Call Rewinding Towards RISC-V Specification

Téo Biton[1,2], Olivier Gilles[1], Daniel Gracia Pérez[1], Nikolai Kosmatov[1], Sébastien Pillement[2]

[1] Thales, cortAIx Labs France, F-91767 Palaiseau, France
[2]Nantes Université, CNRS, IETR UMR 6164, F-44000 Nantes, France

Abstract

Memory vulnerabilities are still being exploited today to perform code-reuse attacks by overwriting the return address. Recently, call rewinding has been introduced as a security countermeasure to mitigate this type of abuse, placing itself on the same perimeter as the Zicfiss extension. We propose a thorough comparison of call rewinding with the Zicfiss extension and discuss on its viability for adoption in industrial systems.

Introduction to Call Rewinding

Memory corruption exploits continue to pose a major security threat to modern computing systems, with return-oriented programming (ROP) [1] attacks being one of the most prevalent code-reuse methods to bypass traditional security defenses. Various mitigation techniques, the most popular being shadow stacks, have been proposed to mitigate these threats, commonly refered to as *backward edge* protections. In this context, call rewinding [2] appears as a novel and efficient hardware-based countermeasure aimed at securing return addresses without the overhead commonly associated with software-based solutions.

Call rewinding leverages a fundamental property of how software should be compiled according to the application binary interface (ABI) in major instruction set architectures (ISAs) such as x86, ARM, and RISC-V: all return instructions should transfer control to a valid call site. The core concept of call rewinding involves fetching the instruction preceding the return target address and verifying whether it is a call instruction. If the validation fails, an exception is raised, preventing the execution of a wrongly taken backward control-flow transfer. This approach ensures minimal performance overhead while maintaining robust security against unauthorized control-flow redirection.

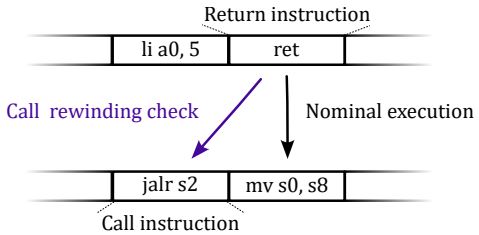

Figure 1: *Rewinded backward control-flow transfer.*

The open source RISC-V ecosystem has played a critical role in enabling and validating call rewinding. The open specifications, combined with freely available open cores and toolchains, provided a flexible and transparent environment for experimentation and implementation. The ability to modify and test CPU microarchitecture without proprietary restrictions accelerated the development and validation of the mechanism. We propose a more in-depth evaluation of call rewinding positionning in relation to the ratified Zicfiss [3] extension. We complement the original paper [2] with an accent on how call rewinding could be made *safer* and *more robust*, in order to be integrated in critical systems.

Positioning with Zicfiss Extension

The RISC-V Zicfiss [3] extension introduces a hardware-assisted shadow stack mechanism designed to enforce backward-edge control-flow integrity. It provides a robust defense against ROP attacks by securely storing return addresses in a separate shadow stack, which is then verified upon function returns. While Zicfiss offers a strong security guarantee, it introduces additional memory and processing overhead, as new instructions are introduced as well as memory pages to store the shadow stacks.

Call rewinding presents an alternative by eliminating the need for dedicated shadow stacks per process, relying instead on the verification of return addresses at the microarchitectural level. Unlike Zicfiss, which requires additional storage and new instructions, call rewinding operates seamlessly within processor architectures and does not require particular states to be saved across contexts. Furthermore, call rewinding leverages return address prediction mechanisms to further optimize performance without sacrificing security.

Performance-wise, the Zicfiss extension introduces

new instructions, and more specifically, instructions that perform memory operations; as the main performance bottleneck in todays systems lies in memory operations, this can introduce execution slowdowns. Shadow stacks require dedicated memory regions and potentially lead to increased cache pressure, while call rewinding is an alternative for scenarios where resource constraints and performance considerations are paramount. It achieves similar levels of backward-edge protection with a negligible cost in most computing environments.

Ultimately, both approaches aim to enhance backward-edge security, but with different tradeoffs. While Zicfiss ensures strict, *fine-grained* return address integrity, call rewinding provides *coarse-grained* return address integrity. In theory, shadow stacks are then more secure, i.e., they leave fewer targets for attackers who tamper with the return address. However, in reality issues can arise (1) when loading dynamic libraries that were not compiled with the extension or (2) regarding the protection of the shadow stack memory. As highlighted above, there are tradeoffs in regards to performance, utilization of hardware resources, and security.

Towards RISC-V Specification

For security mechanisms to be widely adopted in industrial settings, they must provide a balance between effectiveness, efficiency, and ease of integration. Call rewinding aligns well with these requirements, offering a hardware-based solution that requires no modifications to existing binaries and with no footprint on memory. By leveraging return address prediction mechanisms, call rewinding can further optimize performance without sacrificing security.

However, as noted in the original paper [2], call rewinding suffers from false positives in supervisor (S) mode, and the authors have identified one in user (U) mode as well, handled in the exception routine. Since call rewinding cannot be enabled or disabled by software, this can lead to safety issues if unresolved bugs are reported in the kernel or applications. In this respect, the open nature of the RISC-V ecosystem is a strong asset: call rewinding can benefit from the rules of the software environment defined in the Zicfiss specifications, as both are subject to similar constraints. As a concrete example, a first step towards a safe and secure implementation of call rewinding checks could be to enable and disable via writes to a control and status register (CSR) with the granularity of privilege modes.

Another aspect worth noting is the impact of implementing call rewinding on the ISA. Although it does not require any custom instructions, call rewinding changes the behavior of the `ret` pseudo-instruction (which extends to `jalr x0, 0(x1)`). The use of register `x1` to hold the return address is defined in the standard application binary interface (ABI), and the Zicfiss extension is designed to work best with this configuration. Call rewinding, on the other hand, *only* works with this configuration, and any other register used as a link register will not trigger the security checks.

Future work should explore further refinements, such as extending call rewinding to support additional calling conventions and investigating its integration with other control-flow integrity (CFI) techniques. By addressing these issues, call rewinding can play a critical role in enhancing processor security while maintaining the high performance required in modern computing environments.

Perspectives

As the threat landscape continues to evolve, further research will be needed to adapt call rewinding to safety and security requirements of today's systems. It has the potential to be used in standalone mode and provide a really interesting tradeoff between security and performance and resource impact, and in that sense be an alternative to the Zicfiss extension. However, both could work together, especially in the case of dynamically loaded libraries not compiled with the Zicfiss extension, where call rewinding could fill the gap. Collaboration within the open source community will be key to ensuring continuous improvement, fostering innovation, and promoting the specification of call rewinding as a fundamental security feature.

References

[1] Hovav Shacham. "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)". In: *Proceedings of the 14th ACM Conference on Computer and Communications Security*. CCS '07. Alexandria, Virginia, USA: Association for Computing Machinery, 2007, 552–561. ISBN: 9781595937032. DOI: 10.1145/1315245.1315313. URL: https://doi.org/10.1145/1315245.1315313.

[2] Téo Biton et al. "Call Rewinding: Efficient Backward Edge Protection". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.1 (2025), pp. 227–250.

[3] SS-LP-CFI Task Group. *RISC-V Shadow Stacks and Landing Pads, Document Version 1.0.0*. 2024.