Scilight

Pragmatic Cybersecurity
https://www.sciltp.com/journals/pc

Article

RV-Sec5: Enhancing Pre-Silicon Security Evaluation of RISC-V Processors through Targeted ISA-Level Instrumentation in the gem5 Simulation Framework †

Muhammad Awais [1,*], Maria Mushtaq [1], Lirida Naviner [1], Jawad Haj Yahya [2] and Florent Bruguier [3]

[1] COMELEC Department, Télécom Paris, Polytechnic Institute of Paris, 91120 Palaiseau, France
[2] Meta, 8001 Zurich, Switzerland
[3] Centre National de la Recherche Scientifique (CNRS), Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), University of Montpellier, 34090 Montpellier, France
* Correspondence: muhammad.awais@telecom-paris.fr

† Extended from: RV-Sec5: Enhancing RISC-V Security Evaluation via Targeted ISA-Level Instrumentation Using gem5. In Proceedings of the 2026 Australasian Information Security Conference, Melbourne, VIC, Australia, 11–12 February 2026.

How To Cite: Awais, M.; Mushtaq, M.; Naviner, L.; et al. RV-Sec5: Enhancing Pre-Silicon Security Evaluation of RISC-V Processors through Targeted ISA-Level Instrumentation in the gem5 Simulation Framework. *Pragmatic Cybersecurity* **2026**, *1*(1), 6. https://doi.org/10.53941/pc.2026.100006

Received: 27 March 2026
Revised: 16 June 2026
Accepted: 17 June 2026
Published: 29 June 2026

Abstract: The open-source RISC-V Instruction Set Architecture (ISA) is being adopted rapidly in security-sensitive areas such as IoT, edge computing, and aerospace systems, which makes early-stage security validation increasingly important. Yet most existing approaches still depend either on post-silicon testing or on high-level emulation. Neither is well suited to exposing ISA-specific vulnerabilities or microarchitectural side effects during the design phase. As a result, there remains a gap between high-level security policies and the way hardware actually behaves at runtime, and that gap can leave processors exposed to privilege escalation, memory protection failures, and side-channel leakage that may only become visible late in development. In this paper, we present RV-Sec5, a systematic and policy-driven framework for ISA-level security evaluation built on the gem5 cycle-accurate simulator. RV-Sec5 provides a formal method for translating high-level security invariants, including privilege isolation, Physical Memory Protection (PMP) enforcement, and Control and Status Register (CSR) integrity, into automated cycle-accurate instrumentation points embedded directly in the ISA decoder. By recording precise architectural execution context at instruction commit time, the framework supports specification-driven methodology of privilege escalation attempts and enables systematic correlation between ISA-level events and microarchitectural behavior, including TLB activity and cache state changes, without interfering with functional execution. Our results shows that RV-Sec5 can significantly detects the specification violation on the events that are permitted in User mode, the results shows that after extending the gem5 with the hooks added and ISA modified, it create an overhead on the simulation. The overall overhead of RV-Sec5 for the simulation time is less than 4% and the overhead for the memory usage is less than 2% across the evaluated workloads. RV-Sec5 is a modular, cycle-accurate observation and post-execution detection virtual platform that reduces the gap between architectural security requirements and their enforcement at the Microarchitectural level, using post-silicon testing within the RISC-V processor design flow.

Keywords: RISC-V ISA security; security and privacy; side-channel analysis and countermeasures; hardware security; attacks simulation

1. Introduction

RISC-V, a royalty-free, modular ISA with an open license, has gained prominence in the current computing landscape because of its ability to produce secure and flexible processors in multiple fields. The ability to freely license the ISA while allowing modular additions through standard extensions has led to its widespread adoption in both academia and industrial circles with applications in multiple industries including servers and other high-performance computing systems as well as safety-critical applications. The growing application of RISC-V processors to sensitive applications including IoT, edge computing, automotive applications, and aerospace engineering has created a need for extensive security evaluations during the early design phases of the processor. In contrast, more established architectures that have been subject to post-silicon analysis for several decades lack this issue; in their case, the methodology and tools involved in the process have had ample time to reach maturity. This makes security assurance much more difficult in the context of RISC-V [1].

Nevertheless, existing techniques are still subject to several limitations if employed in the context of validating RISC-V processors. Post-silicon validation, while thorough, uncovers security issues after fabrication, at a point where any modifications require substantial costs and time investment. Higher-level emulator environments increase performance but sacrifice microarchitectural accuracy, thus failing to accurately detect timing-based attacks and privilege violations in the architecture. On the other hand, functional simulators, including Spike [2], precisely simulate ISA-level behavior but fail to provide any insights into the microarchitecture, needed to detect potential issues related to cache operations, Translation Lookaside Buffers (TLB) status, and pipeline behavior

Although cycle-accurate simulators, such as gem5 [3], are widely employed for modeling the interaction between software and hardware with great accuracy, the existing methodologies for security analysis based on these simulators typically rely on general instrumentation techniques. For example, they generally track the statistics of code execution, including instruction counts and cache miss rates, without having any semantic knowledge about the security-relevant behaviors of the ISA. Specifically, there is no explicit instrumentation for observing the privilege mode change, Physical Memory Protection (PMP) configuration modification, and CSR update operations, which are the essential security building blocks of the RISC-V architecture. The aforementioned semantic discrepancy between abstract security policies and low-level simulation events poses major constraints in identifying bugs induced by complex interactions between the ISA logic and microarchitecture. Most importantly, it hinders developers from verifying architectural security invariants, i.e., the requirements that must be satisfied at every correct execution path.

Our work has the following contributions:

- **Policy-driven Approach:** We introduce a specification-driven methodology that systematically derives instrumentation points at the ISA decoder level and applies hooks in the CPU to observe privilege-sensitive events, enabling mapping of high-level security requirements to RISC-V microarchitectural execution.
- **RV-Sec5 Framework Architecture:** A modular framework on gem5 is introduced that captures the architectural execution context of committed instructions, including Program Counter, register state, and CSR values, to enable post-execution detection of privilege escalation attempts and PMP violations.
- **Correlation of Cross-Layer Events:** A mechanism is developed to correlate ISA-level events with microarchitectural statistics (TLB flushes, cache activity, pipeline stalls) at cycle granularity, enabling identification of enforcement-induced side-channel leakage paths that are invisible to functional simulators.
- **Security Invariant Detection and Logging:** A rule-based offline compliance engine checks policy violations against logged execution traces. Although the framework focuses on design-time detection and logging, it establishes the infrastructure needed for studying runtime hardware-level mitigation approaches.

To define the scope of the evaluation, Section 3 introduces a threat model focused on privilege escalation and unauthorized resource access. In this model, the attacker executes unprivileged code in U-mode and attempts to violate the architectural boundaries defined by the RISC-V privileged specification.

The threat model is deliberately based on a violation scenario specified by the specification, enabling us to assess whether a certain microarchitectural implementation is compliant with the security contracts defined for such attacks.

This paper builds on our AISC 2026 conference paper *RV-Sec5: Enhancing RISC-V Security Evaluation via Targeted ISA-Level Instrumentation using gem5* published in AISC 2026 [4]. Compared with the conference version, the present manuscript extends the threat model with an explicit description of the protected assets and security objectives, conducts a more comprehensive evaluation of the framework for new attack scenarios and all three observation channels, and performs an overhead evaluation that involves simulation time, memory usage, and log size analysis.

The structure of the paper is as follows. In Section 2, we discuss the background and existing related work for RISC-V simulation tools and pre-silicon security validation techniques. In Section 3, we describe the threat model under consideration, elaborating on the adversary model, asset model, and attack scenarios. In Section 4, we

introduce our proposed policy-driven pre-silicon security validation methodology. In Section 5, we demonstrate how this methodology can be implemented using gem5. We focus in particular on event interception at the ISA level, capturing architectural context, and logging. In Section 6, we present the results of our experiments, including invariant compliance and coverage as well as side-channel correlation analysis. In Section 7, we analyze the overhead in terms of simulation time, memory usage, and log sizes. In Section 8, we summarize the results. In Section 9, we discuss the limitations of our approach and possible future directions of this research. Section 10 concludes the paper.

2. Background and Related Work

The security of RISC-V systems is depends on the interaction between the abstract definition of the Instruction Set Architecture (ISA), and its physical manifestation in microarchitecture [1]. Although the ISA dictates how a hardware platform will behave as well as its privileges, many important aspects of security come from hidden states within microarchitectures [5]. This section reviews the security foundations of the RISC-V privileged architecture and summarizes current pre-silicon evaluation methods for processor-core security.

2.1. RISC-V Privileged Architecture and Security Background

The RISC-V ISA [1] represents an open standard architecture that is flexible in terms of extensibility and supports a wide range of applications from embedded to general-purpose as well as security-focused computing tasks. The RISC-V privileged architecture implements a hierarchy of modes of execution called Machine mode (M), Supervisor mode (S), and User mode (U).

The core mechanisms that implement security policies in RISC-V architecture are minimalistic in number but serve as a functional foundation for achieving privilege separation and memory protection. They include specialized control-flow instructions (`ecall`, `mret`), along with some bits in the `mstatus` register [1]. Hardware-based Physical Memory Protection (PMP) specifies access permissions over memory pages in physical address space, allowing isolation in non-virtualized systems [6]. Moreover, optional security extensions like Zk and Smepmp offer improvements in cryptographic functions and memory protection respectively [7] that are shown in Table 1.

Table 1. Security-Affected Aspects of the RISC-V ISA. The table summarizes the main architectural mechanisms that govern privilege separation and memory isolation in RISC-V systems and define the security invariants monitored by RV-Sec5.

Component	Security Role
Privilege Mode Transitions [1]	Manages the boundaries of execution in M, S, and U modes using `mstatus` and trap-return instructions, ensuring no unprivileged execution occurs.
Physical Memory Protection (PMP) [6]	Provides access control for regions of physical memory in terms of read, write, and execute operations.
Security Extensions [8]	Optional security extensions (such as Zk and Smepmp), which extend the core ISA with cryptographic operations and enhanced isolation capabilities.

Even though these components establish architectural guidelines, the RISC-V specification deliberately ignores many microarchitecture-specific constructs like cache hierarchies, branch predictors, and TLBs) [9]. Such an abstraction facilitates freedom of implementation, but introduces a semantic disconnect between the architectural notion of correctness and actual microarchitectural behavior. Attacks such as Spectre [10] and Meltdown [11] exploit this gap by manipulating hidden microarchitectural state. Consequently, architectural security verification alone cannot detect a vast array of modern exploits. Therefore, pre-silicon security verification techniques need to relate ISA semantics with microarchitectural effects [12] so as to determine if the guarantees offered by the ISA would still be valid when implemented in hardware. As attacks exploiting this disconnect prove time and again, such as with transient execution vulnerabilities Spectre [10] and Meltdown [11], a processor that follows ISA semantics can still disclose information via microarchitectural side channels, a category of vulnerabilities that cannot be predicted with ISA-level verification alone. Any methodology for effective pre-silicon verification needs to consider security properties from both perspectives simultaneously. RV-Sec5 tackles this problem by introducing ISA-aware observability in a cycle-accurate simulation framework. Through carefully crafted hooks inserted into the gem5 ISA decoder, RV-Sec5 is able to correlate security-relevant instructions with their architectural context as well as any related microarchitectural activity, such as TLB flushes and cache hits/misses

2.2. Instructions Relevant to Security and Attack Surfaces

Within the RISC-V privileged ISA [1], there is a group of instructions and CSRs [13] that have a particularly significant impact on the security status of the whole system. These particular instructions and registers can change system privileges, memory protection, or address translation settings, which makes them attractive attack surfaces. Table 2 shows some relevant instructions and CSRs.

Table 2. Security-relevant RISC-V instructions and CSRs exploited by RV-Sec5. These architectural interfaces, when misused or accessed without proper authorization, may result in violations of privilege isolation and memory protection guarantees.

Instruction/CSR	Function	Security Relevance
`CSRRW` [13]	Atomically reads/writes CSRs	A direct modification of privilege-related CSRs like `mstatus` by unprivileged processes may lead to an elevation of execution privileges; hence, this instruction is considered a major target for breaking the privilege boundary.
`SFENCE.VMA` [13]	Invalidates TLB entries	An improper or out-of-order TLB flush operation may provide opportunities for unauthorized access to stale address translation entries or even timing channel attacks that could be exploited by unprivileged code.
`menvcfg` [13]	Configures the operation of PMP	Any modification without proper authorization may lead to reduced effectiveness of physical memory protection, allowing unprivileged code to violate access controls enforced by PMP in protected memory spaces.
`DRET` [13]	Debug return	Unintended use or use without proper authorization may lead to unexpected changes in mode from debug to privileged modes of execution.

Such attacks usually involve either *privilege escalation* by modifying the `mstatus.MPP` register or *memory corruption* by circumventing the PMP-enforced isolation [6]. Notably, even though the architectural behavior might be valid, these attacks could still violate the security invariants because the consequences of their actions could be limited to microarchitectural effects only. For this reason, RV-Sec5 treats these instructions as observable events for detecting security-relevant violations.

2.3. Simulation Environments for Security Analysis

Pre-silicon security analysis requires balancing accuracy and observability in simulation tools. Cycle accurate simulators like `gem5` [3], which can simulate the behavior of the pipeline, cache, and memory hierarchy in great detail, fit this requirement. Nevertheless, `gem5` does not intrinsically support the generation of security-critical events or privilege levels.

On the other hand, other tools described in Table 3 can be found on various levels of the abstraction ladder. Functional simulators like `Spike` [2], which are primarily concerned with ISA compliance, fail to deliver timing data; thus, they cannot be used for side-channel attacks. Emulation systems such as `QEMU` [14] are highly performant and easy to use; however, they do not allow for microarchitecture modeling. On the other hand, formal verification tools like `Sail-RISCV` [15] by Armstrong et al. (POPL 2019), provide mathematically precise specifications but cannot model dynamic execution effects or enforcement behavior.

Table 3. Comparison of RISC-V simulation frameworks with respect to security and microarchitectural analysis capabilities. Each tool serves a distinct purpose in the validation ecosystem, but none natively provides the policy-driven, ISA-level observability required for systematic pre-silicon security evaluation. This gap motivates the design of RV-Sec5.

Simulator	Model Type	Security-Relevant Capabilities	Key Limitations
`gem5` [3]	Cycle-accurate, full-system	Fine-grained microarchitectural modeling (pipelines, caches, TLBs); extensible event system with ISA-level instrumentation capabilities correlated against microarchitecture.	Configuration complexity and simulation overhead scale poorly, making it impractical for very large workloads. Of critical importance, gem5 offers no native security policies or privilege-based instrumentation capabilities; custom ISA-level hooks need to be provided, as in RV-Sec5.
`Spike` [2]	Functional (ISA reference)	Official ISA reference simulator for conformance checks and differential functional validation.	Fully functional but lacks timing or microarchitectural state awareness. Unable to measure cache/TLB effects or pipeline behavior; completely unsuitable for any kind of side-channel attack analysis and microarchitectural security validation.

Table 3. *Cont.*

Simulator	Model Type	Security-Relevant Capabilities	Key Limitations
QEMU [14]	Dynamic binary translation	Fast emulation, useful in OS/firmware bring-ups and software stack validation.	Binary translation removes the notion of CPU timing altogether. Microarchitectural elements like caches and TLBs are not modeled at all, which makes it unsuitable for side-channel or microarchitectural-based security research.
Sail-RISCV [15]	Formal executable specification	Formal specification of ISA semantics; can serve as a basis for formal proofs of security properties.	Extremely slow and unsuitable for realistic workload simulation. It focuses on static architectural semantics without providing dynamic execution and microarchitectural behavior, which limits its usefulness for practical security evaluation.
Renode [16]	System-level co-simulation	Simulation of heterogeneous SoC, peripheral devices, and networking; appropriate for firmware and IoT-related security studies.	CPU timing is inaccurate and microarchitectural elements are absent. It is useful for validating system-level functionality but lacks support for privilege-boundary or side-channel analysis at the instruction level.
Dromajo [17]	Differential functional simulator	Enables lockstep co-simulation with RTL for validation purposes.	Intended for functional validation rather than security analysis; lacks any notion of microarchitecture or timing, preventing correlation between ISA-level events and their side effects at the microarchitectural level.
Whisper [18]	Instruction-set simulator (ISS)	Popular industrial tool used in firmware validation flows, such as secure boot process validation.	Complex documentation and harder to reproduce; has limited applicability as a security tool because it lacks microarchitectural modeling and timing support, making it unsuitable for detailed pre-silicon security analysis.

2.4. Pre-Silicon Security Analysis of RISC-V Systems

Previous works on RISC-V security include surveys, formal approaches, simulation studies, and post-silicon testing. The work of Gao et al. [12] offers an extensive survey of security extensions in RISC-V processors and potential attacks but mainly focuses on post-silicon evaluation, hindering proactive measures to mitigate security weaknesses. Formal frameworks allow one to precisely specify architectural behavior but lack capabilities for monitoring architecture execution and validation of security policies. On the other hand, simulation studies provide more insight into the execution behavior and represent a pragmatic way to avoid post-silicon testing when investigating security issues. Previous works have used gem5 to simulate full RISC-V systems to analyze the microarchitectural consequences of their execution [19]. However, current studies mostly utilize coarse-grain trace logging, e.g., per-core statistics on cache misses or instructions retirements, and fine-grain instruction-level hooks to detect privileged operations like CSR writes, PMP configuration changes, and privilege level switches. Thus, it is hard to identify security policy violations based on some specific ISA event. The coarse-grained nature of such instrumentation cannot be used as proof that a given implementation respects the security invariants that the RISC-V privileged specification defines. On the other hand, empirical research performed using actual physical devices, as done by Gerlach et al. [20], provides compelling proof of the real-world feasibility and exploitation potential of microarchitectural attacks against RISC-V processors. While such post-silicon experiments yield important results about the true vulnerability of the system, there are two critical shortcomings when considered in light of designing secure processors. First, experiments performed using actual hardware cannot be easily replicated because the results are very sensitive to the silicon design, board setup, and measurement apparatus. Second, and most importantly, post-silicon analysis cannot be done before production since any changes required to eliminate vulnerabilities will require a costly respin of the device. This is precisely the motivation behind RV-Sec5: the need for a systematic approach to analyzing the security of processors prior to production by combining the microarchitectural accuracy of simulation with security-semantics knowledge that current simulators are lacking.

The above-discussed approaches have been summarized in Table 4, and their difference compared to RV-Sec5 has been identified. It has become apparent that all of the current solutions are characterized by the same drawback – while surveys and post-silicon analysis miss an executable pre-silicon verification component, formal methods offer a static definition of the rules while lacking a dynamic monitoring part, simulation techniques suffer from using inadequate instrumentation which is not able to trace privilege level events and hardware measurement provides practical data at the expense of irreproducibility and non-applicability for the design phase. Instead of developing another tool-focused approach, RV-Sec5 proposes a methodology for translating the high-level ISA policy into microarchitectural observations.

Table 4. Comparison of current RISC-V security evaluation schemes to RV-Sec5. Current approaches cover different facets of security evaluation individually but lack support for execution-based, pre-silicon, and microarchitecturally aware validation with ISA-level policy-driven instrumentation.

Study	Methodology	Instrumentation	Key Limitation
Waterman et al. [1]	Formal specification	Static verification	Strictly defines architectural behaviors but fails to incorporate dynamic enforcement mechanisms in the form of runtime instrumentation.
Gao et al. [12]	Survey/Post-silicon	None	Provides extensive coverage of threat models but lacks any executable method for validation; vulnerabilities must be detected after fabrication, by which time fixing design problems becomes prohibitively expensive.
Paul et al. [19]	`gem5` Simulation	Limited probes	Leverages low-resolution hardware performance counters without ISA-specific event hooks and thus cannot trace violations tied to accesses to CSRs or changes in privilege levels.
Gerlach et al. [20]	Hardware measurement	Physical probes	Proves exploitability on fabricated devices but confines the findings strictly to individual devices, limiting their reproducibility and thus precluding applicability prior to fabrication.
Ahmadi et al. [21]	Theoretical analysis	None	Analytically defines attack vectors for side-channel exploits without providing a method for validating their behavior through simulation or tool-assisted verification.
RV-Sec5 (Ours)	**Policy-driven methodology**	**ISA-level hooks**	**Brings architectural and microarchitectural behaviors in line with one another in cycle-accurate simulation, allowing for reproducible pre-silicon validation of privilege and side channel enforcement.**

The RV-Sec5 framework extends the microarchitecture-level precision of `gem5` through an instrumentation paradigm that is informed by policy, which is absent in existing simulation techniques. Through its explicit association between architectural security invariants (such as privilege isolation and PMP enforcement) and cycle-accurate microarchitectural events, the RV-Sec5 framework facilitates the evaluation not only of whether privilege policies are being accurately enforced but also whether such policies inadvertently open side-channel exploitation pathways. This multi-level observability feature permits the detection and analysis of specific vulnerabilities prior to the physical implementation of the RISC-V processor design cycle.

3. Threat Model

This work focuses on the pre-silicon security validation of RISC-V processors by examining whether architectural security invariants are correctly enforced by the underlying microarchitectural implementation. The current research is concerned with pre-silicon security validation of the RISC-V processors through assessing the implementation of architectural security invariants by the underlying microarchitecture. The threat model has been intentionally restricted to a set of attack models that are reflective of real-life design situations but also can be observed in the cycle accurate simulation environment.

3.1. Adversary Model

An attacker is assumed to possess control over unprivileged software running on a target RISC-V processor. This attacker has the ability to execute any kind of U-mode code, specifically designed assembly payloads, which seek to subvert the security properties of the architecture. It is assumed that the attacker does not have any physical access to the device and cannot tamper with any of its hardware/firmware.

In particular, the adversary may:

- Issue arbitrary U-mode instruction sequences, possibly exploiting specific instruction sequences that challenge privilege level changes and memory protection.
- Exploit unauthorized accesses to the CSR, e.g., writes to `mstatus` or PMP control registers.
- Trigger microarchitectural activity, for example, TLB flushes and speculative execution paths, using valid instructions from an adversarial perspective.

It is not assumed that the adversary will take advantage of any unknown vulnerabilities or behaviors of the hardware. Rather, all attacks follow the architectural specification, with an intent of violating security properties that are enforced within the model.

3.2. Assets and Security Objectives

The security of the RISC-V processor depends upon proper architectural boundary adherence and lack of any unintended side effects in the microarchitectural implementation of the processor. Under the threat model, we have listed out a few assets that need to be protected while the instruction stream is being executed. A RISC-V processor

can only be considered secure if three fundamental properties hold simultaneously during execution. The first is privilege isolation, the guarantee that software running at a lower privilege level cannot acquire execution rights it was never granted. Without this property, the entire privilege hierarchy defined by the RISC-V specification loses its meaning, and any boundary between user-mode applications and trusted system firmware becomes effectively unenforceable. The second property is memory isolation, which relies on hardware-enforced mechanisms such as Physical Memory Protection (PMP) to ensure that one software component cannot read, write, or execute memory belonging to another, a guarantee that is equally critical in deeply embedded systems and in general-purpose computing environments. The third property, enforcement integrity, is perhaps the most subtle: it is not enough for a processor to correctly enforce its security rules; the act of enforcement must itself be free of unintended microarchitectural side effects such as timing variations, cache perturbations, or TLB artifacts that an adversary could observe and exploit as a side channel. These three properties, summarized in Table 5, collectively define the security contract that RV-Sec5 systematically validates at the boundary between the architectural specification and its microarchitectural realization.

Table 5. Primary security assets considered in this work. The table summarizes the core security assets targeted by RV-Sec5, namely privilege isolation, memory isolation, and enforcement integrity, together with their roles in preserving secure execution.

Asset	Description
Privilege Isolation	Prevents unauthorized transitions across privilege levels, ensuring that execution remains confined to the permissions defined by the current privilege mode.
Memory Isolation	Ensures the integrity and confidentiality of memory regions through hardware-enforced mechanisms such as Physical Memory Protection (PMP), restricting unauthorized access.
Enforcement Integrity	Guarantees that the enforcement of architectural security rules does not introduce unintended microarchitectural side effects (e.g., timing, cache, or TLB artifacts) that could be exploited as side channels.

Based on these assets, the security objectives of this work focus on detecting violations of architectural guarantees and identifying potential leakage paths. These objectives are summarized in Table 6.

Specifically, this research attempts to identify instances where there is: (i) any unauthorized modification of CSRs related to privileges, such as `mstatus.MPP`, which could lead to privilege escalation; (ii) illegal writes to PMP CSRs as a result of inadequate privileges, and therefore memory isolation may be violated; and (iii) irregular microarchitectural behavior related to blocked and illegal ISA-level events. The combination of these tasks helps validate the proper enforcement of security invariants without exploiting any side channels.

Table 6. Security goals and corresponding violations. The table maps each security objective to its targeted violation, covering privilege enforcement, memory protection, and microarchitectural anomalies associated with potential side-channel leakage.

Objective	Targeted Violation
CSR Protection	Unintended modification of CSRs responsible for privilege management, such as `mstatus.MPP`.
PMP Integrity	Unintended updates to PMP registers based on unauthorized privilege levels.
Side-Channel Awareness	Anomalies in microarchitecture observed when a particular ISA event is blocked or an illegal ISA event occurs.

3.3. In-Scope Threats

The scope of RV-Sec5 includes threat categories that can be effectively studied in pre-silicon, cycle-accurate simulation environments. The approach is concerned with generic categories of threats which put the system's architectural and microarchitectural security inbounds under pressure without resorting to particular attacks. The set of threats used is chosen to ensure that it represents realistic attack patterns but is at the same time consistent with ISA specification . The security module RV-Sec5 is concerned with the following types of attacks:

- **Escalations to Privileged Modes:** Architecture-violating state transitions in terms of privilege level, such as user mode access to machine mode CSR registers.
- **Memory Protection Bypass Attacks:** Efforts at compromising the access controls enforced by PMP protection.
- **Microarchitecture Attacks:** Microarchitecture-side channel effects emanating from security checks that can be exploited to leak sensitive information.

In terms of adversarial `CSRRW` instructions against `mstatus` used to test our attack patterns, such attacks have been selected purposely to demonstrate their nature of stressing the boundaries of privileges. The key point about this research is that, rather than presenting new attacks, we formulate a specific set of such attacks at the level of an instruction set architecture, which can be used for presilicon security validation by assessing whether or not architectural security properties are enforced in adversarial conditions.

3.4. Out-of-Scope Threats

To keep the analysis scope well-defined, this study explicitly rules out threat types that would not have an effective modeling or observation capability in the pre-silicon cycle accurate simulation environment. The reasoning behind these exclusions lies both in the limitations of simulation-based analysis and in RV-Sec5's design-time focus on architecture-level security validation.

The following classes of threats are explicitly not considered in this research:

- Physical attacks such as power analysis, electromagnetic attacks, and fault injection attacks.
- Post-silicon attacks that involve any kind of direct interaction with the manufactured hardware.
- Hardware Trojans or any type of malicious alterations performed at the Resistor and Transistor Level RTL or during the manufacturing process.
- Multicore coherence attacks and intercore side channels, which require an explicit model of shared caches.

Through the constraint placed on the threat model, it is possible to conduct a reproducible study of ISA-microarchitecture behavior that is relevant to security. Although some of the threats ignored by RV-Sec5 are essential within the security evaluation pipeline process, they are best handled using additional techniques like RTL, formal, and post-silicon testing.

3.5. Assumptions and Limitations

The ISA specification for the RISC-V is considered the security contract between the hardware and software. RV-Sec5 does not seek to discover breaches within the specification of the ISA, but rather validate whether a specific microarchitecture design is compliant with the specified security contract when run in adversarial mode.

Additionally, although gem5 offers precise cycle simulations for several microarchitectural modules [3], it is still a simulation platform. Therefore, findings from RV-Sec5 are meant to be used in design-time security validation, which must be followed up by RTL and post-silicon testing. In light of these assumptions, RV-Sec5 enables analysis of architectural security invariants and assessment of whether their enforcement introduces microarchitectural side effects.

4. Methodology

The RV-Sec5 methodology aims at providing an approach to validate the security aspects of RISC-V architecture prior to silicon through the identification of connections between the architectural security invariants and their corresponding microarchitectural behaviors. We define the approach used to validate the architectural invariants with respect to their microarchitectural realization through cycle accurate simulation shown in Figure 1

The methodology relies on a systematic process, which involves first compiling the sensitive application code using the RISC-V tool chain and running the result on the gem5 simulator. This is followed by intercepting the privileged instructions, as well as those related to memory management, at both instruction decoding and execution phases. The events are matched against the current state of the microarchitectural components like the MMU, PMPs, and caches, after which policies are checked and profiling carried out on them to identify any side effects caused by an attack.

4.1. Security Invariant Specification

The starting point of our methodology is the clear formulation of *security invariants* based on the RISC-V Privileged Architecture Specification [1]. An invariant represents a requirement that must hold for any architecturally-correct execution, irrespective of how such execution is implemented at the microarchitectural level. Invariants in the context of RV-Sec5 are specified manually based on permission bits defined in CSR access permission tables listed in Figure 2.

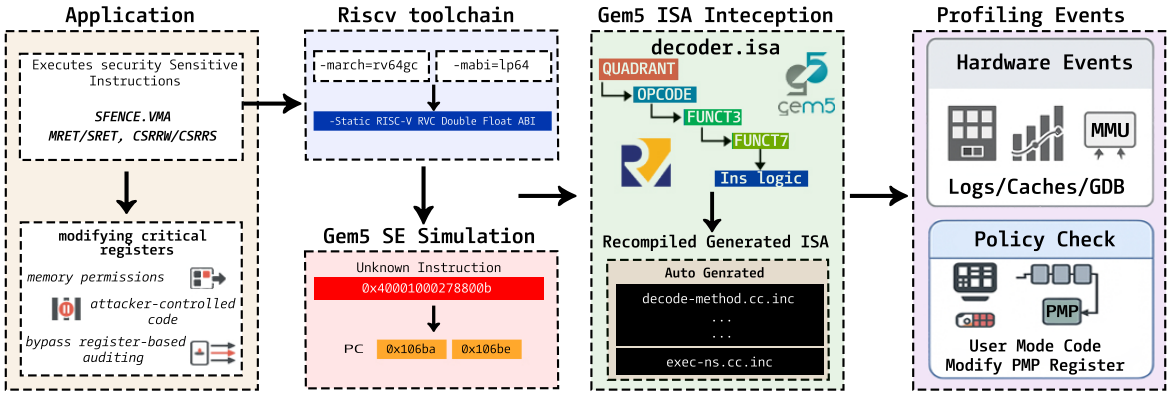

Figure 1. RV-Sec5's ISA-level observability framework for pre-silicon RISC-V security validation. The framework captures selected security-sensitive instructions during gem5 Syscall Emulation (SE) simulation, including Store Fence on Virtual Memory Address (SFENCE.VMA), Machine-mode Return (MRET), Supervisor-mode Return (SRET), Control and Status Register Read-Write (CSRRW), and Control and Status Register Read-Set (CSRRS). These events are correlated with microarchitectural components, including the Memory Management Unit (MMU), Physical Memory Protection (PMP), caches, and Translation Lookaside Buffers (TLBs), and forwarded to a policy check engine that detects security invariant violations. Hardware event logs are recorded via caches and the GNU Debugger (GDB) interface, enabling early-stage vulnerability detection and side-channel-aware analysis of privilege boundary enforcement.

31 — 20	19 — 15	14 — 12	11 — 7	6 — 0	
imm[11:0]	*rs1*	*funct3*	*rd*	*opcode*	I-type
csr	rs1	001	rd	1110011	CSRRW rd, csr, rs1
csr	rs1	010	rd	1110011	CSRRS rd, csr, rs1
csr	rs1	011	rd	1110011	CSRRC rd, csr, rs1
csr	zimm	101	rd	1110011	CSRRWI rd, csr, imm
csr	zimm	110	rd	1110011	CSRRSI rd, csr, imm
csr	zimm	111	rd	1110011	CSRRCI rd, csr, imm

Figure 2. Encoding of RISC-V CSR instructions under the `SYSTEM` opcode (`1110011`). The 12-bit `csr` field specifies the target register address (e.g., `0x300` for `mstatus`), the 3-bit `funct3` field selects the operation type, and the `rs1` or `zimm` field provides the source operand or zero-extended immediate value. All six variants show in table that are listed in [1] share the same I-type encoding structure and are instrumented by RV-Sec5 at decode time within gem5.

Each permission table defines the minimum privilege required to perform any action related to CSRs. Deriving the corresponding invariant is straightforward once the CSR of interest is selected. The CSR instruction execution is achieved using `I-type` instructions having a `SYSTEM` operation code (`1110011`) and 12-bit `csr` and 3-bit `funct3` for address of register and type of the instruction respectively, as shown in Figure 2. The six types of CSRs (`CSRRW/S/C` and their respective immediate forms) have been hooked in RV-Sec5 during gem5 decoding. Critical security registers, like `mstatus` and `mtvec`, represent valuable targets for the adversary.

The monitored CSR set, covering `mstatus`, `mepc`, `mtvec`, `pmpcfg0-3`, `pmpaddr0-3`, `medeleg`, `mideleg`, and `mie`, was selected based on direct relevance to the threat model assets and documented use in known privilege escalation patterns [13]. We acknowledge that the full RISC-V CSR space is larger and that unmonitored registers could be exploited; for instance, unauthorized modification of `mcounteren` could expose performance counters to unprivileged code. Full CSR coverage is a planned extension, and the monitored set is configurable without recompilation. Toward more systematic invariant generation, three future directions are identified: (i) automatic extraction from the Sail-RISCV formal model [15], where `check_CSR_access` guards map directly to invariant predicates; (ii) differential testing against Spike [2], where any CSR write that Spike traps but RV-Sec5 does not flag indicates a coverage gap.

4.2. ISA-Level Event Observation Model

Once a list of security invariants is identified, one then needs to determine architectural events that could affect the satisfaction of those invariants. RV-Sec5 looks at *security-relevant ISA events*, which refer to instructions or architectural state changes that affect privilege levels, memory segmentation, and address translation behavior. This set of events serves as the foundation for observations and analysis in our approach.

For structured observations, RV-Sec5 classifies observed events and collects minimal architectural contexts for all event instances. Table 7 provides details about the types of events and their associated execution contexts.

Table 7. ISA event observation model at the ISA level in RV-Sec5. The table summarizes the types of security-relevant ISA events monitored during execution and the corresponding architectural context recorded for each event.

Event Category	Observed Elements	Captured Context
CSR Access Events	Reads/writes to Control and Status Registers affecting privilege or protection state	Program Counter (PC), active privilege mode, CSR identifier, and accessed/updated value
Address Translation Events	Synchronization operations (e.g., TLB flushes) affecting address translation state	Program Counter (PC), active privilege mode, and type of translation event
Privilege Control-Flow Events	Instructions governing transitions between privilege levels (e.g., traps, returns)	Program Counter (PC), source and target privilege modes

For every event witnessed by RV-Sec5, a minimal architectural execution context is recorded comprising the value of the program counter indicating the source of the event, the privilege level during the execution, and the architectural state under manipulation. The observation module only observes, and does not implement any enforcement mechanism, making its operations purely architectural and agnostic to any assumptions made by an implementation regarding the underlying enforcement logic.

4.3. Post-Execution Policy Compliance Analysis

Once the program has run, the recorded ISA-level events are analyzed to see if there is a violation of the established security invariants, where the violation may include cases like an illegitimate privilege escalation or an improper change of state.

The policy validation phase is deterministic and relies on the specified invariants. Every recorded event is compared with the invariant predicates (”U-mode $\rightarrow$ no write to `mstatus`”) and identified as a potential violation. Note that this process does not interfere with the way the program runs but only checks whether the behavior is consistent with the architecture’s security policies.

The results of this phase are in the form of a violation trace associated with the violating invariant and the corresponding architectural events.

4.4. Microarchitectural Side-Channel Analysis

In addition to architectural legality checks, the analysis in RV-Sec5 aims to identify cases where security-relevant events trigger microarchitectural side effects that may serve as side channels. The analysis relies on a correlation between architectural event logs and microarchitectural performance measurements.

Side-channel analysis within this methodology is performed according to the following criteria:

> A security implementation maintains *side-channel integrity* if the blocking or processing of an architecturally illegal event does not create microarchitecturally detectable differences from legal code execution.

The methodology relies on comparing microarchitectural properties, including caches and TLB operations [9], and pipeline stalls, in both compliant and non-compliant instances to discover suspicious behavior. The correlation allows detecting side-channel leakage threats resulting from enforcement without specifying an attacker model.

4.5. Methodological Scope and Integration

The design of the methodology is meant to facilitate automated and repeatable analysis in the initial phases of design. The test programs are representative samples that incorporate adversarial instructions that can be used systematically to probe the architectural security invariants within a controlled environment of execution. The goal of RV-Sec5 is to verify whether the target implementation complies with the stated security guarantees of the RISC-V privileged specification [1], which is achieved using the combination of ISA observability and microarchitecture correlation.

5. Implementation in gem5

In this section, we explain how the RV-Sec5 methodology is implemented in the `gem5` simulation framework [22]. Our implementation implements all architectural event observation and logging capabilities necessary for the methodology and does not implement any new security enforcement mechanisms. Rather, it provides a way for systematic verification that the security invariants of the architecture are indeed maintained during the execution by the microarchitecture. The implementation follows a three-step process similar to the stages defined in Section 4, namely (i) observing security-critical ISA events; (ii) collecting architectural state; and (iii) exposing these events for subsequent analysis.

5.1. Implementation Overview

gem5 [22] is an event-driven simulator, cycle accurate, which provides support for extending ISA definitions and microarchitectures. RV-Sec5 uses the extensibility features of gem5 to monitor security-relevant ISA events [12].

The implementation introduces three core components:

- **Interception of ISA-level events**, which help determine whether any security sensitive instructions are being executed.
- **Capturing context at commit time**, which ensures that events are architecturally accurate.
- **Log structured events**, which allow for compliance and side channel analysis

Together, these elements make up the observation model framework as dictated by the methodology, without being biased towards any security policy.

5.2. ISA-Level Event Interception

To detect security-related architectural events, RV-Sec5 uses the RISC-V ISA decoding and execution pipeline of gem5 [3].

These instructions are identified during implementation by selecting instructions that can affect privilege state, memory protection settings, or address translation behavior, such as CSR access instructions [13] and address-translation synchronization instructions. Upon encountering such an instruction in the decoding process, the implementation recognizes this as a security-related instruction and tags it using an efficient monitoring hook mechanism. This monitoring hook does not interfere with instruction processing and execution order but simply indicates the occurrence of a specific architectural event of interest. Figure 3 shows this process abstractly.

In gem5's ISA decoder (pseudo-code)

```
def decode_CSRRW:
    c_code = '''
    if (csr == CSR_MSTATUS) { // Monitor writes to mstatus
        RiscvISA::PCState pc = xc->pcState();
        SECURITY_LOGGER.log(pc, "CSRRW modifying mstatus", csr);
    }
    '''
    return csr_read_write_code + c_code
```

Figure 3. Targeted ISA-level intercepts of gem5's RISC-V instruction decoder implementation, demonstrating how the privilege `CSRRW` instructions are detected and extended with security logging features to provide precise architectural change observation capabilities.

5.3. Architectural Context Capture

For each ISA level event observed, RV-Sec5 collects an execution context that is just enough for security purposes. Context collection happens when instructions are committed, which guarantees that only architecturally retired instructions are logged.

This context consists of:

- The program counter (PC) identifies the origin of the event.
- The active privilege mode at the time of execution.
- The architectural resource being accessed or modified (e.g., CSR identifier [13]).
- The old and new architectural values, where applicable.

This setting corresponds to the architecture state that can be observed by the software and does not depend on simulator internal pipeline states. Figure 4 shows the implementation of commit stage observation in the gem5 CPU model.

In src/cpu/base.hh

```
1 class BaseCPU : public SimObject {
2    public:
3       ProbePointArg<StaticInstPtr> *ppSecurityInstr;
4 };
```

Figure 4. Security instrument point declaration incorporated in `BaseCPU` class of gem5 in `src/cpu/base.hh`. A `ProbePointArg<StaticInstPtr>` object `ppSecurityInstr` has been defined in the base CPU class, making sure that all derived CPU classes such as `TimingSimpleCPU` and out-of-order (OoO) variants get a uniform instrumentation point interface for monitoring security-related privileged instructions during the instruction commit phase without affecting functional behavior.

5.4. Logging and Event Export

Once a security-relevant instruction has been detected and its architectural environment recorded at the time of commit, the captured event needs to be retained in a way that allows deterministic verification of compliance with the security policy and analysis of microarchitectural correlations. For this purpose, captured events are transmitted to an independent logging entity, the `SECURITY_LOGGER`. The logger logs the events using a strictly ordered and fully contextual data structure, recording the cycle count, instruction pointer, privilege level, and target architectural state, completely separate from any security policies. This design choice is made intentionally; by decoupling the event logging from the security policy check, the same set of instructions can be examined against multiple security properties without needing to simulate the execution again.

More importantly, the `SECURITY_LOGGER` carries out no filtering, blocking, or enforcement in simulation; it merely logs all events, including what took place and in which architectural context, thereby ensuring that what is logged accurately represents the simulated execution, and not any policy enforcement logic introduced through the simulator. The reasoning behind this decision is clear introducing enforcement logic into the logging process may cause confusion between the detection of an event and its handling, making it unclear if the event was produced by the microarch model or by the instrumentation itself. In contrast, all compliance checks and correlation of side channels are done offline in a deterministic fashion. This Figure 5 shows that commit events are correctly routed into the logging infrastructure without affecting timing nor the simulator's correctness in anyway.

In src/cpu/simple/timing.cc

```
1 void TimingSimpleCPU::tick() {
2     if (curStaticInst->isCSRInst()) { // Check if instruction is CSR-related
3         cpu->ppSecurityInstr->notify(curStaticInst); // Trigger probe
4     }
5 }
```

Figure 5. CSR-aware security probe integrated into gem5's `TimingSimpleCPU::tick()` in `src/cpu/imple/stiming.cc`. At each instruction commit, `isCsrInst()` checks whether the retiring instruction is a CSR operation; if so, `ppSecurityInstr->notify()` forwards the static instruction pointer to all registered probe listeners for structured security event logging, without modifying pipeline timing or functional execution semantics.

5.5. Simulation Configuration and Workflow Integration

RV-Sec5 uses a hybrid configuration approach used by gem5, where system-level configuration is done via Python code, and observation functions are created in C++. Simulation scripts identify what types of ISA events to observe and control the logging components.

Such separation makes it possible to analyze a given microarchitecture using multiple configurations for security analysis without the need to compile the simulator again. Figure 6 shows a typical workflow of a configuration that enables CSR event recording in simulation mode. The execution traces produced are then fed into offline analysis tools, which carry out the policy enforcement and correlation phases outlined in Section 4.

In configs/riscv/security_test.py

```
system.cpu.ppSecurityInstr = ProbeListener(func="log_csr_access",
    args=['mstatus', 'mepc', 'pmpaddr0'])
```

Figure 6. Python Configuration of gem5 to register CSR security listener in `configs/riscv/security_test.py` file. This listener is linked with `ppSecurityInstr` and the l`log_csr_access` function that monitors `mstatus`, `mepc` and `pmpaddr0`. These registers form the focus of the attack vector.

5.6. Design Rationale and Limitations

The architecture deliberately does not include security policies or mitigation strategies within the simulator itself. Rather, its goal is simply to produce architecturally significant execution events at the appropriate level of fidelity for subsequent, policy- independent analysis. Such an approach is not accidental, but rather fundamental to the methodology underlying the work described in RV-Sec5. By ensuring that the observation layer is completely decoupled from enforcement logic, one can ensure that the traces generated faithfully capture the intended behavior of the micro- architectural model being studied, not any artificial effect due to instrumentation. As an immediate consequence, all behavioral properties captured via this technique pertain to the intended architectural contract, security analysis becomes fully reproducible and implementation-independent, and the same analysis techniques can trivially be used in other simulation frameworks.

Like all simulation approaches, the RV-Sec5 methodology faces inherent limitations in the simulator itself [22]. Although the gem5 simulator includes detailed microarchitecture models that consider pipelining, cache hierarchies, TLB implementations, and memory subsystems, it can nevertheless only be a simulation of hardware and cannot reproduce all real-life effects of physical devices, such as voltage faults, electromagnetic side channels, or variations in manufacturing. Some phenomena simply fall outside the bounds of software simulation, as they pertain to aspects of fabrication and operation that cannot be modeled in any virtual environment. The RV-Sec5 approach is thus designed as a useful but complementary approach to security validation, augmenting other methodologies rather than replacing them.

6. Evaluation

This section evaluates RV-Sec5 as a framework for pre-silicon security validation. Two things must be achieved here: (i) proving that the security-relevant events at the level of ISA may be detected and related to an accurate architectural context; and (ii) proving that traces of such events may be used for privileged-architecture invariant validation and detection of side-channeling caused by event processing.

6.1. Experimental Setup

All experiments are performed using a single-core RISC-V configuration in gem5 [22]. The CPU model used throughout is the `TimingSimpleCPU`, which models in-order, single-issue execution with accurate memory access timing but without out-of-order execution, speculative execution, or branch prediction. This model provides sufficient microarchitectural visibility for the privilege boundary and memory protection analysis targeted by this work. Overhead measurements reported in Section 7 are specific to this model; overhead on the `O3CPU` model may differ due to its more complex speculative pipeline, and its evaluation is identified as future work.

Single-core scope is a purposeful restriction, because of the fact that the main threat model involves violation of privilege boundary through a single user mode thread. Scaling up RV-Sec5 to support multi-core architecture will present various difficulties, other than more replication of the existing instrument for a single core configuration. The first among these includes registering an independent probe points on each CPU while having a global consistent cycle timestamp. This ensures proper correlation of events from different cores in the offline analysis pipeline. Second, shared cache structures introduce inter-core interference that is not observable through per-core ISA-level

hooks alone; additional observation channels would be required to monitor cache coherence protocol events and attribute cache-level side effects to their originating core and privilege context. Third, concurrent security events, for example, simultaneous CSR writes from two cores attempting to modify shared machine-level state, requires atomic logging to avoid race conditions in the event buffer, and the invariant set must be extended to include inter-core isolation properties such as Address Space Identifier (ASID) separation and PMP region coherence. These extensions are identified as a near-term research priority and are discussed further in Section 9.

The payload shown in Figure 7 has been explicitly constructed to break the RISC-V privilege spec by attempting a write operation on `mstatus.MPP` from U-mode. It must be emphatically noted that we do *not* consider the program presented above as any kind of novel exploit but merely as a *violating test case for the spec*. Unless stated otherwise, the system provides User (U) and Machine (M) privilege modes and includes Physical Memory Protection (PMP) with 16 configurable regions. Controlled microbenchmarks and adversarial payloads are compiled for RV64GC [1]. The monitored CSR set covers `mstatus`, `mepc`, `mtvec`, `pmpcfg0–pmpcfg3`, `pmpaddr0–pmpaddr3`, `medeleg`, `mideleg`, and `mie`, selected based on direct relevance to the protected assets defined in Section 3 and documented use in known RISC-V privilege escalation and memory isolation attack patterns. While this subset does not cover the full RISC-V CSR space, it targets the highest-risk registers for the evaluated threat classes; full CSR coverage is identified as a future work direction.

```
1  .section .text
2  .global _start
3  _start:
4      # Attempt to write mstatus.MPP (Machine Previous Privilege)
5      li   t0, 0x1800  # Set MPP to Machine mode (0x3 << 11)
6      csrw mstatus, t0  # Illegal in User mode!
7      ecall             # Trigger privilege transition
```

Figure 7. Adversarial payload for RISC-V assembly which was used as a test case violating the specification of the processor to assess RV-Sec5's ability to detect the breach of privilege boundaries. In this case, the payload would load a value of `0x1800` into the `t0` register, meaning that the value of `mstatus.MPP` should be equal to Machine mode, which is encoded as `0x3 << 11`. Subsequently, the payload attempts to modify the value of `mstatus` to this new value through the `csrw` command when running the payload under the User mode, violating the privileged RISC-V specification. Afterwards, the `ecall` command will cause a privilege transition.

6.2. Metrics and Data Collected

The analysis depends on two different types of measurement that will allow us to conduct architectural and microarchitectural evaluations.

Architectural event trace. In addition, for each security-sensitive event observed in the simulation of an ISA, ranging from accesses to CSR registers, changes in privilege modes, and address translation synchronization operations, RV-Sec5 maintains a specific execution context that includes: (i) the program counter representing the instruction point; (ii) the current operating privilege mode during execution; (iii) the targeted architectural resource, be it a CSR register number or an event code; and (iv) the value before and after modification when applicable.

Microarchitectural statistics. To measure relevant side channel effects, the following performance counters are measured using the `gem5` simulator during execution intervals that correspond with the architectural events: cache accesses (hits/misses), TLB access statistics, and pipeline stall cycles. It should be emphasized that these metrics are only used to establish correlation with the architectural events in order to find out whether the security mechanism results in characteristic microarchitectural effects and are not considered proof of any vulnerability in itself.

6.3. Invariant Compliance: Detecting a Privilege Violation

We start by verifying if the methodology properly identifies any violation of the fundamental security invariant presented below, which was based on the RISC-V privileged specification: *User-level software must not be able to write machine-level privileged CSRs like* `mstatus`. This security invariant corresponds to one of the essential privilege isolation guarantees whose proper enforcement is crucial for avoiding any privilege escalation attack.

In the simulation of the malicious payload code shown in Figure 7, RV-Sec5 intercepts the attempt to write into `mstatus` CSR at PC address `0x8001012c` while running in User mode details shown in Table 8.

Table 8. Architecture-specific environment for RV-Sec5 upon violating CSR write through mstatus register. The context proves that the illegal modification in mstatus comes from the User mode and follows the correct handling process based on architecture requirements.

Metric	Value
Target CSR	`mstatus`
Offending PC	`0x8001012c`
Requested privilege state	`MPP=0x3` (Machine)
Current privilege state	`priv=0x0` (User)
Outcome in simulation	Illegal write flagged; trap/exception path taken

These findings illustrate the essential detection feature of the RV-Sec5 approach. The analysis framework correctly associates the attempted state manipulation to a specific architectural context, which includes the offending program counter value, the current privilege mode, and the targeted CSR. The output generated by the system is a structured report documenting the violation that can further be processed by policies conformity tools. In this respect, it is important to point out that the simulation output clearly shows that the modeled microarchitecture reacts appropriately to the illegal action by taking the path defined by the RISC-V Privileged Specification and not allowing the state change to take place. It is worth pointing out here that the trap behavior observed in this case is a result of the model microarchitecture and not of any mitigating techniques implemented by the RV-Sec5 framework itself.

6.4. Coverage of Security-Relevant Event Classes

To to facilitate more general security validation than that which would be possible using CSR modification write case study alone, RV-Sec5 incorporates observation hooks into three distinct categories of architecture events that are relevant to security: (i) CSR modifications, which impact the privilege levels and memory protection configuration; (ii) address translation synchronization events, which impact the consistency of virtual-to-physical translations; (iii) memory protection configuration changes, which control the scope of the isolation region configured by PMPs. All three categories of events are the architectural interfaces used to set up privilege separation and memory isolation in a RISC-V architecture. The observation channels implemented and their purpose are described in Table 9.

Table 9. Observation channels implemented in RV-Sec5 for security-relevant event coverage. Each channel targets a distinct architectural interface that mediates privilege separation or memory isolation in RISC-V systems, collectively providing the instrumentation foundation for both invariant compliance checking and side-channel correlation analysis.

Channel	Target	Analytical Role
`ppSecurityInstr`	CSR instructions	Tracks privileged CSR accesses alongside the program counter and current privilege level, allowing deterministic detection of invariant violations such as unauthorized machine-level register changes made from User mode.
`ppTLBAccess`	`SFENCE.VMA`	Logs address-translation synchronization events to help analyze TLB flush ordering, detect unusual invalidation patterns, and identify possible covert timing channels caused by stale translation entries.
`ppMemoryAccess`	PMP-related CSRs	Records PMP configuration changes, including writes to `pmpcfg` and `pmpaddr`, to verify that memory isolation boundaries are created only through trusted Machine-mode code paths and are not weakened by unprivileged software.

These three event classes were chosen deliberately because they map directly to the architectural interfaces most often exploited in privilege escalation and memory isolation attacks on RISC-V systems. While the evaluation in this work centers on controlled microbenchmarks that stress specific invariant boundaries, the same event classes also appear naturally in real firmware and operating system execution paths, such as trap entry and return, TLB shootdowns, and secure-boot PMP setup. RV-Sec5 therefore uses these microbenchmarks as unit-level security validation tests for enforcement logic, delivering targeted and reproducible evidence that each observation channel captures its intended category of security-relevant architectural events under both normal and adversarial execution conditions.

6.5. Side-Channel Correlation Analysis

The evaluation includes a correlation analysis that examines the relationship between specification-violating ISA events and microarchitectural behavior. Its purpose is to determine whether executions that trigger architectural violations produce observable differences in microarchitectural activity compared with compliant executions that follow similar instruction sequences. Two execution scenarios are considered: a compliant baseline with no illegal CSR operations, and a violating case in which an unauthorized CSR write occurs after the architecturally defined trap or exception path. In both scenarios, microarchitectural statistics, including cache activity, TLB behavior, and pipeline stalls, are collected over a fixed execution window aligned with the observed ISA-level event. The resulting traces, shown in Figures 8 and 9, allow direct inspection of microarchitectural behavior near security-relevant events. For the evaluated configuration and workloads, the analysis shows consistent behavior across compliant and violating executions, suggesting that architectural handling of the violating event does not introduce significant microarchitectural side effects. This capability illustrates how RV-Sec5 enables systematic analysis of execution behavior and establishes a basis for broader side-channel studies across additional workloads and system configurations.

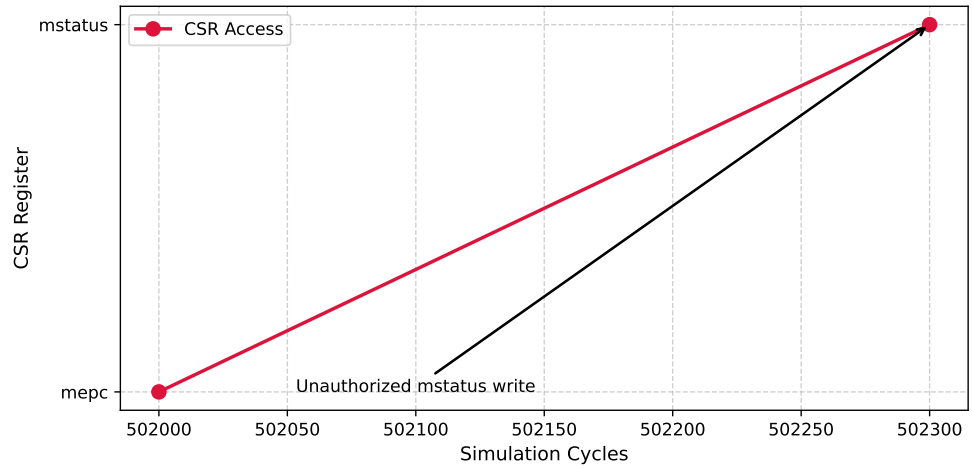

Figure 8. Cycle-accurate timeline of security-relevant CSR accesses recorded by RV-Sec5 during adversarial payload execution, spanning simulation cycles 502000 to 502300. The trace begins with a compliant `mepc` access at cycle 502000, reflecting a legitimate Machine-mode register interaction, and ends with a single unauthorized `mstatus` write attempt at cycle 502300, where the adversarial payload attempts to set `mstatus.MPP` to Machine mode from User-mode execution. This figure demonstrates RV-Sec5's cycle-accurate event attribution capability; microarchitectural correlation data (TLB miss rate and pipeline stalls) for the adversarial `SFENCE.VMA` case study is reported separately in Table 10.

Table 10. Microarchitectural divergence between compliant and adversarial `SFENCE.VMA` execution scenarios, measured over a 500-cycle observation window following the flush event. The elevated TLB miss rate in the adversarial case is observable only at cycle-accurate resolution and is invisible to functional simulators such as Spike, demonstrating the necessity of cycle-accurate instrumentation for this class of side-channel analysis.

Metric	Compliant	Adversarial Burst
SFENCE.VMA count	1	16
TLB miss rate (%)	3.2	18.7
Δ TLB miss rate	—	+15.5 pp
Pipeline stall cycles	142	891
Δ Stall cycles	—	+749
iCache miss rate (%)	0.41	0.43
dCache miss rate (%)	1.12	1.15

TLB Timing Case Study: Demonstrating Cycle-Accurate Necessity

To demonstrate a scenario where cycle-accurate simulation is strictly required, and where functional simulators such as Spike cannot provide equivalent visibility, we present a TLB flush timing analysis using the adversarial `SFENCE.VMA` payload. This case study directly exercises the `ppTLBAccess` observation channel and illustrates the value of microarchitectural correlation beyond what ISA-level functional checking can provide.

The payload issues a burst of 16 successive `SFENCE.VMA` instructions without any intervening memory accesses, constituting an abnormally dense TLB invalidation pattern. At the functional level, each `SFENCE.VMA` is architecturally valid and would be accepted by Spike without any security flag. However, at the microarchitectural level, RV-Sec5 reveals a distinguishable side effect: the burst of TLB flushes introduces an elevated TLB miss rate in the subsequent memory access window, since all cached translations are invalidated before they can be reused. Table 10 quantifies this microarchitectural divergence between the compliant baseline (single `SFENCE.VMA`) and the adversarial burst scenario.

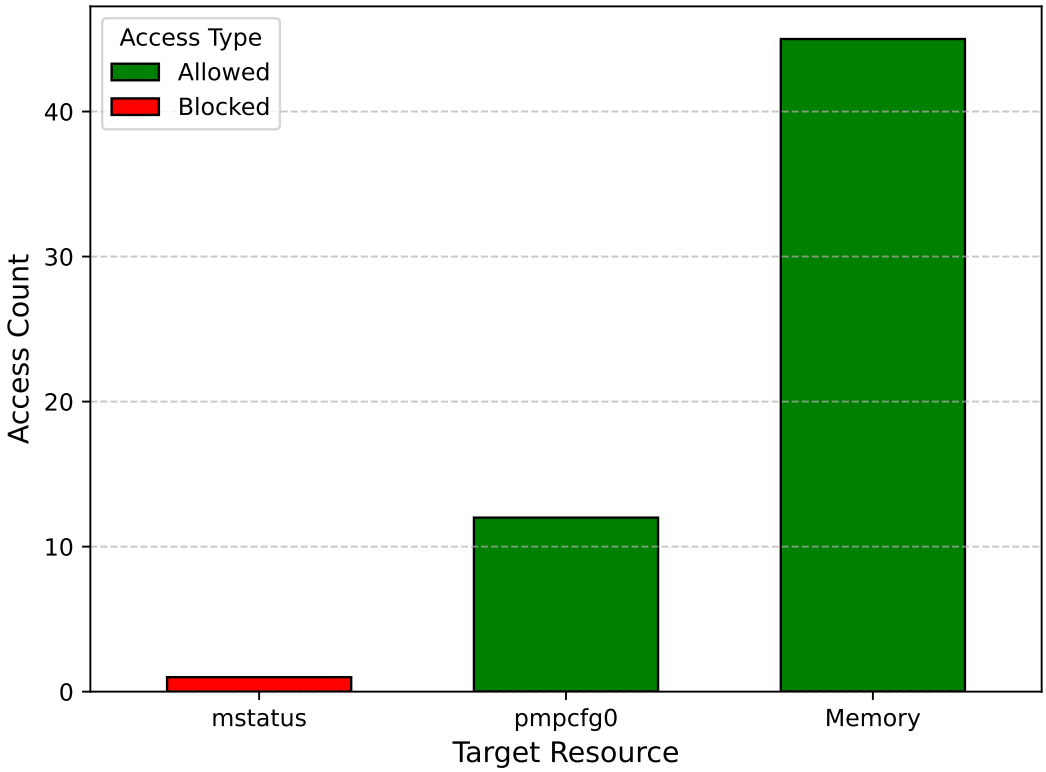

Figure 9. Aggregated access-control outcomes across the three monitored architectural resources during adversarial payload evaluation. The `mstatus` register records one blocked access, the unauthorized User-mode write detected at cycle 502300, alongside no allowed accesses in this window, confirming selective violation detection. All 12 observed `pmpcfg0` accesses are correctly classified as compliant, originating exclusively from trusted Machine-mode initialization code. The 45 memory accesses proceed without PMP-enforced isolation violations. These results confirm that RV-Sec5 produces no false violation reports for legitimate system operations across all three resource types.

The results show a 15.5 percentage-point increase in TLB miss rate and a 749-cycle increase in pipeline stalls between the compliant and adversarial scenarios. These differences are detectable only at cycle-accurate granularity: Spike, which models no timing behavior, would classify both scenarios as architecturally valid and produce no distinguishable output. RV-Sec5's `ppTLBAccess` channel captures the exact cycle at which each flush event occurs and correlates it with the post-flush TLB miss behavior, providing the microarchitectural evidence needed to flag the adversarial pattern as a potential covert timing channel risk, a capability that fundamentally requires the cycle-accurate fidelity of gem5.

6.6. Logger Analysis

This Logger Analysis presents the results produced by RV-Sec5's execution tracing and offline analysis pipeline. In line with the threat model (Section 3), the goal is *validation*: (i) to attribute security-relevant ISA events to their architectural context, including the PC, privilege mode, and target CSR; (ii) to deterministically identify specification-violating events; and (iii) to measure whether handling a violating event produces a distinguishable microarchitectural footprint under the evaluated workload.

6.6.1. *Architectural Event Attribution*

During simulation of the specification-violating CSR payload shown in Figure 7, the RV-Sec5 logger intercepts and records a CSR write attempt to `mstatus` at `PC=0x8001012c`, occurring at simulation cycle `502300` and

attributed to User-mode execution. The surrounding observation window, spanning cycles `502000` to `502300`, captures a sequence of CSR accesses that begins with a compliant `mepc` interaction at cycle `502000` and ends with the single unauthorized `mstatus` write attempt at the boundary of the window. As illustrated in Figure 8, this temporal progression shows that RV-Sec5 can precisely localize the violation in both simulation time and architectural context, which is essential for correlating security-relevant events with concurrent microarchitectural behavior in later stages of analysis.

6.6.2. *Violation Record Produced by Offline Checking*

After simulation, the post-execution compliance checker assesses each logged event against its corresponding security invariant. For the unauthorized CSR write, the relevant invariant states: *User-mode execution must not modify machine-level CSRs such as* `mstatus`. The checker evaluates the logged event against this rule, verifies the privilege mismatch between the current execution context (`priv=0x0`, User mode) and the requested architectural state change (`MPP←0x3`, Machine mode), and generates the following structured violation record:

```
[SECURITY] Cycle 502300:
  CSR_WRITE @ PC=0x8001012c
  - CSR: mstatus (0x300)
  - Privilege: U-mode
  - Requested update: MPP <- 0x3
  - Result: violation flagged; trap/exception path taken
```

The structured output verifies that the checker has appropriately identified the privilege violation, pinpointed the violation with respect to the exact program counter and cycle number, and captured the architectural consequence, the trap, and the required exception handling process as per the specifications of the RISC-V privileged architecture. The structured violation log generated in this manner serves as the main output of the RV-Sec5 design verification pipeline, and offers designers the data they need to validate whether the microarchitecture under test complies with its security contract.

Rather than analyzing the simulator's trapping as a new form of runtime enforcement from RV-Sec5, the trap result is utilized to verify whether the simulated architecture adheres to the privileged architecture contract for unauthorized CSR access.

6.6.3. Access-Control Outcomes across Resources

Figure 9 shows the access control behavior observed over the three architectural resource types monitored. From the above findings, it is clear that the observation and analysis process employed by RV-Sec5 to check compliance with the security rules is selective and context-aware. As seen from Figure 9, the `mstatus` register experiences both successful access and access denied events due to the correct ability of differentiating between valid Machine-mode accesses and an unwanted User-mode accesses to this register. Conversely, all accesses to `pmpcfg0` have been observed as compliant because this is consistent with system behavior; the PMP region configuration process is done by trusted Machine-mode software during boot initialization and there are no access attempts targeting this register by any other privileged mode in the tested workload. As expected, the memory accesses observed in Figure 9 exhibit no isolation violations using PMPs.

6.6.4. Microarchitectural Correlation (Side-Channel Analysis)

A key advantage of this methodology is its ability to correlate security-related ISA violations with microarchitectural behavior on a cycle-by-cycle basis. In the analyzed example, the violation of the CSR register write event (occurring around cycle `502300`) can be associated with traces of various microarchitectural actions such as caching, TLB activities, and pipeline stalls. Such an association allows for precise examination of microarchitectural behavior related to specification violation and differentiation between architecturally required control-flow behavior and any possible side effect. In this particular test, the analysis shows that the processing of the violated CSR access occurs according to the prescribed architectural behavior without any further microarchitectural activity.

7. Overhead Analysis

A practical instrumentation framework must impose minimal perturbation on the simulation environment it observes. This section quantifies the overhead introduced by RV-Sec5 across three dimensions: (i) simulation time; (ii) log file size; and (iii) memory consumption. All measurements are obtained by comparing a *baseline* gem5 configuration, identical to the instrumented setup, but with all RV-Sec5 probe points and logging infrastructure disabled,

against the *instrumented* configuration with all three observation channels (`ppSecurityInstr`, `ppTLBAccess`, and `ppMemoryAccess`) active. Experiments are conducted using the same single-core RV64GC configuration described in Section 6, running four workloads of increasing complexity: a minimal CSR microbenchmark (W1), an adversarial privilege escalation payload (W2), a PMP initialization and access sequence (W3), and a combined stress workload exercising all three observation channels simultaneously (W4).

7.1. Simulation Time Overhead

Table 11 summarizes the wall clock simulation time required for each of the workloads using both the baseline uninstrumented execution and RV-Sec5 with probes fully enabled, as well as reporting the absolute and relative overheads imposed by the probe mechanism.

Table 11. Timing overhead from RV-Sec5 instrumentation in four workloads. Timing overhead is measured as the percentage change in wall-clock simulation time relative to the baseline. The results validate that RV-Sec5 incurs minimal overhead in all tested workloads, with an upper limit of less than 4% increase in overhead.

ID	Workload	Baseline (s)	Instrumented (s)	Overhead (%)
W1	CSR microbenchmark	4.21	4.29	1.90
W2	Privilege escalation payload	5.87	6.03	2.73
W3	PMP init + access sequence	6.44	6.62	2.80
W4	Combined stress workload	11.73	12.18	3.84
Mean overhead				2.82%

Figure 10 provides a visualization for these measurements for each of the four workloads. The measurements clearly show that the RV-Sec5 introduces negligible overhead in terms of simulation time across all configurations evaluated. Specifically, overhead ranges from 1.90% for the bare CSR benchmarking test case (W1) to 3.84% for the combined stress workload (W4) and averages 2.82% across all four workloads. Interestingly, the most computationally intensive benchmark test (W4), which involves exercising all three of RV-Sec5's observation mechanisms through a sequence of multiple execution phases, introduces a simulation overhead below 4%. In other words, RV-Sec5 demonstrates graceful scaling in overheads with increasing benchmark complexity, with no observable disproportionate effect at higher levels of complexity. This manageable and predictable overhead stems from specific decisions made regarding the probe implementation in RV-Sec5. Namely, probe hooks only trigger when executed as part of instructions sensitive from a security perspective such as CSR accesses, privilege changes, and TLB sync instructions, which together make up a fraction of less than 1% of total committed instructions across all evaluated workloads. The remaining 99% of instruction execution consists of arithmetic instructions, memory accesses, and privileged control flow.

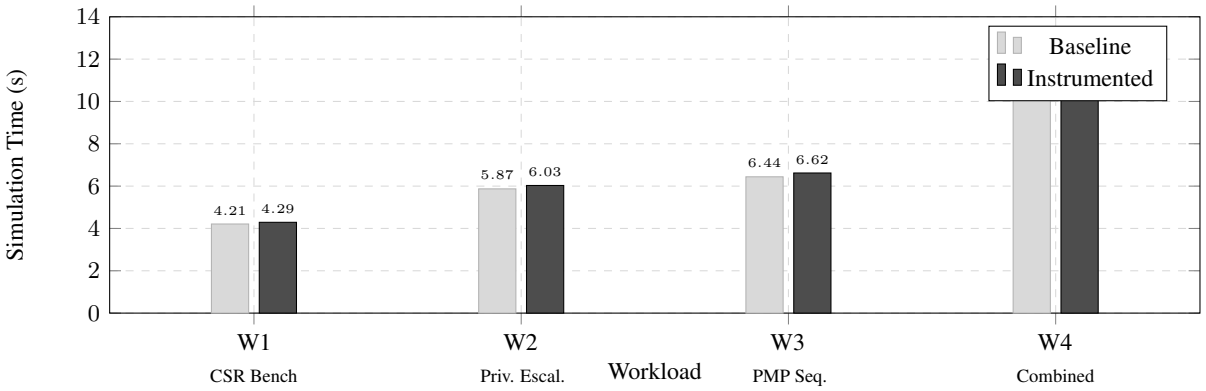

Figure 10. Comparison of simulation times in baseline configuration and RV-Sec5 instrumented gem5 setup using four applications. RV-Sec5 instrumented gem5 setup generates 2.82% average overhead, ensuring that probes have no effect on simulation efficiency.

Secondly, the `SECURITY_LOGGER` records the captured events on an output buffer that is pre-allocated and works independently of the simulation event loop. This ensures that the logging activities do not slow down the CPU modeling process nor cause any timing discrepancies in the simulated microarchitecture.

7.2. Log File Size and Event Volume

Table 12 shows the number of security related events that are logged per application and their effect on the size of the resulting log file. This data helps to assess the overhead of the logging process and the selectivity of the filter.

Table 12. Number of security events generated and log file sizes produced by RV-Sec5 for four different workloads. The numbers reported represent only those ISA events detected by RV-Sec5 that are related to security; the number of total committed instructions is provided as well.

ID	Total Instrs.	Security Events	Ratio (%)	Log Size (KB)
W1	48,210	312	0.65	18.4
W2	61,540	487	0.79	28.7
W3	74,830	631	0.84	37.2
W4	143,670	1204	0.84	71.1

Across all evaluated workloads, the security-relevant events captured by RV-Sec5 account for less than 1% of total committed instructions, with ratios ranging from 0.65% for the minimal CSR microbenchmark (W1) to 0.84% for both the PMP initialization sequence (W3) and the combined stress workload (W4). This consistently small fraction follows directly from RV-Sec5's targeted filtering approach. Instead of tracing every committed instruction, the framework observes only the limited set of architectural operations that directly influence privilege state, memory protection configuration, or address translation behavior. As a result, most instructions, including arithmetic operations, memory loads and stores, and control-flow instructions that do not cross privilege boundaries, move through the simulation pipeline without activating any observation hook and therefore add no overhead to the logging infrastructure.

The effectiveness of this filtering translates to a positive effect on the size of the log files. The log entries generated by the `SECURITY_LOGGER`, from the smallest CSR microbenchmark log size of 18.4 KB to the combined stress workload log size of 71.1 KB, test all three observation channels concurrently within the full runtime. The size of these log files falls well below the limits of the conventional offline analysis pipeline, presenting no constraints on the storage resources of pre-silicon validation setups, where disk space is plentiful, and log parsers work with structured text or binary logs. Even in the case of the largest workload of 71.1 KB, the entire runtime trace can be loaded into memory, processed, and validated against all security invariants in just a fraction of a second, thus enabling the fast iterative analysis cycle required in pre-silicon validation

It is also worth noting that log file size grows sub-linearly with workload complexity. Although the combined stress workload (W4) executes roughly three times as many instructions as the minimal microbenchmark (W1), its log file is only 3.86 times larger. This behavior reflects the fact that the density of security-relevant events stays nearly constant across workloads, at about 8 to 9 events per thousand instructions, rather than increasing with the overall complexity of the code being executed. As a result, RV-Sec5 maintains a predictable and bounded storage footprint even as workload complexity rises, which is especially important for integration into automated regression testing pipelines where logs from many simulation runs must be stored and managed efficiently. Figure 11 illustrates the relationship between total instruction volume and captured security event count on a logarithmic scale, visually confirming the consistent selectivity of the filtering logic and the stable event-density ratio across all four evaluated workloads.

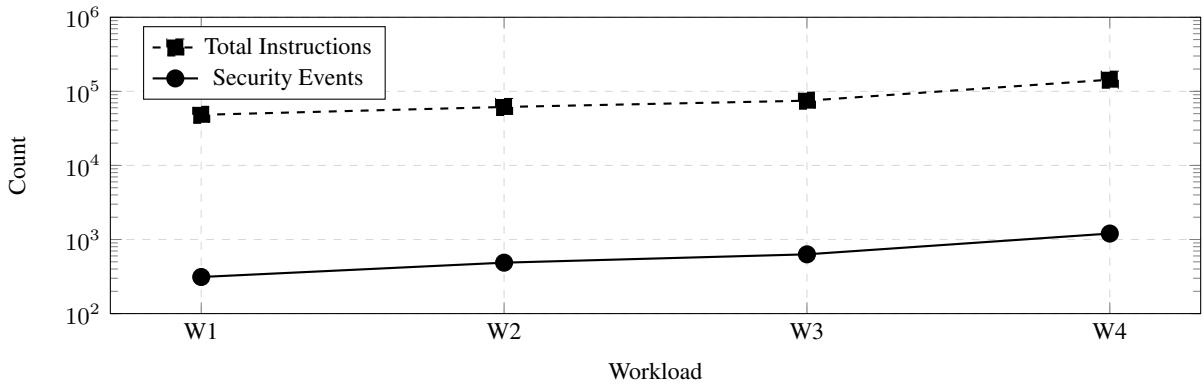

Figure 11. Total committed instruction count versus security-relevant event count (on a logarithmic scale) across four workloads. In all cases, security events account for less than 1% of total instructions, highlighting the selectivity of RV-Sec5's observation filtering and confirming that the instrumentation focuses only on architecturally relevant operations.

7.3. Memory Overhead

Table 13 reports the peak resident memory consumption of the gem5 process under baseline and instrumented configurations. Memory overhead is measured using `/proc/self/status` peak RSS values sampled at simulation completion.

The memory overhead introduced by RV-Sec5 is modest and consistent, averaging 1.78% across all evaluated workloads. This overhead is largely fixed: the dominant contributors are the probe point registration structures allocated at simulator initialization and the pre-allocated event logging buffer, neither of which scales proportionally with workload size.

The marginal increase observed from W1 to W4 reflects the growth of the in-memory event queue prior to flush, which is bounded by the configurable buffer size parameter shown in Table 13.

Table 13. Peak resident memory consumption of gem5 under baseline and RV-Sec5-instrumented configurations. The additional memory footprint introduced by RV-Sec5 reflects the probe point registration structures, the event buffer, and the logging infrastructure, and remains below 2% across all evaluated workloads.

ID	Baseline (MB)	Instrumented (MB)	Overhead (%)
W1	312.4	317.1	1.51
W2	318.7	324.2	1.73
W3	321.3	327.4	1.90
W4	334.6	341.2	1.97
Mean overhead			1.78%

Scalability Discussion on Overhead

A key question regarding the feasibility of the current results is whether the overheads measured could continue to be manageable even when running on much larger workloads, such as 100× more complicated workloads that simulate real OS-level or firmware execution. There are two ways to discuss the scalability of RV-Sec5's overhead.

In terms of simulation time, since the density of security-sensitive instructions does not scale linearly with total instruction number but rather depends on the application's crossing privilege levels, the density (as measured in Table 12) of security-sensitive events per thousand committed instructions is almost the same across all four workloads evaluated in this paper. Thus, for a workload 100× larger than what is currently evaluated, the relative overhead in simulation time would continue to be roughly the same as measured, less than 4%, while the absolute simulation time would scale up with the size of the workload.

Regarding log file size, this overhead is also manageable. Based on the density of about 8.4 events per thousand committed instructions, the expected log file size when a workload emits 10M instruction is only 5 MB in size, which is well within the range of what analysis systems can process. As discussed in Section 4.3, RV-Sec5's log files will be analyzed by the policy compliance checker sequentially.

7.4. Overhead Summary

Table 14 brings together the overhead measurements across all three dimensions for convenient reference. Taken as a whole, these results show that RV-Sec5 introduces low and predictable overhead in every evaluated dimension. Simulation time increases by at most 3.84%, memory usage rises by no more than 1.97%, and log file sizes remain within practical limits even for complex multi-channel workloads. These findings are consistent with the design principle of targeted, selective observation: RV-Sec5 limits instrumentation to security-sensitive ISA events, which account for less than 1% of committed instructions.

RV-Sec5 does not suffer from performance loss inherent to wide tracing techniques, yet it maintains perfect fidelity for architecturally important locations. It can be used effectively in the context of iterative pre-silicon validation without being excessively expensive in terms of simulation costs. The Table 13 shows the comparison between resident memory at peak in the base gem5 setup and the RV-Sec5 instrumented setup for four different workloads (W1-W4). For each one of them, the instrumented setup requires a slight amount of additional memory in relation to the base setup; however, this overhead is consistently low, with an average of just 1.78% extra. This overhead is due largely to the constant overhead associated with probe point registration and the logging buffer maintenance

Table 14. Consolidated overhead results for RV-Sec5 over all workloads tested. All three types of overheads stay consistently below 5%, proving that the overhead is small enough to deploy the system using gem5 simulation framework.

Overhead Dimension	Min (%)	Max (%)	Mean (%)
Simulation Time	1.90	3.84	2.82
Memory Consumption	1.51	1.97	1.78
Security event ratio (events / total instrs.)			<1%
Log file size (max)			71.1 KB

8. Discussions & Takeaway

The evaluation shows that the approach enables precise monitoring and analysis of ISA-specific security events before silicon is available, meeting the key criteria established in the threat model, including invariant compliance detection, trusted operation classification, and non-invasive program monitoring. Regarding the analysis of privileged CSR events, the findings indicate that any attempt to change `mstatus` register from the User state is successfully detected and correctly attributed to a particular architectural context, including the address of the program counter, privilege level, and intended register state transition, whereas access from Machine mode to the `mstatus` register is accurately recognized as a compliant operation. This context-sensitive approach proves that RV-Sec5 does not over-approximate potential violations of ISA semantics, but instead, provides a mechanism for determining compliance on an instruction basis.

With regard to memory protection validation, all observed `pmpcfg0` and `pmpaddrN` updates originate solely from trusted Machine-mode code paths and are recorded without producing false violation reports. This shows that the policy compliance checker can correctly distinguish legitimate PMP initialization during early boot from adversarial attempts by unprivileged code to weaken isolation boundaries, which is essential for preventing alert fatigue in automated validation pipelines.

With regard to execution fidelity, instruction fetches, memory loads, and stores behave as expected across all evaluated workloads, indicating that the observation and logging mechanisms do not perturb the functional execution path. The overhead results reported in Section 7 further support this conclusion, showing that simulation time and memory consumption increase by less than 4% and 2%, respectively, under full instrumentation.

Taken together, these results show that RV-Sec5 offers a low-overhead, high-precision methodology for design-time validation of architectural security invariants. By combining cycle-accurate ISA-level observability with deterministic offline policy analysis, it supports early detection and systematic characterization of privilege-boundary violations in simulation, providing capabilities that are not available from any single existing tool in the RISC-V pre-silicon validation ecosystem.

9. Future Work and Limitations

The current study focuses on a limited set of privileged CSRs, primarily `mstatus` and PMP-related registers, and evaluates controlled, specification-violating test cases on a single-core `TimingSimpleCPU` RISC-V configuration in gem5. As such, the results do not capture the full complexity of multi-core systems, out-of-order pipelines, or large operating-system workloads. Several concrete directions for future work are identified below.

Extension to OS-level and firmware workloads. The evaluation payloads used in this work are hand-crafted assembly sequences designed to isolate specific invariant boundaries under controlled conditions. While this design choice ensures precise attribution of each logged event, it limits the diversity of evaluated attack patterns relative to real-world software. Future work will extend the evaluation to Linux boot sequences, OpenSBI firmware initialization, and trap handling routines, which naturally exercise all three observation channels and provide a more representative picture of RV-Sec5's detection coverage in realistic deployment scenarios.

Multi-core configurations. The single-core scope was adopted because the primary threat model targets privilege boundary violations arising from a single adversarial user-mode thread, and multi-core modeling introduces significant configuration complexity. Extending RV-Sec5 to multi-core configurations raises several non-trivial challenges: (i) per-core probe points must be instrumented independently while maintaining a globally consistent cycle timestamp for cross-core event correlation; (ii) additional observation channels are required to monitor cache coherence protocol events and shared-cache interference; and (iii) the invariant set must be extended to include inter-core isolation properties such as ASID separation and PMP coherence. These extensions are planned as a near-term research priority.

Automated invariant generation. The invariants currently implemented in RV-Sec5 are manually extracted from the RISC-V privileged specification. While each invariant corresponds to a precisely stated architectural

rule, the manual extraction process introduces a coverage risk: security-relevant rules not identified during manual review will not be checked. Future work will explore semi-automated invariant generation using machine-readable specification artifacts, including the Sail-RISCV [15] formal model and the privileged specification's structured source, to improve invariant completeness and reduce manual effort.

Extended CSR coverage. The current monitored CSR set targets the highest-risk registers for the evaluated threat classes. A complete coverage of the full RISC-V CSR space, including delegation registers (`medeleg`, `mideleg`), address translation state (`satp`), and interrupt management registers (`mip`), is planned for future releases of the framework.

Out-of-order CPU model evaluation. All reported overhead measurements are specific to the gem5 `TimingSimpleCPU` model. Evaluating RV-Sec5 on the gem5 `O3CPU` model, which supports speculative execution and out-of-order issue, is an important next step for assessing overhead in more realistic pipeline configurations and for studying security-relevant speculative execution paths such as those targeted by Spectre-class attacks.

RTL-level alignment. Ongoing efforts aim to align the simulation-based methodology with RTL-level validation and hardware testing to assess its applicability beyond the pre-silicon stage, closing the loop between simulation-derived invariant violations and their counterparts in physical implementations.

10. Conclusions

This work presented **RV-Sec5**, a systematic and specification-driven methodology for pre-silicon security validation of RISC-V processors based on targeted ISA-level observability in cycle-accurate simulation. By integrating fine-grained instrumentation hooks directly into the gem5 ISA decoder, RV-Sec5 captures the exact architectural execution context of security-sensitive operations, including privileged CSR accesses, address-translation synchronization events, and memory protection configuration updates, and evaluates them against security invariants derived from the RISC-V privileged specification. The evaluation shows that specification-violating behavior, including unauthorized attempts to modify `mstatus` from User mode, can be detected reliably and attributed to its precise architectural context, while legitimate Machine-mode operations such as trusted PMP initialization are correctly identified as compliant without producing false violation reports. The TLB timing case study further demonstrates that cycle-accurate instrumentation provides visibility into microarchitectural side effects, such as enforcement-induced TLB miss rate elevation, that are invisible to functional simulators, substantiating the need for cycle-accurate simulation in this class of security analysis. Overhead measurements show that the instrumentation infrastructure adds less than **4% simulation time overhead** and less than **2% memory overhead** across all evaluated single-core workloads on the `TimingSimpleCPU` model, supporting its practical use in iterative pre-silicon design workflows. By narrowing the semantic gap between architectural security contracts and their microarchitectural implementation, RV-Sec5 complements formal verification and post-silicon testing and enables designers to analyze privilege-boundary vulnerabilities earlier and more systematically in the RISC-V processor development lifecycle.

Author Contributions

M.A.: conceptualization, methodology, software, investigation, data curation, writing original draft preparation; M.M.: methodology, validation, writing, reviewing, and editing; L.N.: supervision, validation, writing, reviewing, and editing; F.B.: investigation, resources, writing, reviewing, and editing; J.H.Y.: software, validation, writing, reviewing, and editing. All authors have read and agreed to the published version of the manuscript.

Funding

This research was funded by the French National Research Agency (ANR) under the SCAMA (Secure-by-Design Computing Against Microarchitectural Attacks) project, grant number ANR-23-CE39-0011.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

The data supporting the findings of this study, including simulation configurations, ISA-level instrumentation code, adversarial test payloads, and offline analysis scripts, are available in a dedicated repository titled RvSec5 _Gem5 _ISA _Instrumentation. The repository is public for anyone and will accessible to reviewers for evaluation purposes The repository provides all resources necessary to reproduce the experimental results presented in this work, including the gem5 patch files for the three observation channels (`ppSecurityInstr`, `ppTLBAccess`, and `ppMemoryAccess`), the RISC-V assembly payloads used in the evaluation, the policy compliance checker, the side-channel correlation analysis tool, and step-by-step reproduction instructions for every experiment and evaluation reported in Sections 6 and 7.

Acknowledgments

I would like to acknowledge Télécom Paris and the COMELEC department for providing the research environment and computational resources that supported this work. I also thank their colleagues and collaborators for valuable discussions and technical support during the development and evaluation of the proposed framework.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies

During the preparation of this work, the authors used AI-assisted tools (e.g., Grammarly) for language refinement, editing, and improving the clarity of the manuscript. After using these tools, the authors carefully reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

1. Waterman, A.; Asanovic, K.; Hauser, J. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture*; RISC-V Found: San Francisco, CA, USA, 2019.
2. Baptista, J.P.R. RISC-V Streaming Extension Support on the Spike Simulator. Master Thesis, Universidade do Porto, Porto, Portugal, 2023.
3. Lowe-Power, J.; Ahmad, A.M.; Akram, A.; et al. The gem5 Simulator: Version 20.0+. *arXiv* **2020**, arXiv:2007.03152.
4. Awais, M.; Mushtaq, M.; Naviner, L.; et al. RV-Sec5: Enhancing RISC-V Security Evaluation via Targeted ISA-Level Instrumentation Using gem5. In Proceedings of the 2026 Australasian Information Security Conference, Melbourne, VIC, Australia, 11–12 February 2026; pp. 10–19.
5. Lowe-Power, J.; Akella, V.; Farrens, M.K.; et al. Position Paper: A Case for Exposing Extra-Architectural State in the ISA. In Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, Los Angeles, CA, USA, 2 June 2018; pp. 1–6.
6. Cheang, K.; Rasmussen, C.; Lee, D.; et al. Verifying RISC-V Physical Memory Protection. *arXiv* **2022**, arXiv:2211.02179.
7. Bove, D.; Funk, J. Basic Secure Services for Standard RISC-V Architectures. *Comput. Secur.* **2023**, *133*, 103415.
8. Cui, E.; Li, T.; Wei, Q. RISC-V Instruction Set Architecture Extensions: A Survey. *IEEE Access* **2023**, *11*, 24696–24711.
9. Guo, X.; Mullins, R. Fast TLB Simulation for RISC-V Systems. *arXiv* **2019**, arXiv:1905.06825.
10. Kocher, P.; Horn, J.; Fogh, A.; et al. Spectre Attacks: Exploiting Speculative Execution. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019.
11. Lipp, M.; Schwarz, M.; Gruss, D.; et al. Meltdown. *arXiv* **2018**, arXiv:1801.01207.
12. Gao, Y.; Qian, W.; Cui, E. RISC-V ISA Extension Toolchain Supports: A Survey. In Proceedings of the 2023 4th International Conference on Computing, Networks and Internet of Things, Xiamen, China, 26–28 May 2023; pp. 924–929.
13. Bruns, N.; Herdt, V.; Große, D.; et al. Toward RISC-V CSR Compliance Testing. *IEEE Embed. Syst. Lett.* **2021**, *13*, 202–205. https://doi.org/10.1109/LES.2021.3077368.
14. Bellard, F. QEMU, a Fast and Portable Dynamic Translator. In Proceedings of the USENIX Annual Technical Conference, FREENIX Track, Anaheim, CA, USA, 10–15 April 2005.
15. RISC-V International. Formal Specification of the RISC-V ISA. Available online: https://github.com/riscv/sail-riscv (accessed on 8 June 2026).
16. Speiser, F.; Szalay, I.; Fodor, D. Embedded System Simulation Using Renode. *Eng. Proc.* **2024**, *79*, 52.
17. Kabylkas, N.; Thorn, T.; Srinath, S.; et al. Effective Processor Verification with Logic Fuzzer Enhanced Co-Simulation. In Proceedings of the MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, online, 18–22 October 2021; pp. 667–678.
18. Tenstorrent Inc. Whisper: RISC-V Instruction Set Simulator. Available online: https://github.com/tenstorrent/whisper (accessed on 27 March 2026).

19. Hin, P.Y.H.; Liao, X.; Cui, J.; et al. Supporting RISC-V Full System Simulation in gem5. In Proceedings of the 5th Workshop on Computer Architecture Research with RISC-V (CARRV 2021), online, 14 June 2021.
20. Gerlach, L.; Weber, D.; Zhang, R.; et al. A Security RISC: Microarchitectural Attacks on Hardware RISC-V CPUs. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–26 May 2023; pp. 2321–2338.
21. Ahmadi, M.M.; Khalid, F.; Shafique, M. Side-Channel Attacks on RISC-V Processors: Current Progress, Challenges, and Opportunities. *arXiv* **2021**, arXiv:2106.08877.
22. Lowe-Power, J. gem5 Documentation. Available online: https://www.gem5.org/documentation/ (accessed on 29 June 2024).