Perspective: A Principled Framework for Pliable and Secure Speculation in Operating Systems*

Tae Hoon Kim, David Rudo, Kaiyang Zhao, [1]Zirui Neil Zhao, and Dimitrios Skarlatos
Carnegie Mellon University, [1]The University of Texas at Austin

1 Summary of the Paper

1.1 Introduction

Transient execution attacks [3] exploit *transient* instructions that may execute but not subsequently commit. The primary building blocks of these attacks are *transient execution gadgets*—code sequences that can speculatively bypass software bounds checks to access sensitive data and transmit the data through a microarchitectural covert channel to the attacker. Transient execution gadgets are potent as *a single gadget can leak any information within the address space in which it resides.*

The operating system (OS) kernel is particularly vulnerable to transient execution attacks. This is because modern OSs like Linux adopt a monolithic kernel address space, to which the entire physical address space is mapped. As a result, *a transient execution gadget in the kernel can leak the entire system memory.* This threat is exacerbated by the growing size and complexity of OS code bases. For example, the Linux kernel has more than 23 million lines of code, making the task of eliminating transient execution gadgets in the kernel virtually impossible. This is evidenced by a continuous stream of new transient execution vulnerabilities in the kernel.

In general, defenses for transient execution attacks can be categorized as software-based and hardware-based approaches. Software-based solutions [1,6] do not require hardware changes and can be deployed to existing systems. However, they require recompiling programs and are often "spot mitigations" with limited protections. Importantly, these solutions incur a high performance overhead. For example, SLH [1] on the Linux kernel shows a system call overhead of 65% on average [84].

Hardware-based defenses [4,5,7,8] aim to be software transparent and backward compatible, but this transparency comes at a cost. Without any information from the software, hardware defenses must be always-on and conservative, leading to a non-negligible performance cost. Additionally, the best-performing hardware solutions require extensive modifications to critical processor components such as the core pipeline and cache hierarchy. As a result, these designs are impractical for adoption due to the high implementation and verification cost.

Our contributions. We present *Perspective*, a principled framework for building efficient, lightweight transient execution defenses for the OS kernel. The key insight of Perspective is to provide a pliable software-hardware interface that allows the OS to communicate its security requirements to the underlying hardware protection mechanisms, enabling selective protection. As a result, the hardware protection mechanism can be as simple as blocking speculative execution of vulnerable instructions while still attaining a low execution overhead.

The design of Perspective is driven by a taxonomy of transient execution attacks and CVEs in the OS kernel. Specifically, we identify two attack scenarios depending on the process that initiates the speculative access to the secret data. These scenarios are: (i) *Active transient execution attacks* or *active attacks*, where the attacker process exploits its own kernel thread to speculatively execute a transient execution gadget in the kernel to access and leak the victim's data. (ii) *Passive transient execution attacks* or *passive attacks*, where the attacker coerces the victim process's kernel thread to speculatively execute a kernel function containing transient execution gadgets, forcing the victim to leak their own data. Note that this taxonomy is agnostic to attack variants, such as Spectre v1, Spectre v2, Retbleed, BHI, and others.

Based on our taxonomy, Perspective introduces two types of speculation views, Data Speculation Views (DSVs) and Instruction Speculation Views (ISVs) to mitigate active and passive attacks respectively. A speculation view can be associated with an execution context such as a process or a container.

Intuitively, a DSV defines a set of kernel data owned by each execution context. If an attacker process speculatively accesses data outside of its DSV, those accesses are protected—e.g., delayed until they are guaranteed to retire. A process's DSV is tracked by the kernel and cannot be tampered with by the attacker process. Therefore, DSVs eliminate the active attacks.

An ISV defines a set of kernel code that an execution context can speculatively execute. Intuitively, an ISV represents the kernel code trusted by the context to be gadget-free. If a process's execution speculatively jumps outside its ISV, the hardware blocks speculative execution of transmitter instructions, whose execution can leak secrets, thus mitigating the passive attack. Unlike DSVs, ISVs are provided by each context and can be generated through static or dynamic program analysis.

The security benefits of ISVs are manyfold. First, ISVs offer an interface for swiftly mitigating newly-discovered gadgets in the kernel code without patching the kernel. Second, programs can exclude infrequently used kernel code from their ISVs to reduce the attack surface, similar to kernel debloating. Lastly, ISV limits the scope of and thus speeds up kernel gadget finding, as an ISV often contains only a small fraction of kernel functions. We then can exclude the vulnerable functions identified by the gadget finding process from the old ISV to form a stricter ISV, further reducing the passive attack surface.

Our security evaluation demonstrates that Perspective eliminates active attacks and reduces the passive attack surface by 95%. It incurs minimal performance overhead: 3.5% on microbenchmarks and 1.2% on datacenter applications, compared to an unprotected kernel.

1.2 Transient Execution Attacks in the OS

We first categorize transient execution attacks in the OS into active and passive transient execution attacks based on the context that initiates the speculative access to the secret data. Importantly, our taxonomy is agnostic to specific variants, such as Spectre v1, Spectre v2, Spectre RSB, and others.

*The full paper is "Perspective: A Principled Framework for Pliable and Secure Speculation in Operating Systems", which appears in proceedings of the 51st International Symposium on Computer Architecture (ISCA 2024).

Active Transient Execution Attacks. The attacker process's kernel thread directly initiates unauthorized speculative accesses to the victim's secret data. For instance, after the attacker identifies a transient execution gadget inside a kernel function, the attacker can exploit it by crafting system call arguments to trigger the vulnerable function.

Passive Transient Execution Attacks. The attacker coerces a victim's kernel thread to execute kernel functions containing transient execution gadgets and leak the victim's secrets. This attack can be achieved by speculatively hijacking the victim's control flow into the vulnerable function, using techniques like Spectre v2 and Retbleed. Unlike active attacks, the victim's kernel thread speculatively accesses and transmits its own data in a passive attack.

Study of Transient Execution Attack CVEs in the Kernel. To better understand the threats posed by transient execution attacks, we then examine vulnerabilities in the Linux kernel. Our study reveals that transient execution gadgets are often hidden within rarely used code, making them difficult to detect. Moreover, OS defenses against speculative control-flow hijacking also frequently fall short. These vulnerabilities can be exploited through both active and passive attacks.

1.3 Perspective Design

Perspective's design is based on our taxonomy to introduce two types of speculation views: Data Speculation Views (DSVs) and Instruction Speculation Views (ISVs). Perspective's principled approach based on this taxonomy, instead of attack variants, enables Perspective to defend against all attack variants.

Data Speculation Views (DSVs). DSVs define which data can be speculatively accessed by a given execution context. Any speculative access to data outside the DSV is protected. For simplicity, we assume a simple but aggressive protection mechanism that blocks a speculative memory access until it becomes non-speculative. As the majority of accesses during kernel execution do not violate ownership, DSVs impose very low execution overhead even if the protection mechanism is as aggressive as blocking speculative accesses.

Figure 1 illustrates how DSVs work. The userspace attacker process $Proc_a$ and victim process $Proc_v$ live on the upper side of the figure, while the monolithic kernel space contains the kernel code and data. In Figure 1, DSVs are depicted as a gray pattern with a colored border to represent the process that owns the data. When process $Proc_a$ executes vulnerable function Ⓒ that contains a transient execution gadget, $Proc_a$ is blocked from speculatively accessing and leaking data belonging to $Proc_v$. Yet, the speculative access to $Proc_a$'s data from function Ⓕ is allowed. This mechanism effectively eliminates active attacks by preventing unauthorized speculative access across DSV boundaries.

One potential DSV design is to define data ownership based on memory allocation. Perspective distinguishes two types of memory allocations: explicit and implicit allocation. Explicit allocations occur when the kernel directly allocates resources requested by userspace, such as memory, sockets, or files. Implicit allocations, on the other hand, are made by the kernel for bookkeeping and metadata management.

Implicit allocations pose a unique challenge, as they are managed by the slab allocator, which optimizes memory usage by packing allocations together. In modern OSs like Linux, data from mutually distrusting processes may end up allocated within the same cache line, creating potential security risks. To address this, Perspective introduces a secure slab allocator that ensures strong isolation across contexts.

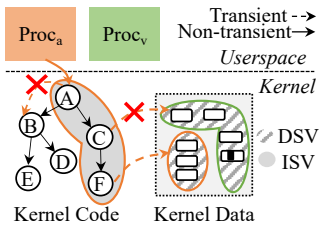

Figure 1: An example of data speculation views (DSVs) and instruction speculation views (ISVs).

Instruction Speculation Views (ISVs). ISVs define which kernel instructions can be speculatively executed. To simplify the discussion, we assume that ISVs are defined at a kernel function granularity. However, in practice, ISV protection is applied at an instruction granularity such that any transmitter instructions from kernel functions outside the ISVs are protected such as being blocked from speculative execution. The ISV becomes active when a process transitions from userspace to the kernel space, e.g., due to a system call or interrupt.

Figure 1 illustrates the ISV of $Proc_a$ using a call graph at the bottom left of the figure. Kernel functions that belong to its ISV are enclosed in the orange line. Now consider the control-flow edge from function Ⓐ to function Ⓑ. Since Ⓑ is outside the ISV, the program cannot speculatively execute transmitter instructions from node Ⓑ.

Perspective introduces two techniques—static and dynamic—to define ISVs for an application, by marrying concepts from system call interposition used for application sandboxing. First, we identify the application's potential entry points to the kernel. Then, using static or dynamic analysis, we determine the kernel functions that these entry points may access. These kernel functions form the application's trusted ISV. Unlike system call interposition, ISVs can be conservative, as the kernel can non-speculatively execute functions outside the ISV instead of crashing. Finally, Perspective uses transient execution gadget detection tools like Kasper [2] to analyze the ISVs, exclude vulnerable functions, and create a stricter ISV—*ISV++*.

We discuss the full OS and hardware design and implementation of DSV and ISVs in Sections V and VI of the paper.

1.4 Evaluation

We build Perspective's software components in the Linux kernel and model its hardware in gem5. We evaluate Perspective using a microbenchmark suite for Linux, as well as a suite of datacenter applications: Redis, Apache, Nginx, and Memcached.

Security Evaluation. Perspective's DSVs eliminate active attacks by blocking speculative accesses that violate ownership. For passive attacks, Perspective's ISVs dramatically reduce the attack surface. On average, static ISVs reduce speculatively executable functions by 91%, while dynamic ISVs reduce this further to 95%. Our analysis, based on Kasper [2], shows that Perspective blocks over 91% of potential speculative execution gadgets with dynamic ISVs, and 100% with stricter ISVs++. Lastly, by drastically narrowing the gadget search space to ISVs, Perspective accelerates the Kasper gadget discovery rate by an average of 1.57×, and up to 2.23×.

Performance Evaluation. Perspective has a minimal execution overhead over an unprotected kernel of 3.5% on microbenchmarks and 1.2% on datacenter applications, significantly outperforming other solutions like Fencing.

Please see Sections VIII and IX of the paper for more details.

2 Potential for Long-Term Impact

Perspective introduces the first principled framework for secure speculation in operating systems. It makes numerous contributions that cut across security, operating systems, and computer architecture that can lead to long-term impact.

1. Perspective presents a detailed study of speculative execution vulnerabilities in Linux since Spectre in 2018.

Perspective offers a comprehensive study of speculative execution vulnerabilities in Linux, analyzing vulnerabilities through source code origins, Linux CVEs, and relevant academic research. Key findings reveal: (i) transient execution gadgets are often concealed within rarely used code paths, making detection challenging. (ii) OS defenses against speculative control-flow hijacking frequently fall short due to inadequate or misapplied mitigations. (iii) Patches for vulnerabilities sometimes introduce new gadgets, necessitating multiple updates. (iv) An attacker can directly exploit, or coerce a victim kernel thread to exploit, an unpatched transient execution gadget deep within the 23 million lines of kernel code.

2. Perspective presents a variant-agnostic taxonomy of transient execution attack scenarios, classifying them into active and passive types, allowing OS designers and hardware architects to address mitigations holistically.

Perspective is designed around a taxonomy of transient execution attacks in the OS kernel, identifying two main scenarios: (i) active attacks, where an attacker exploits its own kernel thread to speculatively execute a transient execution gadget, and (ii) passive attacks, where the attacker induces a victim's kernel thread to speculatively execute a vulnerable function. This taxonomy is variant-agnostic, covering Spectre v1, Spectre v2, Retbleed, and others, allowing researchers, OS developers, and hardware architects to systematically address transient execution attacks and mitigations.

3. Perspective introduces the first principled framework for pliable and secure speculative execution in the OS.

Perspective is the first framework for efficient, lightweight speculative execution in the OS kernel, offering a flexible interface for the OS to communicate security requirements to hardware protection mechanisms. It achieves this with two types of Speculation Views: Data Speculation Views (DSVs) and Instruction Speculation Views (ISVs), which mitigate active and passive attacks, respectively. DSVs define permissible data access during speculative execution based on data ownership, while ISVs allow an execution context to specify trusted kernel functions, blocking speculative execution of instructions outside these functions. Perspective opens a new design space for tailored protection, reducing unnecessary safeguards, reducing hardware complexity, and minimizing overhead.

4. Perspective provides robust security and performance benefits while drastically simplifying hardware.

Perspective delivers strong security guarantees with minimal performance overhead. DSVs fully eliminate active attacks in the OS kernel, a primary attack surface. ISVs add multiple layers of security: they offer a swift interface for mitigating newly discovered vulnerabilities in kernel functions, essential given the ongoing identification of speculative execution vulnerabilities. ISVs also allow a victim program to exclude rarely used kernel functions, similar to kernel debloating, thereby reducing passive attack surfaces. Furthermore, by blocking speculative execution outside of specified ISVs, security audits can focus on a smaller subset of kernel functions, streamlining the process and allowing the exclusion of identified gadgets, further strengthening security.

5. Perspective is receiving major interest from Intel. Since publishing the paper, we've engaged in discussions and presentations with Intel security researchers and architects. We are now exploring ways to upstream standalone software components of Perspective into Linux, focusing on deployable DSV elements like the secure slab allocator, which can be integrated into current architectures without hardware changes.

6. Perspective opens avenues for future work on static analysis of Instruction Speculation Views.

The Perspective ISV security model significantly reduces the Linux codebase needing audit—from 23 MLOC to a few functions, reducing the attack surface by 95%. In Perspective, we explored techniques like fuzzing and taint tracking for identifying gadgets. ISVs greatly accelerate the auditing process and can enable previously intractable analyses, such as symbolic execution, for finding gadgets in the kernel. We believe this approach can drive the elimination of gadgets in large codebases.

7. Perspective is a versatile framework applicable to multi-domain environments, including userspace libraries and runtimes.

Perspective can extend beyond OS-level applications to other multi-domain sandboxing environments, such as internet browsers (e.g., Chrome), language runtimes (e.g., JVM) and sandboxing technologies (e.g., WebAssembly). We believe Perspective will catalyze research into principled and secure speculation across these platforms.

8. Perspective paves the way for future development of leakage-free operating systems and hypervisors.

We believe Perspective opens a promising research path toward a leakage-free operating system by design. Perspective outlines key requirements for redesigning or developing clean-slate OSs and hypervisors to eliminate speculative vulnerabilities. Specifically, Perspective advocates for strict data ownership and can enable the use of high-level languages (e.g., Rust) to prevent both active and passive attacks from the ground up.

Citation for the paper if it won the test of time award:

This paper introduced Perspective, the first principled framework for secure speculation in operating systems.

REFERENCES

[1] C. Carruth, "Speculative Load Hardening." `https://llvm.org/docs/SpeculativeLoadHardening.html`, 2018.

[2] B. Johannesmeyer, J. Koschel, K. Razavi, H. Bos, and C. Giuffrida, "Kasper: Scanning for Generalized Transient Execution Gadgets in the Linux Kernel," in *NDSS*, Apr. 2022.

[3] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *IEEE S&P*, 2019.

[4] E. M. Koruyeh, S. Haji Amin Shirazi, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "SpecCFI: Mitigating Spectre attacks using CFI informed speculation," in *IEEE S&P*, 2020.

[5] G. Saileshwar and M. K. Qureshi, "CleanupSpec: An "undo" approach to safe speculation," in *MICRO*, 2019.

[6] P. Turner, "Retpoline: a software construct for preventing branch-target-injection." `https://support.google.com/faqs/answer/7625886`, 2018.

[7] M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. W. Fletcher, and J. Torrellas, "Invisispec: Making speculative execution invisible in the cache hierarchy," in *MICRO*, 2018.

[8] J. Yu, M. Yan, A. Khyzha, A. Morrison, J. Torrellas, and C. W. Fletcher, "Speculative Taint Tracking (STT): A comprehensive protection for speculatively accessed data," in *MICRO*, 2019.