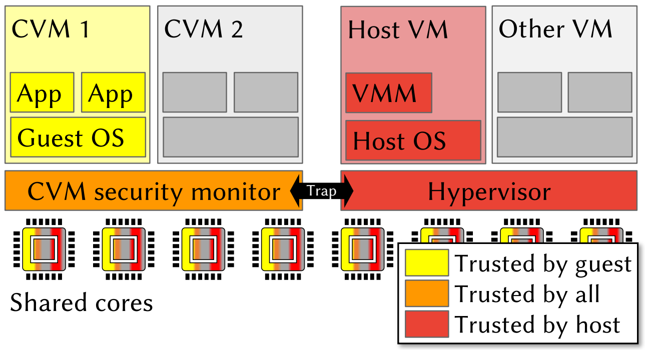

Figure 1. Generic architecture support for confidential VMs [4, 30, 58]. A trusted security monitor protects CVMs, which may run on any core. Cores context switch between trusted and untrusted domains in response to VM entry/exit.

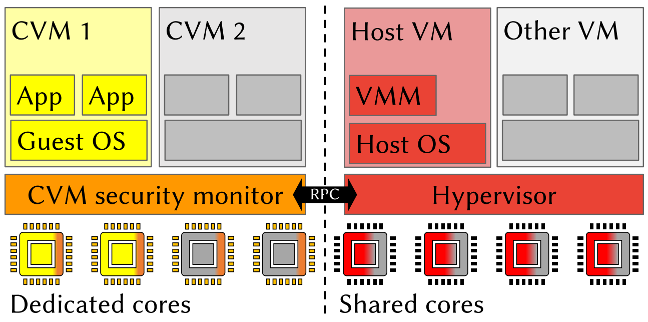

Figure 2. Core-gapped confidential VMs (compare to fig. 1). Physical cores are bound to specific CVMs; VM exits are handled by cross-core RPCs rather than context switching.

particular, CVMs promise to remove the hypervisor (and hence the platform provider) from the guest's TCB.

The design of CVM ISA extensions has mostly converged toward a model in which isolation is enforced through cooperation between hardware and software components, with significant functionality being placed in platform firmware that is attested and trusted by both host and guests. Indeed, Arm CCA/RME [4, 39] (specified as part of Armv9), Intel TDX [30] (commercially available since 2022), and CoVE [58] (a leading RISC-V extension proposal) all follow a very similar design, depicted in fig. 1. Those extensions introduce a new privileged CPU mode which grants access to the physical memory backing confidential VMs, and in which a *CVM security monitor* runs. Physical memory ranges can be configured as confidential (and thus inaccessible to host software) using hardware mechanisms under the control of trusted firmware within the confidential mode. The CVM security monitor enforces isolation between CVMs using second-level page tables, similar to how a traditional hypervisor provides isolation between VMs. Table 1 maps each extension's unique terminology into our unified CVM model.

Besides managing physical memory and CVM second-level page tables, the CVM security monitor also plays the role of the arbiter between CVMs and the host hypervisor. In particular, the security monitor interposes on all transitions between the CVMs and the host to save and restore contexts, validate host-provided configuration, and selectively share the register state of guest VMs. For all situations where it would access guest context, modify guest page translations, or simply initiate guest execution, the hypervisor is required to invoke the security monitor. A call to the security monitor always implies a transition to confidential mode, and a transition back to normal mode requires explicit intervention of the security monitor to save/restore context. This prevents the host from observing guest register state on interrupts or exceptions, although it retains overall control of the CPU.

During transitions to/from the host, the security monitor may attempt to mitigate transient-execution attacks by flushing microarchitectural state; for example, Intel's TDX module flushes branch history on return to the host [34]. However, such flushing carries an inevitable cost, and requires that the relevant microarchitectural structures be exposed to software control; as we will discuss shortly in §2.2 it is often applied only retroactively. Moreover, flushing cannot protect against *architectural* leaks due to bugs in processor implementations [14, 51, 52, 79].

As a notable exception to the architecture depicted in fig. 1, AMD's confidential VM extension, SEV-SNP [1], relies more heavily on the CPU ISA to directly enforce isolation. Rather than a new privileged mode, SEV-SNP places responsibility for protecting guest register context on the implementation of the `VMRUN` instruction used to launch confidential VMs. Furthermore, the page tables for SEV-SNP VMs remain directly accessible to the host, with a shadow reverse-map structure ("RMP") consulted by the processor on each page walk to ensure integrity of a confidential VM's address space. A secure coprocessor manages the lifecycle of confidential VMs, supports attestation, and controls the contents of the RMP table, but is *not* directly involved in the ongoing execution of CVMs. We will discuss the challenges of implementing core-gapped CVMs on AMD SEV-SNP later, in §6.2, but for now we will not consider it further in our design.

2.2 Transient-execution vulnerabilities

While we must defer to prior work for a more thorough treatment of past vulnerabilities [15, 25, 77], we present in fig. 3 a timeline of disclosed vulnerabilities (including transient-execution vulnerabilities and architectural bugs) that have broken processor security isolation on mainstream CPUs (Intel, AMD, and Arm) since 2018. Not all vulnerabilities affected all CPUs and isolation mechanisms; for example, some only affected Intel SGX enclaves [14, 45], while others targeted features specific to Apple's Arm CPUs [17, 55, 60]. What is generally true, however, is that each such vulnerability represents a defeat of the same forms of hardware security mechanism that are now being relied upon to isolate CVMs. For example, the ÆPIC leak [14] (a bug in Intel's local APIC that exposed the processor's internal state) was demonstrated by extracting secrets from an SGX enclave, but the same vulnerability arising today would be just as likely to leak from a TDX VM.

Across all these vulnerabilities, only *one* demonstrated a cross-core leak severe enough to warrant a security advisory from the processor vendor and mitigation by cloud providers. That bug was 2020's CrossTalk [53], which exploited a speculative leak in a microarchitectural staging buffer used in the implementation of instructions such as `CPUID` and `RDRAND` and, crucially, that was shared by multiple cores. By contrast, the recent GhostRace [54] attack relies on multiple cores to steer transient execution, but assumes a setting with a shared kernel between attacker and victim cores; it would be mitigated by core gapping.

Notwithstanding the impact of CrossTalk, that the vast majority of vulnerabilities (30+) were *not* exploitable across core boundaries (even though many were exploitable by a sibling hardware thread) is significant, yet unsurprising. The nature of CPU microarchitecture design dictates that performance-critical structures be kept as close as possible to the execution units of a core and thus be duplicated (rather than shared) across cores; first-level (and often second-level) caches, TLBs, register files, reorder buffers, store buffers, and more are all per-core resources. Fewer shared resources leads to fewer opportunities for leaks and exploits; moreover, for the remaining shared resources located further from the core (and thus, with higher request latencies) such as last-level caches, memory controllers, and on-chip I/O devices, it becomes practical to implement explicit partitioning and resource management to close the remaining side channels [41].

To be clear, cross-core exploitation of transient-execution vulnerabilities remains conceptually possible as long as an interface to trigger the vulnerability and a side channel through which to observe the results of transient execution exist. Such side channels may include contention in the on-chip interconnect [18] or memory controller, or coherence states of memory shared between attacker and victim cores [78]. NetSpectre [62] even demonstrated a remote attack that relies solely on the timing of network traffic, although in its current form the leakage rate is extremely low (fewer than 10 bits per hour in a cloud setting).

To date, the known vulnerabilities have been mitigated (albeit after sometimes lengthy pre-disclosure delays [44]) by a combination of microcode patches, software workarounds, and new hardware. Besides the cost and engineering complexity of developing such mitigations, they can also have a significant performance and functionality impact. Past mitigations have imposed dramatic overheads [13], disabled features such as transactional memory [35], and rendered hyperthreading quasi-unusable [47] on systems that host

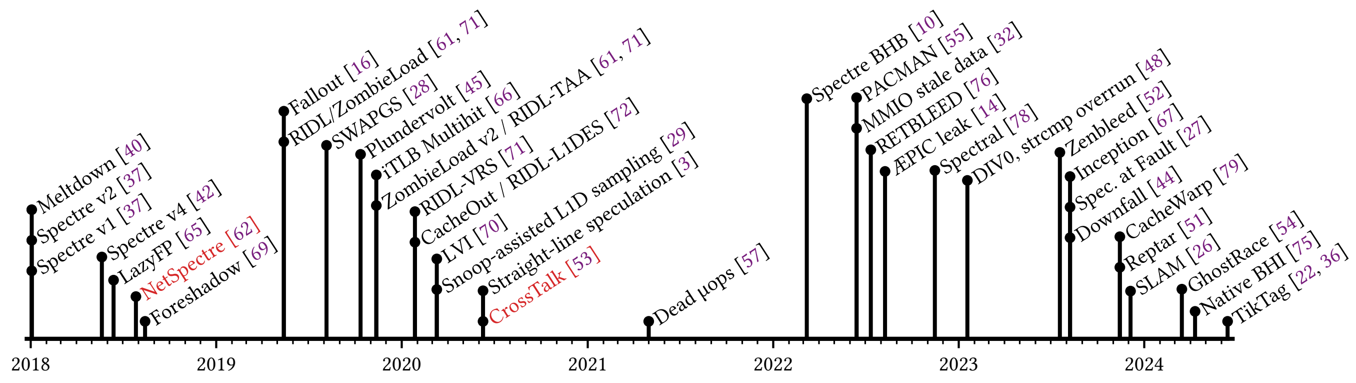

Figure 3. Transient-execution vulnerabilities and CPU bugs that break security isolation. Only NetSpectre [62] and CrossTalk [53] demonstrated cross-core leaks in typical cloud VM settings.

SGX enclaves [33, 69]. We believe that the status quo of ad-hoc mitigations for each discovered attack is neither secure nor sustainable, and seek a more comprehensive solution.

2.3 Core gapping as the basis of isolation

We are not the first to observe that cores offer a natural boundary on which to build stronger security isolation: two recent systems leverage the same observation in different contexts (we will cover other related work later, in §7). The first, Core Slicing [80] also uses isolated cores to avoid transient-execution vulnerabilities and side channels, but replaces guest VMs with statically-partitioned bare-metal tenants and hence requires hardware support. The second, Quarantine [25], runs VMs on distinct physical cores from their hypervisor as we do, but does not consider the unique challenge of confidential VMs. Quarantine is designed to protect the host from malicious guests, but not to protect guests from a malicious host. Applying Quarantine's design to CVMs cannot give guests any better security guarantees than the status quo, as it relies on the hypervisor to implement core isolation. We seek to work within the constraints of existing hardware designs for confidential VMs, while guaranteeing meaningful (attested) isolation to CVM guests.

Like the above systems, our use of core gapping changes the way cores are allocated and CPU time is accounted. Because host CPUs are statically bound to guest vCPUs, it becomes impractical to overcommit host CPUs by time-slicing vCPU execution. Furthermore, virtualization tasks performed by the host (VM exit handling, I/O emulation, etc.) must run on a distinct CPU from the guest vCPUs. Naively, this would seem to require more host resources, or come at an inherently large cost in performance, since fewer vCPUs can run on the same host. In reality, however, the impact is not so significant. In the specific case of infrastructure-as-a-service cloud VMs, Zhou et al. [80] already noted that (a) host CPUs are not generally overcommitted, and (b) I/O is becoming fully virtualised through SR-IOV devices, with minimal time spent in the host handling VM exits. Indeed, while private clouds heavily rely on overcommitment, public clouds are converging to a "slice of hardware" model for IaaS VMs that offers guaranteed non-overcommitted resources for predictable performance. We anticipate that the same will also be true of confidential VMs in the cloud: they will have guaranteed CPU resources and will evolve towards fully hardware-virtualised I/O [2, 31], so we target such a setting in our design, but unlike Core Slicing we do not require it. In particular, we measure the overhead of software-emulated I/O in our evaluation.

More generally, as per-socket core counts steadily increase, the relative cost of dedicating cores to specific purposes decreases. Rather, the dominant performance bottleneck becomes off-chip bandwidth (the "memory wall"); in this regard we can expect remote exit handling to improve performance, because by running the host on a separate core we avoid evicting the guest's working set from per-core caches and the TLB [64]. Nevertheless, in our experiments, we will follow the lead of Hertogh et al. [25] and fairly account for the resource overhead of core gapping by using the same number of *physical* cores in all comparisons.

2.4 Threat model

We adopt the confidential computing threat model, which seeks to guarantee the integrity and confidentiality of guest CVMs from a malicious host, but we strengthen its handling of CPU vulnerabilities and side channels.

We consider distinct TCBs from the perspective of the host (such as the cloud provider), and guests (who run in confidential VMs). Host and guests distrust each other, but both must trust the CPU hardware, trusted firmware including the CVM security monitor, and the systems that remotely attest them; software and hardware vulnerabilities in those components are generally out of scope, with an important exception: hardware side channels and transient-execution vulnerabilities that leak state (breaking confidentiality) or affect execution (breaking integrity) *only within a single core* are in scope and are mitigated by core gapping. For example, side channels in the last-level cache are out of scope (although we recommend the use of hardware cache partitioning mechanisms as an effective mitigation), but attacks exploiting the L1 cache, TLBs, store buffers, reorder buffers, etc. are all defended against. Other lower-level hardware side channels including power and electromagnetic leaks are out of scope.

Guests trust the contents of their confidential VMs and the attestation mechanisms (e.g., rooted in a certificate from the platform vendor) that give them confidence in the integrity of their CVMs and the trusted firmware. Guests can execute arbitrary binary code within their CVMs, seeking to exploit transient execution vulnerabilities in the host or in neighbour VMs. We do not aim to harden isolation boundaries within CVMs; for example, the strength of address-space isolation within a CVM remains as in the status quo. The host controls the hypervisor and thus resource allocation and execution of guest CVMs; denial of service of CVMs is out of scope.

3 Design

The essential properties that core gapping seeks to maintain are: (a) all instructions of a confidential VM's vCPU are executed by the same host core, and (b) from the first to the last instruction of the vCPU, only guest-trusted code (i.e., the security monitor) may run on that core. (From a security perspective, it is not strictly necessary to guarantee that vCPUs cannot migrate *within* the set of trusted cores used by a given CVM, but relaxing this condition adds complexity for no obvious benefit.)

To achieve these properties, as illustrated by fig. 2 we make two changes to the basic CVM software architecture:

1. The security monitor maintains and enforces a binding of CVM vCPUs to host cores. Any attempts by the hypervisor to dispatch a vCPU on the "wrong" core fail, which prevents a malicious guest's vCPU from being dispatched on the same core as a victim VM.
2. When a running vCPU exits the VM because of an exception or external interrupt and requires the service of the host hypervisor (for example, to emulate untrusted I/O devices), rather than saving context and exiting from the privileged mode back to the host hypervisor, the security monitor writes the exit cause information to shared (non-confidential) memory permitting the hypervisor to handle it *on another core.*

The second point is the more invasive of the two changes, yet it is critical: if the security monitor were to permit a switch back to the hypervisor, it would lose control of the core, and would have no way of knowing what had executed and what secrets may have leaked through microarchitectural state. However, never returning to the hypervisor has significant implications – it starves the kernel scheduler on the affected cores, but it is difficult for a general-purpose host such as KVM to ensure that no other threads are scheduled nor any interrupts delivered to such cores. Our insight here is that we can take these cores completely "offline" from the point of view of the host scheduler, by using the existing processor hotplug support, and then dedicate them to the security monitor and its guest CVM for the life of that guest. Then, instead of executing the guest VM via a local call to the security monitor, the host is changed to turn all such calls including running a guest vCPU into cross-core RPCs that the security monitor handles on a dedicated core. In handling some VM exits, the hypervisor may require further action by the security monitor, for example to manipulate page tables, and these are also implemented by changing the existing APIs to cross-core RPCs.

Replacing local context switches between the hypervisor and security monitor with cross-core RPCs may add latency, but as we will show it also reduces overheads, both direct (saving/restoring register state and flushing microarchitectural structures to mitigate transient execution vulnerabilities) and indirect (cache and TLB pollution, and cold microarchitectural state due to the aforementioned flushing). Such performance tradeoffs between same-core privilege transitions and shared-memory RPCs have been explored in the literature [21, 50, 64, 74] and do not fundamentally differ here; our observation is that the remote case can not only improve performance, but also *strengthen* security isolation.

The final aspect of our design is a user-mode *core planner* component that performs admission control on CVMs, assigns cores to run them, and orchestrates the process of dedicating those cores to confidential mode and later returning them to normal execution. This can be considered on one hand as a complement to the cloud resource allocators that schedule VMs to nodes [24, 73] (logically extending them to consider resource allocation inside a node), and on the other as an extension of existing VM schedulers that seek to maintain affinity of vCPUs for NUMA locality [43, 49] (making explicit longer-term placement decisions). The primary practical change is that whereas previously vCPU threads would be "pinned" to specific host cores only as a performance optimisation, a static allocation of vCPUs to host cores is now enforced from the start of a CVM. To avoid long-term fragmentation of available cores (and thus poor locality), we envisage permitting limited changes of the vCPU-to-core binding at coarse (e.g. 10s of seconds) time scales; this could be implemented in the CVM security monitor, but we defer further investigation of it to future work.

4 Implementation

We describe a prototype implementation of core-gapped confidential VMs for Arm CCA and Linux/KVM. We begin with a brief overview of the relevant architecture mechanisms, before describing the basic implementation of core gapping, followed by optimisations to reduce its overhead.

4.1 Arm CCA

Arm's confidential compute architecture (CCA) follows the design principles outlined in §2.1. A lightweight *realm management monitor* (RMM) [5] executes in a privileged *realm world* [4] where it manages second-level page tables and CPU context to isolate confidential VMs from each other. The host runs in normal world and can not access realm world memory. To interact with the RMM the host must use a *secure monitor call* [7] to trusted firmware, which then switches the CPU into realm world and forwards the call to the RMM which handles the call and ultimately returns to the host, again via trusted firmware.

Arm specifies a standard RMM interface [5] to enable interoperability between host and RMM implementations. The host remains responsible for resource management and virtualization, but the RMM is responsible for validation and enforcement. This interface exposes functions for creating and managing confidential VMs, delegating memory to realm world, requesting updates to guest page tables, and running vCPUs (among others). In this model a guest's VM exit is perceived by the host as a return from an RMM call to run the vCPU, with the return value including selected guest registers required for exit handling.

We chose to develop our prototype on Arm because, even though hardware that implements the necessary realm-management extension (RME) ISA has not yet shipped, Arm provides both a reference implementation of the trusted software for CCA [68] and an architectural simulator [6] on which to run it. Furthermore, as soon as RME-capable Armv9 systems ship, we expect they will be able to run our prototype, since it uses only standard architectural features. By contrast, despite the earlier availability of Intel TDX hardware, it is unsuitable for such research – a hardware-enforced security policy limits the user to running only TDX module binaries signed by Intel (see discussion in §6.1).

Our prototype is based on the reference RMM (version 0.3.0) and Linux/KVM (version 6.2) with Arm's modifications to support CCA. In total, we modified a further 1030 lines of code in Linux and 860 lines (2.7% increase) in the RMM to implement core gapping. We did not change the APIs that the RMM exposes to either host or guests, so no changes are required in the guest VM. Besides running our prototype on the ISA simulator (which is not performance accurate), we also modified it to run on current (Armv8) hardware, to permit a performance evaluation on real servers. In this mode, the RMM runs in normal world (not the new realm mode) on dedicated cores using memory reserved for it at system boot time, but without any security isolation from the host.

4.2 Dedicating cores

The crux of core gapping is to ensure that no two security contexts (no two CVMs, nor a CVM and any host code) are co-located on the same physical core. This requires us to (1) inform the host that some cores can no longer be used for its own operations; (2) prevent control of those cores being returned to the host before the CVM using them terminates; and (3) ensure that the security monitor will never permit two CVMs to be co-scheduled on the same core.

We use the Linux hotplug interface to gracefully restrict the host OS to a dynamic subset of physical cores. When taking a core offline for hotplug, the Linux kernel migrates all tasks away from the core, re-configures interrupts to target other cores in the system, and marks it as unusable for future operations.[1] The only modification required to the existing hotplug shutdown procedure is to disable the frequency scaling request initiated by the scaling governor during clean-up, so the "offline" cores continue running at normal power and frequency settings. We then introduced a final call to the security monitor to transition the "offline" cores into realm mode rather than halting.

Once a core is dedicated to realm world, no transitions back to normal world should occur on that core until the CVM it is running terminates. To ensure this, we rely on the architecture that delivers all exceptions and interrupts occurring in realm world to the RMM. The intended use is that the RMM saves and wipes guest register state before (in typical operation) switching back to the host hypervisor to handle the exception. We instead modify the RMM so that it never returns execution to the host, effectively taking control of the dedicated cores.

Additionally, our modified RMM enforces a stricter core assignment policy to prevent the host from time-slicing two mutually distrustful CVMs on a single core. Similarly to baseline CCA operations, the host is responsible for resource management, including vCPU placement. Our prototype exposes an additional `ioctl` for the VMM to act as a user-space core-planner; we also provide a default placement policy to support unmodified VMMs. We block the host from co-scheduling vCPUs by marking a physical core as dedicated from the first entry of a vCPU on that core until the corresponding vCPU is destroyed. Any later attempt to dispatch the same vCPU on another core will also return an error to the host. Only when the CVM is terminated (either by the host, or because it exited gracefully) and its vCPUs are destroyed can the host schedule another CVM or reclaim control of the dedicated cores, whereupon they are brought back "online" through another hotplug event.

[1]Since none of our target Arm platforms support hardware threads, we use the term "core" here for simplicity. However, on a threaded processor, we would dedicate *all* hardware threads of the same core to confidential VMs, and *all* threads of a single core would be bound to the same CVM.

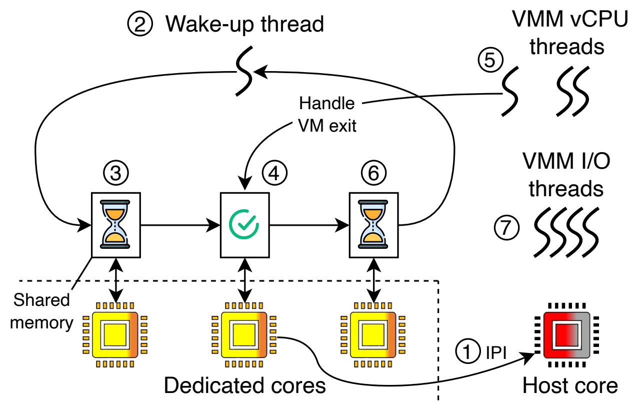

Figure 4. Asynchronous call mechanism. A wake-up thread (new to our design) polls RPC channels to wake runnable vCPU threads in response to an IPI.

4.3 Remote RMM calls

Since host and guest cannot share a physical core, core gapping replaces CPU mode switches with remote procedure calls on shared memory. Practically, the synchronous local RMM call that KVM uses to run a guest vCPU is replaced by an RPC to the RMM on a dedicated core. This call returns when the guest vCPU exits, whereupon KVM (typically) performs some emulation before issuing the next run call. Thus, although KVM is in fact the RPC *client*, an alternative (logical) view of the situation is of the RMM *calling* KVM to handle VM exits. To maximise the utilisation of guest-dedicated cores and thus the performance of guest CVMs, we must optimise for the latter view, i.e. the time between a vCPU exit event that causes a run call to return, and issuing the next run call that resumes VM execution. In the rest of the paper we refer to this delay as the *run-to-run latency*. Meanwhile, to maximise the scalability of the system where a few host cores service many guest CVM cores, we must also minimise the RPC CPU overhead on the host. Our prototype gracefully integrates with KVM's thread-per-vCPU design by selectively leveraging asynchronous RPCs to concurrently manage multiple vCPUs on a single host core, and supports all the same KVM functionality as unmodified Arm CCA VMs.

We make a distinction between vCPU run calls and all other RMM invocations. Because they involve guest execution, run calls are unbounded operations (often taking tens of milliseconds or longer) whereas all other RMM calls are only for quick manipulations of CVM state, such as installing page translations. For the latter short-lived calls, we use a simple synchronous call mechanism with busy-waiting polling on both host and RMM cores. Specifically, the host thread writes the call arguments into shared memory before polling for a response. Meanwhile, on the RMM side, all cores dedicated to RMM execution either run their corresponding vCPU or poll

Table 2. Comparison of null RMM call latencies.

Call	Latency
Core-gapped asynchronous (vCPU run calls)	2757.6 ns
Core-gapped synchronous (e.g., page table update)	257.7 ns
Same-core synchronous	>12.8 µs

shared memory for incoming host calls (to minimise their latency). There is a one-to-one mapping between host vCPU threads (those controlled by the userspace VMM process) and RMM dedicated cores. Thus, while a VM exit is serviced by a vCPU thread (which is ideally rare and quick) there is no useful work to do on the corresponding dedicated core.

For CVM run calls, the host vCPU thread does not busy-wait for a response, but instead blocks immediately after writing the call arguments into shared memory. Fig. 4 shows the sequence of events that will unblock it. When a vCPU exits (and thus when the host's run call must return), the RMM writes the exit reason and selected guest registers to a shared page and sends an inter-processor interrupt (IPI) to the host to signal an asynchronous return (①). Arm supports 16 possible IPI numbers, 7 of which are already reserved by Linux, so we cannot convey information in the IPI itself and allocate only one further IPI as a notification of CVM exits. The handler for that IPI activates a wake-up thread (②) which polls shared memory until a stopped vCPU is found (③–④). The wake-up thread then unblocks the corresponding KVM vCPU threads (⑤) before continuing to poll for other vCPU exits (⑥) and finally suspending itself until the next IPI. To minimise run-to-run latency, the vCPU threads run at high priority (Linux's FIFO scheduler class), so they typically run until completion whereupon they resume the suspended vCPU and again block until its next exit. Unlike Quarantine [25], we rely on IPIs rather than continuous polling to handle VM exits, because we found that although this increased the run-to-run latency on an uncontended system, polling competed for CPU time with other host threads such as those in the VMM process responsible for I/O emulation (⑦) resulting in worse overall performance on emulation-intensive workloads.

Table 2 shows the round-trip latencies of synchronous and asynchronous remote RMM calls on a current Arm server (§5.1). This supports our decision to use asynchronous invocations only for the long-running calls, with busy-waiting for the rest, improving both throughput and run-to-run latency. In both cases we change only the RPC transport, but not the API exposed by the RMM to the host.

Without suitable hardware we cannot measure the cost of a same-core RMM call, but as a point of comparison we report the latency of a null call to trusted firmware (Arm EL3) on the same system. This is only part of the end-to-end RMM call path yet takes more than four times longer

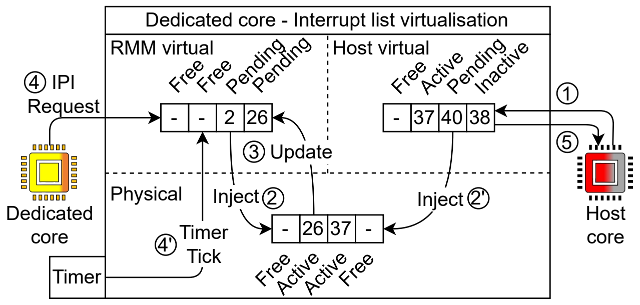

Figure 5. Virtualization of virtual interrupt list registers. The host observes and manipulates a virtual list containing a subset of the guest interrupts, allowing the RMM to inject interrupts without perturbing host operations.

than a remote call due primarily to mitigations for transient-execution attacks. Although we hope that future hardware will not require such mitigations, experience (§2.2) suggests that they may accrue over time.

4.4 Interrupt management

The RMM itself executes with interrupt disabled, but intercepts physical interrupts during vCPU execution that cause vCPU exits in accordance with the RMM specification. In addition, the baseline RMM configures the CPU registers related to the local interrupt controller to induce traps, delegating their emulation to the host. We found that interrupt management was by far the most significant source of exits to the host for compute-bound workloads, with exits caused both by physical interrupts and the virtualization of the guest interrupt controller. For example, each tick of the virtual timer induces two exits, and in our experiments over 90% of exits when running CoreMark-PRO [19] are due to local timer interrupts. Such a high exit frequency might be tolerable on a shared core where interrupts are handled locally, but becomes a significant source of overhead with core gapping due to congestion on the remote host core. Moreover, because we dedicate cores to vCPUs, there is less need for a fully virtual interrupt controller.

In our prototype, we address this problem by delegating some interrupt management to the RMM. There is an intrinsic tension between maintaining a small TCB and offloading more emulation to the RMM. However, we note that some interrupt sources can be emulated very simply while significantly reducing the total interrupt count. In particular, we emulate both the virtual timer and inter-processor interrupts directly in the RMM, accounting for 150 and 70 new lines of code respectively. Note that this does not require MMIO emulation (which remains delegated to the host), but rather changes handling of CPU register writes that already trap to the RMM. By doing so we also further isolate the CVM guest from a malicious host; arguably, providing a trusted source of time to a confidential guest is of more importance than some additional emulation logic in the TCB.

We modified the RMM's interrupt handling logic to make the offloading transparent to KVM. On Arm CCA, virtual interrupts are controlled by a list of active and pending virtual interrupts (`ich_lr<n>_el2` registers). The RMM interface specification expects the host to manage virtual interrupts, therefore the run call takes an interrupt list as argument, and returns a potentially updated list of interrupts on exit. We make interrupt virtualization offloading transparent to KVM by exposing a filtered list of virtual interrupts instead, hiding some interrupts from KVM while managing the true list in the RMM. Fig. 5 illustrates interrupt management on our prototype. As a parameter to the vCPU run call, the host sends a list of virtual interrupts to install on the interrupt controller (①). The RMM then injects new pending interrupts if it can find free slots in the physical list (② & ②'), and updates the state of virtual slots to reflect interrupt life-cycles (③). During guest operation, the RMM can decide to inject interrupts in response to timer ticks or IPI requests *without host interaction* (④ & ④'). The KVM host can still request exits, for instance to inject an interrupt from an emulated I/O device, by sending an IPI as it does for shared-core VMs. On exit to the host, the RMM synchronises the slots one last time and forwards the virtual list to KVM (⑤), which can resume operation transparently.

Table 3. Virtual interprocessor interrupt latency.

	IPI latency
Core-gapped CVM, without delegation	43.9 µs
Core-gapped CVM, with delegation	2.22 µs
Shared-core VM	3.85 µs

Table 3 reports the time to deliver and acknowledge (in shared memory) a virtual IPI; interrupt delegation is particularly effective at reducing this cost, because it avoids VM exits on both sender and receiver cores. In particular, interrupt delegation completely skips the host's scheduler, which might delay virtual interrupt injections for share-core VMs.

5 Evaluation

In this section we describe a performance evaluation of our Arm CCA-based prototype of core-gapped confidential VMs. We demonstrate that core-gapped CVMs offer competitive performance across a variety of workloads, and sometimes even slightly outperform a shared-core baseline.

5.1 Experimental setup

Since Armv9 hardware supporting CCA is not yet available and the architectural simulator is not cycle accurate, we evaluate our prototype on a cloud server using an AmpereOne SoC with Armv8.6 cores at 3 GHz. To run our prototype on this platform without interference with the host OS, we statically reserve system memory for it at boot time (much as the CCA firmware would do), and load the RMM image into it after boot. When dedicating a core to run a CVM, we do not attempt to switch to realm mode but rather jump directly to the previously-loaded RMM running in normal world (which thus cannot enforce isolation from the host). This version of our RMM prototype has been further modified to avoid interaction with trusted firmware (EL3) and to handle all RMM calls from the host OS directly, which suffices to run CVMs without RME hardware.

Overall, we expect our evaluation platform to perform comparably to a true CCA platform, with two notable differences. First, CCA hardware will encrypt the memory of confidential VMs. On other platforms (e.g., Intel TDX), memory encryption adds a 2–3% overhead to memory-intensive workloads [23], a cost that will be the same with or without core gapping. Second, CCA hardware performs additional physical memory ownership (granule protection) checks in the TLB miss handler. Although we cannot predict the cost of such checks, because core-gapped CVMs avoid interference from the host OS, we expect them to achieve better TLB locality than shared-core CVMs on the same hardware.

More challenging than the core-gapped CVM is the choice of baseline system to which its performance may be compared. Without suitable hardware, we cannot run the RMM in shared-core mode, as it would conflict with the host OS's control of normal world. Therefore, our experiments use a traditional (non-confidential) shared-core VM as the baseline. This configuration avoids not only the cost of hardware memory encryption and EL3 calls (as in our own CVM configuration), but it also avoids the overhead of world switches, context save/restore sequences for transitions to/from realm world, and RMM bookkeeping. Thus, our experiments will unfortunately exaggerate any performance overheads of core gapping, and we can expect core-gapped CVMs to offer relatively *better* performance compared to shared-core *confidential* VMs when such a comparison becomes possible.

We use Linux 6.2 for the host kernel with the *performance* scaling governor and kvmtool as the VMM, both modified to support Arm CCA [8, 9]. To fairly account for the resource requirements of core gapping, we use the same number of physical cores in all experiments. In particular, for a workload with N cores, we run an N core VM in the baseline, restricting the VMM threads to run on a fixed set of N host cores. In the core-gapped CVM configuration, we instead run an $N-1$ core VM (on dedicated cores) for the same workload, while pinning all VMM threads on the host to a single additional core. Therefore, the core-gapped CVMs in our experiments always runs with one fewer vCPU. Unless specified otherwise, we use 16 cores for all of our experiments, thus 16 vCPUs for shared-core VMs and 15 vCPUs for core-gapped CVMs.

Table 4. Interrupt delegation effect on CoreMark-PRO

	Without delegation	With delegation
Interrupt-related exits	33954 ± 161	390 ± 3
Total exits	37712 ± 504	1324 ± 60

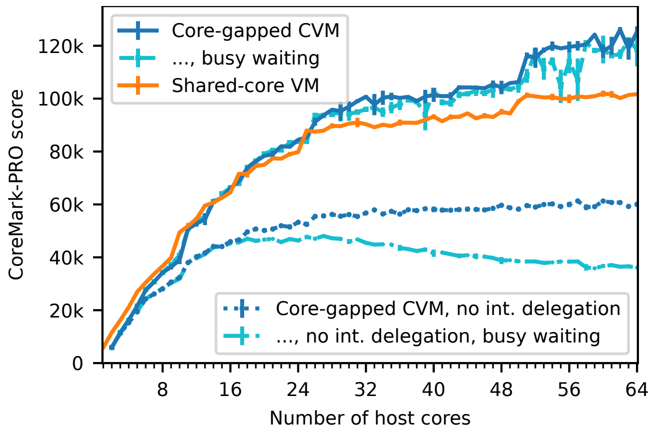

Figure 6. CoreMark-PRO [19] scaling for shared-core (baseline) VMs and core-gapped CVMs. Core-gapped CVMs use $N-1$ dedicated cores and 1 host core. Higher is better.

5.2 Scaling a CPU-intensive workload

We first evaluate the performance implications of core gapping on a CPU-intensive workload, and quantify the impact of our optimisations. We expect to see performance impacts from core gapping due to costlier exits to the host, contention on the host core for both I/O and vCPU threads, and RMM overhead. On the other hand, core-gapped CVMs may benefit from greater locality, potentially boosting performance.

We start by evaluating the impact of scaling up the number of dedicated cores running CoreMark-PRO [19] (a CPU-intensive benchmark), while keeping a single host core running all VMM threads. All guest VMs are assigned 16 GiB of RAM. Fig. 6 shows that a core-gapped VM can scale to a high number of dedicated cores (here up to 63) without any contention on the host core. Indeed, for this workload run-to-run latency does *not* noticeably increase with the guest core count and remains stable at 26.18 ± 0.96 µs. The host overhead can be kept low despite high core counts thanks to the combined effect of interrupt delegation and asynchronous calls: by handling most interrupts locally we reduce the total exit count by 28× (table 4), and by using IPI notifications rather than Quarantine-style yield-polling [25] we avoid saturating the host core with always-runnable VMM threads (cyan-coloured "busy waiting" lines in the figure).

Fig. 7 shows that our prototype can also scale to multiple CVMs. Rather than a single VM with an increasing number of vCPUs, we run an increasing number of 4-core VMs and plot the aggregate throughput. Because each instance of CoreMark has only 4 vCPUs, the aggregate scales linearly,

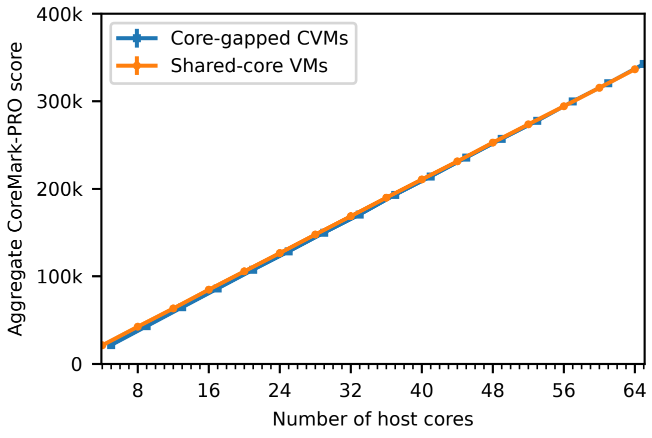

Figure 7. Scaling to multiple VMs. We plot the aggregate CoreMark-PRO score for an increasing count of 4-core VMs/CVMs.

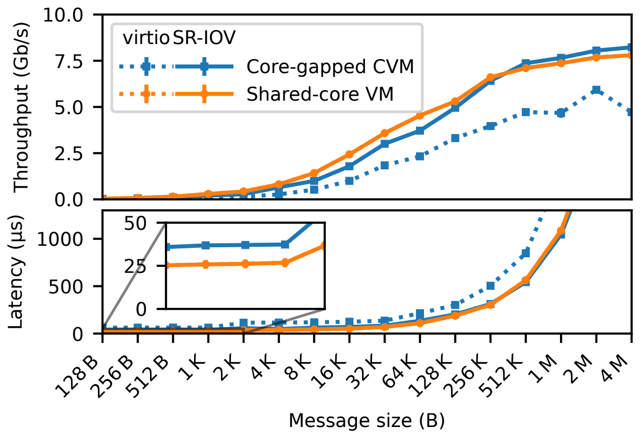

Figure 8. NetPIPE [63] TCP results, 1500-byte IPv6 packets.

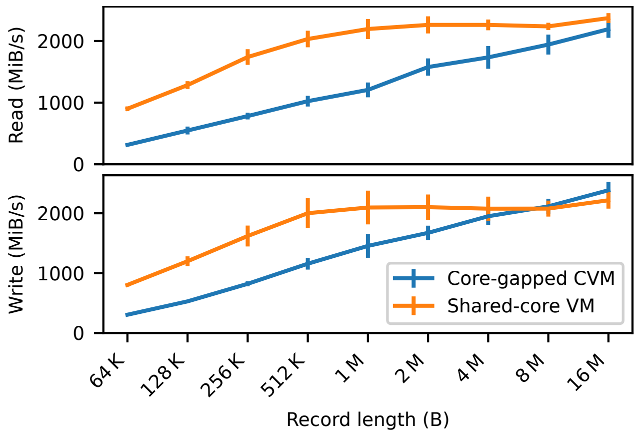

Figure 9. IOzone [46] sync read/write throughput to a virtio block device, using O_DIRECT to bypass the guest's cache.

Table 5. Redis benchmark. 50 clients, 512-byte objects.

		Throughput (krps)	Latency (ms) Mean	p95	p99
SET	shared core	51.7	0.52	0.60	1.20
	core gapped	56.2	0.63	0.97	1.44
GET	shared core	48.8	0.54	0.64	1.20
	core gapped	55.3	0.57	0.78	1.24
LRANGE 100	shared core	11.6	1.51	2.03	2.38
	core gapped	14.5	1.24	1.56	1.82

and we see that running up to 16 VMMs pinned on a single host core does not harm throughput.

5.3 I/O performance

We next investigate I/O performance, where we expect to see the impact (if any) of higher VM exit latencies.

To measure the performance of network virtualization, we configure our guest VMs with two different network interfaces: a virtio interface emulated by kvmtool running on the host, and a virtual function network interface of an Intel E2000 200 GbE IPU, that we assign to the guest VMs using SR-IOV pass-through. Another equivalent but unmodified system serves as a network client. Fig. 8 shows that the virtio interface (dashed lines) has substantially (up to 2×) higher latency and 30–70% lower throughput in a core-gapped CVM due to the exit- and emulation-intensive nature of virtual I/O. However, with an SR-IOV network interface, data flows directly between the guest VM and the NIC hardware, with the host serving only to deliver interrupts. The SR-IOV configuration, which is becoming the norm for cloud VMs, achieves latency within 10–20 µs of the baseline with up to 5% higher throughput for larger message sizes as the workload becomes more compute intensive. The additional interrupt latency can be attributed to a limitation of our prototype, which does not support direct interrupt delivery and must therefore rely on the host to inject interrupts to the VM. Direct interrupt delivery could be supported through further changes to KVM and RMM.

Finally, fig. 9 shows that core-gapping suffers from the increased latency of virtio block I/O, achieving similar throughput only on large (>10 MiB) virtual disk I/Os.

5.4 Application benchmarks

As a sample network-intensive server application, we use `redis-benchmark` [56] to exercise Redis v7.0.14 (also using SR-IOV); table 5 reports the results. Core gapping achieves around 10% higher throughput in line with microbenchmarks considering that redis saturates the guest CPU, but also higher latency (up to 20% at p99) due to contention on the host. The longer-running and memory-intensive LRANGE query is an exception, delivering lower latency due to reduced contention.

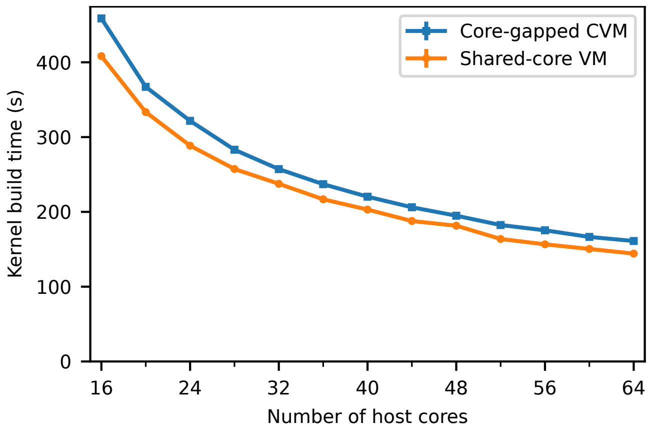

Figure 10. Linux kernel build; virtio disk & 48 GiB RAM.

Lastly, fig. 10 shows that kernel builds on a core-gapped CVM scale similarly to a shared-core VM. This benchmark relies on virtio disk, which puts core-gapping at a disadvantage due to contention for I/O emulation on the host core as illustrated by fig. 9, yet core-gapped CVMs achieve comparable performance despite using one fewer vCPU.

5.5 Summary

We showed that our performance optimisations are effective at reducing VM exit rates and that core-gapped CVMs offer competitive performance with shared-core VMs, especially for CPU-intensive workloads that benefit from less host interference. Conversely, core-gapping suffers the highest overhead for emulated I/O due to its exit-intensive nature. We consider these results promising, for two reasons. First, cloud servers are moving towards full I/O offload with SR-IOV for both network interfaces and local storage. Second, confidential VMs on shared cores will have higher VM exit latencies than the non-confidential baseline used in our experiments, because of the added cost of world switches to/from the confidential execution mode. Together with hardware support for direct I/O [2, 31] it is therefore plausible that core-gapped CVMs will outperform shared-core CVMs in the near future.

6 Discussion

Arm is not the only CPU architecture at risk of transient-execution vulnerabilities. Indeed, most of our motivating vulnerabilities (§2.2) were found in Intel and AMD systems (a fact we attribute primarily to greater attention from security researchers due to their larger market share to date in server and desktop systems). We therefore discuss how our design might apply to those CVM platforms.

6.1 Intel TDX

As we noted in §2.1, Intel TDX is architecturally very similar to Arm CCA's realm-management extension. Both extensions introduce a new privilege mode running trusted firmware that manages the register context and page tables of confidential VMs, and mediates their interactions with the host hypervisor. In TDX, that trusted firmware is known as the *TDX module*, and it provides a host-facing API that is intended to be invoked through a call-gate mechanism and bears a not-unsurprising similarity to the RMM's realm management interface. We therefore expect that our design would apply directly to TDX, with a modified TDX module that avoids relinquishing control of cores dedicated to CVM execution and supports a shared-memory RPC interface with the host rather than same-core calls.

Perhaps the biggest architectural difference between CCA and TDX is that the latter uses separate secure and insecure page tables for confidential VMs, allowing the host to manipulate untrusted portions of guest address space without calling the firmware. By contrast, the RMM is invoked for all page table modifications; thus we might expect a core-gapped version of TDX to have moderately better relative performance, due to fewer cross-core RPCs.

However, TDX also comes with a major drawback from the point of view of a systems researcher: due to restrictions implemented in the trusted code that configures and loads the TDX module that are ultimately enforced by the processor, *only* TDX module binaries signed by Intel can be loaded. Moreover, even though Intel has released source code for the TDX module, there is no documented mode (not even a debug mode) that would permit loading a non-Intel-signed TDX module. While such restrictions may appear necessary for the security of the overall TDX platform, this is not the case. Just as CCA includes the measurement of the RMM in the chain of trust for remote attestation, TDX also establishes the user's trust in the specific implementation of the TDX module by including its measurement as part of the overall attestation signature. Thus, there is no specific reason that only Intel-authored TDX modules need be attestable, let alone that non-Intel-authored TDX modules be prevented from loading on an end user's platform.

Of course, Intel could add support for core gapping to the TDX module themselves, but we hope they will go further and consider relaxing such restrictions in time for the wider availability of TDX hardware (at present it is available only to some cloud providers). Ultimately, enabling research only strengthens the security of platforms such as TDX.

6.2 AMD SEV

AMD SEV-SNP [1] differs notably from the design of other confidential VM platforms, relying solely on the CPU's instruction set (specifically, the VMRUN instruction) and its exception handling mechanisms to mediate control transfers between confidential VMs and the host. In particular, no software execution on the CPU is more privileged than the hypervisor, which is not trusted by CVMs. Adapting our design to provide core-gapped CVMs on AMD platforms is thus substantially more challenging. Although we could easily anticipate VMRUN restricting the physical cores on which confidential vCPUs could be dispatched, reporting and handling VM exits wholly on a *remote* core without relinquishing control to untrusted host code seems challenging. As a reminder, in our threat model of arbitrary unknown same-core vulnerabilities, the exception handlers that run on the same cores as the CVMs must be trusted by the guest. Thus, even though the code needed to turn a VM exit into a cross-core RPC, such as the shim hypervisor in Quarantine [25], may be simple, because it is not part of the confidential VM that is measured, attested and protected by SEV the end user cannot rely on its existence.

One approach to core-gapped SEV VMs might leverage its unique support for multiple VM privilege levels (VMPLs) *inside* a single CVM, and modify the architecture to reflect guest exceptions to a higher privilege level within the CVM. In combination with an extension to the VMRUN instruction to enforce a static binding of vCPUs to host cores, this could enable core gapping. However, it's not clear that such a design could avoid fatal nested exceptions, nor that it would retain sufficient control to the host; for example, the host must be able to terminate a CVM and regain control of its cores.

Overall, this illustrates the limitations of embedding complex trusted execution functionality in a processor ISA [12], even if it may be implemented by a flexible microcode.

7 Related Work

The most closely related work is Quarantine [25], which implements core gapping for normal (non-confidential) VMs on x86, with a trusted hypervisor. Although the same design does not directly apply to confidential VMs (see §2.3), we would expect to see broadly similar performance trade-offs. In this regard, it's interesting to note that Quarantine's performance evaluation showed a single host core becoming saturated after around 10 guest cores, whereas our prototype scales beyond 64 guest cores. We attribute this difference to our asynchronous RPCs for long-running calls (§4.3) and delegated interrupt management (§4.4). Fig. 6 in §5.2 supports this: when using busy waiting in place of asynchronous RPCs and disabling interrupt delegation, core gapping suffers from the same scalability bottlenecks as Quarantine.

Core Slicing [80], which seeks to address similar security threats as core gapping, proposes hardware modifications to enforce static allocation of cores and physical DRAM ranges to bare-metal cloud tenants. In contrast, core-gapped VMs work on existing hardware architectures, requiring changes only to software (and trusted firmware) to bind vCPUs to physical cores. Thus, like other VMs but unlike core slicing, core-gapped VMs can support dynamic memory allocation and deallocation, virtual I/O, host-initiated suspend/resume, and live migration (once the RMM supports it). Finally, core-gapping retains the familiar VM abstraction and control-path APIs for both the guests and the host, and thus is compatible with unmodified VMs and existing user-space VMMs.

A number of systems have used cross-core RPCs in place of local exceptions to improve locality and performance. FlexSC [64] showed the benefits of handling system calls on a remote kernel core, especially in conjunction with a user-mode thread library to exploit application-level concurrency. Our handling of VM exits on a remote core has similar locality benefits, but the VM interface precludes concurrent guest execution while handling an exit. SplitX [38] proposed hardware extensions to reduce the cost of remotely-handled VM exits; core gapping may benefit from such extensions, but we suspect they are no longer required since improvements in hardware virtualization support have reduced the rate of VM exits. A similar cross-core technique is also used to mitigate the high direct cost of SGX enclave crossings [21, 50, 74].

A number of efforts have been made to verify safety and correctness properties of the reference RMM [20, 39], but transient-execution attacks and processor vulnerabilities are out of scope; such efforts are therefore complementary to our work.

Our use of the Linux CPU hotplug facility to dedicate cores was inspired by the Linux kernel support for AWS Nitro Enclaves [11]. Nitro enclaves allow a VM to donate a subset of its cores to an enclave child at runtime. The donation is mediated by the Nitro hypervisor, which triggers a virtual hotplug event and removes the corresponding vCPUs from the parent VM before starting a new Nitro enclave VM with the same number of vCPUs, effectively donating them from the parent to the child. In this configuration the Nitro hypervisor must be trusted by both parent VM and child enclave. Unlike Nitro enclaves and VMs, the core-gapping host kernel and the RMM run on bare metal. The host kernel directly triggers the hotplug procedure and transfers control to the RMM. Our approach does not depend on a trusted intermediary for donating vCPUs, but requires a modification to the hotplug path to skip the final step that would otherwise physically shut down the cores or put them in a low-power mode.

8 Conclusion

A ceaseless tide of CPU bugs threatens the security of confidential VMs. Core gapping offers a practical mitigation to such vulnerabilities whose performance can even exceed that of traditional shared-core VMs, especially as cloud providers move to full hardware offload for I/O virtualization. We evaluated an Arm prototype, and argued for the adoption of core-gapped CVMs on platforms such as Intel TDX.

Acknowledgements

We thank Edouard Bugnion, David Culler, Bernhard Grill, Stanko Novakovic, the anonymous reviewers and our shepherd Shih-Wei Li for their feedback.

References

[1] *AMD SEV-SNP: Strengthening VM isolation with integrity protection and more.* AMD, January 2020. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[2] *AMD SEV-TIO: Trusted I/O for Secure Encrypted Virtualization.* AMD, March 2023. https://www.amd.com/content/dam/amd/en/documents/developer/sev-tio-whitepaper.pdf.

[3] *Straight-line Speculation.* Arm Limited, June 2020. ver. 1.0 https://developer.arm.com/documentation/102825/latest/.

[4] *Arm Realm Management Extension (RME) System Architecture.* Arm Limited, November 2021. Document DEN0129 ver. A.b https://developer.arm.com/documentation/den0129/ab.

[5] *Realm Management Monitor specification.* Arm Limited, December 2022. Document DEN0137 ver. 1.0-bet2 https://developer.arm.com/documentation/den0137/1-0eac5/.

[6] *Armv-A Base RevC AEM fixed virtual platform (FVP).* Arm Limited, June 2023. https://www.arm.com/products/development-tools/simulation/fixed-virtual-platforms v11.22.

[7] *SMC Calling Convention.* Arm Limited, August 2023. Document DEN0028F ver. 1.5 https://developer.arm.com/documentation/den0028/latest/.

[8] *CCA-capable kvmtool VMM.* Arm Limited, May 2023. https://gitlab.arm.com/linux-arm/kvmtool-cca tag: cca-rfc-v1.

[9] *CCA-capable Linux kernel.* Arm Limited, May 2023. https://gitlab.arm.com/linux-arm/linux-cca tag: cca-full-rfc-v1.

[10] Enrico Barberis, Pietro Frigo, Marius Muench, Herbert Bos, and Cristiano Giuffrida. Branch history injection: On the effectiveness of hardware mitigations against cross-privilege Spectre-v2 attacks. In *Proceedings of the 31st USENIX Security Symposium*, pages 971–988, August 2022. ISBN 978-1-939133-31-1. https://www.usenix.org/conference/usenixsecurity22/presentation/barberis.

[11] Jeff Barr. AWS Nitro Enclaves – isolated EC2 environments to process confidential data, October 2020. https://aws.amazon.com/blogs/aws/aws-nitro-enclaves-isolated-ec2-environments-to-process-confidential-data/.

[12] Andrew Baumann. Hardware is the new software. In *Proceedings of the 16th ACM Workshop on Hot Topics in Operating Systems*, pages 132–137, 2017. doi: 10.1145/3102980.3103002.

[13] Jonathan Behrens, Adam Belay, and M. Frans Kaashoek. Performance evolution of mitigating transient execution attacks. In *Proceedings of the 17th ACM European Conference on Computer Systems*, pages 251–265, 2022. doi: 10.1145/3492321.3519559.

[14] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. ÆPIC leak: Architecturally leaking uninitialized data from the microarchitecture. In *Proceedings of the 31st USENIX Security Symposium*, pages 3917–3934, Boston, MA, August 2022. ISBN 978-1-939133-31-1. https://www.usenix.org/conference/usenixsecurity22/presentation/borrello.

[15] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. A systematic evaluation of transient execution attacks and defenses. In *Proceedings of the 28th USENIX Security Symposium*, pages 249–266, August 2019. ISBN 978-1-939133-06-9. https://www.usenix.org/conference/usenixsecurity19/presentation/canella.

[16] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. Fallout: Leaking data on Meltdown-resistant CPUs. In *Proceedings of the 26th ACM Conference on Computer and Communications Security*, pages 769–784, 2019. doi: 10.1145/3319535.3363219.

[17] Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher W. Fletcher, David Kohlbrenner, Riccardo Paccagnella, and Daniel Genkin. GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers. In *Proceedings of the 33rd USENIX Security Symposium*, August 2024. https://gofetch.fail/.

[18] Miles Dai, Riccardo Paccagnella, Miguel Gomez-Garcia, John McCalpin, and Mengjia Yan. Don't mesh around: Side-channel attacks and mitigations on mesh interconnects. In *Proceedings of the 31st USENIX Security Symposium*, pages 2857–2874, August 2022. ISBN 978-1-939133-31-1. https://www.usenix.org/conference/usenixsecurity22/presentation/dai.

[19] *CoreMark PRO.* EEMBC, July 2019. v1.1.2743 https://www.eembc.org/coremark-pro.

[20] Anthony C. J. Fox, Gareth Stockwell, Shale Xiong, Hanno Becker, Dominic P. Mulligan, Gustavo Petri, and Nathan Chong. A verification methodology for the Arm confidential computing architecture: From a secure specification to safe implementations. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1), April 2023. doi: 10.1145/3586040.

[21] Adrien Ghosn, James R. Larus, and Edouard Bugnion. Secured routines: Language-based construction of trusted execution environments. In *Proceedings of the 2019 USENIX Annual Technical Conference*, pages 571–586, Renton, WA, July 2019. ISBN 978-1-939133-03-8. http://www.usenix.org/conference/atc19/presentation/ghosn.

[22] Floris Gorter, Taddeus Kroes, Herbert Bos, and Cristiano Giuffrida. Sticky Tags: Efficient and deterministic spatial memory error mitigation using persistent memory tags. In *Proceedings of the 45th IEEE Symposium on Security and Privacy*, May 2024. doi: 10.1109/SP54263.2024.00263.

[23] Aditya Gulavani, Hormuzd M. Khosravi, Balaji Masanamuthu Chinnathurai, and Shiny Sebastian. Performance considerations of hardware-isolated partitioned VMs with Intel TDX, 2023. https://www.intel.com/content/www/us/en/developer/articles/technical/tdx-performance-isolated-partitioned-vms.html.

[24] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. Protean: VM allocation service at scale. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation*, pages 845–861, November 2020. ISBN 978-1-939133-19-9. https://www.usenix.org/conference/osdi20/presentation/hadary.

[25] Mathe Hertogh, Manuel Wiesinger, Sebastian Österlund, Marius Muench, Nadav Amit, Herbert Bos, and Cristiano Giuffrida. Quarantine: Mitigating transient execution attacks with physical domain isolation. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, October 2023. doi: 10.1145/3607199.3607248.

[26] Mathé Hertogh, Sander Wiebing, and Cristiano Giuffrida. Leaky Address Masking: Exploiting unmasked Spectre gadgets with noncanonical address translation. In *Proceedings of the 45th IEEE Symposium on Security and Privacy*, May 2024. doi: 10.1109/SP54263.2024.00158.

[27] Jana Hofmann, Emanuele Vannacci, Cedric Fournet, Boris Kopf, and Oleksii Oleksenko. Speculation at fault: Modeling and testing microarchitectural leakage of CPU exceptions. In *Proceedings of the 32nd USENIX Security Symposium*, pages 7143–7160, August 2023. ISBN 978-1-939133-37-3. https://www.usenix.org/conference/usenixsecurity23/presentation/hofmann.

[28] *Speculative Behavior of SWAPGS and Segment Registers.* Intel Corp., August 2019. ID 660228 https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/speculative-behavior-swapgs-and-segment-registers.html.

[29] *Snoop-Assisted L1 Data Sampling.* Intel Corp., October 2020. ID 758367 https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/snoop-assisted-l1-data-sampling.html.

[30] *Intel Trust Domain CPU Architectural Extensions.* Intel Corp., September 2020. Ref. #343754-001US https://software.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-cpu-architectural-specification.pdf.

[31] *Intel TDX Connect Architecture Specification.* Intel Corp., May 2021. https://www.intel.com/content/www/us/en/content-details/773614/intel-tdx-connect-architecture-specification.html.

[32] *Processor MMIO Stale Data Vulnerabilities.* Intel Corp., June 2022. ID 758360 https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/processor-mmio-stale-data-vulnerabilities.html.

[33] *Intel SGX Attestation Technical Details.* Intel Corp., July 2023. https://www.intel.com/content/www/us/en/security-center/technical-details/sgx-attestation-technical-details.html.

[34] *Trust Domain Security Guidance for Developers.* Intel Corp., 2023. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/trusted-domain-security-guidance-for-developers.html. Accessed Nov 2023.

[35] *Intel Transactional Synchronization Extension (TSX) Disable Update for Selected Processors.* Intel Corp., June 2023. Document. #643557 rev. 2.3 https://cdrdv2.intel.com/v1/dl/getContent/643557.

[36] Juhee Kim, Jinbum Park, Sihyeon Roh, Jaeyoung Chung, Youngjoo Lee, Taesoo Kim, and Byoungyoung Lee. TikTag: Breaking ARM's memory tagging extension with speculative execution. *CoRR*, abs/2406.08719, June 2024. https://arxiv.org/abs/2406.08719.

[37] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *Proceedings of the 40th IEEE Symposium on Security and Privacy*, pages 1–19, 2019. doi: 10.1109/SP.2019.00002.

[38] Alex Landau, Muli Ben-Yehuda, and Abel Gordon. SplitX: Split guest/hypervisor execution on multi-core. In *3rd Workshop on I/O Virtualization (WIOV 11)*, Portland, OR, June 2011. USENIX. https://www.usenix.org/conference/wiov11/splitx-split-guesthypervisor-execution-multi-core.

[39] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. Design and verification of the Arm confidential compute architecture. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation*, July 2022. https://www.usenix.org/conference/osdi22/presentation/li.

[40] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *Proceedings of the 27th USENIX Security Symposium*, pages 973–990, August 2018. ISBN 978-1-939133-04-5. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

[41] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, pages 605–622, 2015. doi: 10.1109/SP.2015.43.

[42] *Analysis and mitigation of speculative store bypass (CVE-2018-3639).* Microsoft, May 2018. https://msrc.microsoft.com/blog/2018/05/analysis-and-mitigation-of-speculative-store-bypass-cve-2018-3639/.

[43] *Manage Hyper-V hypervisor scheduler types.* Microsoft, May 2023. https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/manage/manage-hyper-v-scheduler-types.

[44] Daniel Moghimi. Downfall: Exploiting speculative data gathering. In *Proceedings of the 32nd USENIX Security Symposium*, pages 7179–7193, August 2023. ISBN 978-1-939133-37-3. https://www.usenix.org/conference/usenixsecurity23/presentation/moghimi.

[45] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against Intel SGX. In *Proceedings of the 41st IEEE Symposium on Security and Privacy*, pages 1466–1482, 2020. doi: 10.1109/SP40000.2020.00057.

[46] William D. Norcott, Don Capps, et al. IOzone filesystem benchmark, May 2023. v3.506 https://www.iozone.org/.

[47] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. Varys: Protecting SGX enclaves from practical side-channel attacks. In *Proceedings of the 2018 USENIX Annual Technical Conference*, pages 227–240, July 2018. ISBN 978-1-939133-01-4. https://www.usenix.org/conference/atc18/presentation/oleksenko.

[48] Oleksii Oleksenko, Marco Guarnieri, Boris Köpf, and Mark Silberstein. Hide and seek with spectres: Efficient discovery of speculative information leaks with random testing. In *Proceedings of the 44th IEEE Symposium on Security and Privacy*, pages 1737–1752, May 2023. doi: 10.1109/SP46215.2023.10179391.

[49] *Hard Partitioning with Oracle Linux KVM.* Oracle, June 2022. https://www.oracle.com/a/ocom/docs/linux/ol-kvm-hard-partitioning.pdf.

[50] Meni Orenbach, Pavel Lifshits, Marina Minkin, and Mark Silberstein. Eleos: ExitLess OS services for SGX enclaves. In *Proceedings of the 12th ACM European Conference on Computer Systems*, pages 238–253, 2017. doi: 10.1145/3064176.3064219.

[51] Tavis Ormandy. Reptar. https://lock.cmpxchg8b.com/reptar.html, November 2023.

[52] Tavis Ormandy. Zenbleed. https://cmpxchg8b.com/zenbleed.html, July 2023.

[53] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. CrossTalk: Speculative data leaks across cores are real. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy*, May 2021. doi: 10.1109/SP40001.2021.00020.

[54] Hany Ragab, Andrea Mambretti, Anil Kurmus, and Cristiano Giuffrida. GhostRace: Exploiting and mitigating speculative race conditions. In *Proceedings of the 33rd USENIX Security Symposium*, August 2024. https://www.vusec.net/projects/ghostrace/.

[55] Joseph Ravichandran, Weon Taek Na, Jay Lang, and Mengjia Yan. PACMAN: Attacking ARM pointer authentication with speculative execution. In *Proceedings of the 49th IEEE International Symposium on Computer Architecture*, 2022. doi: 10.1145/3470496.3527429.

[56] *Redis benchmark.* Redis, 2023. https://redis.io/docs/management/optimization/benchmarks/.

[57] Xida Ren, Logan Moody, Mohammadkazem Taram, Matthew Jordan, Dean M. Tullsen, and Ashish Venkat. I see dead μops: Leaking secrets via Intel/AMD micro-op caches. In *Proceedings of the 48th IEEE International Symposium on Computer Architecture*, pages 361–374, 2021. doi: 10.1109/ISCA52012.2021.00036.

[58] Ravi Sahita, Vedvyas Shanbhogue, Andrew Bresticker, Atul Khare, Atish Patra, Samuel Ortiz, Dylan Reid, and Rajnesh Kanwal. CoVE: Towards confidential computing on RISC-V platforms. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, CF '23, pages 315–321, 2023. doi: 10.1145/3587135.3592168.

[59] Jose Rodrigo Sanchez Vicarte, Pradyumna Shome, Nandeeka Nayak, Caroline Trippel, Adam Morrison, David Kohlbrenner, and Christopher W. Fletcher. Opening Pandora's Box: A systematic study of new ways microarchitecture can leak private data. In *Proceedings of the 48th IEEE International Symposium on Computer Architecture*, pages 347–360, 2021. doi: 10.1109/ISCA52012.2021.00035.

[60] Jose Rodrigo Sanchez Vicarte, Michael Flanders, Riccardo Paccagnella, Grant Garrett-Grossman, Adam Morrison, Christopher W. Fletcher, and David Kohlbrenner. Augury: Using data memory-dependent prefetchers to leak data at rest. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy*, 2022. doi: 10.1109/SP46214.2022.00089.

[61] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. ZombieLoad: Cross-privilege-boundary data sampling. In *Proceedings of the 26th ACM Conference on Computer and Communications Security*, pages 753–768, 2019. doi: 10.1145/3319535.3354252.

[62] Michael Schwarz, Martin Schwarzl, Moritz Lipp, Jon Masters, and Daniel Gruss. NetSpectre: Read arbitrary memory over network. In *Proceedings of the 24th European Symposium on Research in Computer Security (ESORICS 2019)*, pages 279–299. Springer-Verlag, September 2019. doi: 10.1007/978-3-030-29959-0_14.

[63] Quinn Snell, Armin Mikler, and John Gustafson. NetPIPE: A network protocol independent performance evaluator. In *Proceedings of the IASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[64] Livio Soares and Michael Stumm. FlexSC: Flexible system call scheduling with exception-less system calls. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, October 2010. https://www.usenix.org/conference/osdi10/flexsc-flexible-system-call-scheduling-exception-less-system-calls.

[65] Julian Stecklina and Thomas Prescher. LazyFP: Leaking FPU register state using microarchitectural side-channels. *CoRR*, abs/1806.07480, August 2018. https://arxiv.org/abs/1806.07480.

[66] *iTLB multihit*. Tacito Security, 2019. https://www.tacitosecurity.com/multihit.html.

[67] Daniël Trujillo, Johannes Wikner, and Kaveh Razavi. Inception: Exposing new attack surfaces with training in transient execution. In *Proceedings of the 32nd USENIX Security Symposium*, pages 7303–7320, August 2023. ISBN 978-1-939133-37-3. https://www.usenix.org/conference/usenixsecurity23/presentation/trujillo.

[68] *Realm management monitor reference implementation (TF-RMM)*. Trusted Firmware, May 2023. https://github.com/TF-RMM/tf-rmm v0.3.0.

[69] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium*, pages 991–1008, 2018. ISBN 978-1-939133-04-5. https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.

[70] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lippi, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. LVI: Hijacking transient execution through microarchitectural load value injection. In *Proceedings of the 41st IEEE Symposium on Security and Privacy*, pages 54–72, 2020. doi: 10.1109/SP40000.2020.00089.

[71] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue in-flight data load. In *Proceedings of the 40th IEEE Symposium on Security and Privacy*, pages 88–105, May 2019. doi: 10.1109/SP.2019.00087.

[72] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. CacheOut: Leaking data on Intel CPUs via cache evictions. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy*, pages 339–354, 2021. doi: 10.1109/SP40001.2021.00064.

[73] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the 10th ACM European Conference on Computer Systems*, 2015. doi: 10.1145/2741948.2741964.

[74] Ofir Weisse, Valeria Bertacco, and Todd Austin. Regaining lost cycles with HotCalls: A fast interface for SGX secure enclaves. In *Proceedings of the 44th IEEE International Symposium on Computer Architecture*, pages 81–93, 2017. doi: 10.1145/3079856.3080208.

[75] Sander Wiebing, Alvise de Faveri Tron, Herbert Bos, and Cristiano Giuffrida. InSpectre Gadget: Inspecting the residual attack surface of cross-privilege Spectre v2. In *Proceedings of the 33rd USENIX Security Symposium*, August 2024. https://www.vusec.net/projects/native-bhi/.

[76] Johannes Wikner and Kaveh Razavi. RETBLEED: Arbitrary speculative code execution with return instructions. In *Proceedings of the 31st USENIX Security Symposium*, pages 3825–3842, August 2022. ISBN 978-1-939133-31-1. https://www.usenix.org/conference/usenixsecurity22/presentation/wikner.

[77] Wenjie Xiong and Jakub Szefer. Survey of transient execution attacks and their mitigations. *ACM Computing Surveys*, 54(3), May 2021. doi: 10.1145/3442479.

[78] Ruiyi Zhang, Taehyun Kim, Daniel Weber, and Michael Schwarz. (M)WAIT for it: Bridging the gap between microarchitectural and architectural side channels. In *Proceedings of the 32nd USENIX Security Symposium*, pages 7267–7284, August 2023. ISBN 978-1-939133-37-3. https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-ruiyi.

[79] Ruiyi Zhang, Lukas Gerlach, Daniel Weber, Lorenz Hetterich, Youheng Lü, Andreas Kogler, and Michael Schwarz. CacheWarp: Software-based fault injection using selective state reset. In *Proceedings of the 33rd USENIX Security Symposium*, August 2024. https://cachewarpattack.com/.

[80] Ziqiao Zhou, Yizhou Shan, Weidong Cui, Xinyang Ge, Marcus Peinado, and Andrew Baumann. Core slicing: closing the gap between leaky confidential VMs and bare-metal cloud. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation*, pages 247–267, July 2023. ISBN 978-1-939133-34-2. https://www.usenix.org/conference/osdi23/presentation/zhou-ziqiao.