Distinguishing Exploit Failure from Effective CHERI Protection on RISC-V

Andreas Hinterdorfer, Manfred Schlägl, Daniel Große

Institute for Complex Systems, Johannes Kepler University Linz, Austria
andreas.hinterdorfer@jku.at, manfred.schlaegl@jku.at, daniel.grosse@jku.at

Abstract

CHERI extends conventional ISAs with hardware-enforced capabilities to provide fine-grained memory protection and its integration in RISC-V is gaining momentum with RVY*. As adoption grows, implementations must be evaluated to ensure working CHERI protection mechanisms. We show that existing memory-corruption exploit implementations do not directly carry over to CHERI-enabled architectures, and that observed exploit failures (i.e., unsuccessful exploits) do not necessarily imply effective protection. To resolve this ambiguity, we propose a methodology that temporarily disables CHERI enforcement within a* RISC-V VP*. Comparing exploit behavior with and without CHERI enforcement under otherwise identical conditions makes it possible to distinguish exploit failure from effective CHERI protection.*

Introduction

Capability Hardware Enhanced RISC Instructions (CHERI) extends conventional *Instruction Set Architectures* (ISAs) by replacing integer pointers with unforgeable capabilities that enforce fine-grained memory protection in hardware. With the RISC-V CHERI extension (RVY) the integration into RISC-V is gaining momentum. While CHERI is already well specified and validated, its growing adoption raises the challenge of verifying that implementations correctly enforce these guarantees. One approach is to port known exploit techniques to the CHERI architecture and observe whether they are prevented.

However, exploits are often fragile as they rely on undefined or architecture-specific behavior and subtle assumptions about memory layout [1]. When ported to CHERI systems, these assumptions may no longer hold, causing exploit failures unrelated to CHERI's protection mechanisms. This makes it difficult to determine whether an unsuccessful exploit reflects effective CHERI protection or simply a broken exploit.

We propose that temporarily disabling CHERI enforcement on a *Virtual Prototype* (VP) provides a controlled reference configuration for distinguishing exploit failure from effective CHERI protection on CHERI architectures [2]. Using this approach, we successfully adapted all applicable attack scenarios of Wilander and Kamkar [3] into viable exploits on vulnerable CHERI-enabled RISC-V architectures.

Analysis of Exploit Assumptions

To illustrate the challenges of porting existing exploits to CHERI architectures, we analyze one in detail. The exploit given in Listing 1 relies on a buffer overflow to (A) overwrite a saved return address on the stack, thereby (B) redirecting control flow to attacker-controlled code, as illustrated in Figure 1. Although the attack buffer is constructed in this function, it reflects a common memory corruption pattern, where attacker-controlled input overflows a buffer.

```
1  void vulnerable_function(uintptr_t dummy){
2    long buffer[16];
3    uint8_t offset = 12;
4    long overflow_len = dummy - (uintptr_t)&buffer - offset;
5    memset(overflow_buffer, 'A', overflow_len);
6    overflow_buffer[overflow_len/long_size - 1] = (long)&shellcode;
7    memcpy(buffer, overflow_buffer, overflow_len);
8  }
9  void calling_function(){
10   unsigned dummy = 0;
11   vulnerable_function((uintptr_t)&dummy);
12 }
```

Listing 1: *Code of the described exploit.*

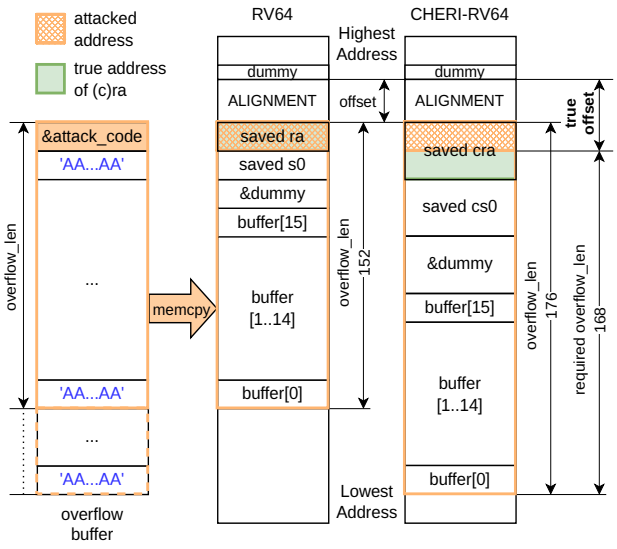

Figure 1: *Stack layout (stack growing downwards) of buffer-overflow exploit showing: (a) constructed exploit buffer, (b) exploit on* RV64*, (c) exploit on CHERI-*RV64*, (d) fix for CHERI-*RV64*.*

While the exploit successfully redirects control flow on RV64, it fails if CHERI is enabled due to changes in stack layout and memory representation. Because pointers are replaced by 128-bit capabilities, the saved

return address `ra` becomes `cra`, shifting the actual address down by 8 bytes to accommodate for the added metadata. As illustrated in Figure 1(c), the exploit's intended target (orange) remains unchanged, while the return address (green) moves downward. Consequently, the exploit overwrites the metadata with `&attack_code`, while the address field is filled with `'AAAAAAAA'`. Therefore, even if CHERI protections were disabled or faulty, the unmodified exploit would not redirect control flow to the attacker's intended code, but instead to a (most likely) invalid address, causing a trap or other unintended behavior.

VP-based Exploit Evaluation

We use *RISC-V VP++* [4], recently extended with CHERI support [5], providing a platform for both conventional and CHERI-enabled architectures, with the option to disable CHERI enforcement.

Although this VP can run full operating systems, we focus on bare-metal execution to minimize external effects and ensure comparable conditions.

When executed on conventional `RV64`, the analyzed exploit successfully redirects control flow to the attacker's code, as shown in Figure 1(b). However, when built and executed with CHERI enabled, the buffer overflow caused by the `memcpy` instruction in Line 7 of Listing 1 is prevented by CHERI, causing a trap and terminating execution. This termination hides that the exploit is not valid on the CHERI-enabled architecture, as established by the previous analysis.

By disabling CHERI enforcement on the VP, we can observe the exploit's behavior beyond the point where CHERI would normally intervene, while still logging violations to provide insights into the involved protection mechanisms. In this configuration, the exploit is allowed to write over the buffer's bounds, as the occurring *LengthViolation* is ignored. Upon returning from the vulnerable function, the corrupted capability is dereferenced, causing a *TagViolation* as the tag bit is cleared by hardware when parts of the capability are overwritten. This demonstrates that multiple CHERI protection mechanisms independently prevent this exploit. Because the *TagViolation* is also ignored, execution jumps to an invalid address, causing the VP to trap and terminate when the next instruction is fetched. Without detailed logging or trap analysis, this behavior could be misinterpreted as effective CHERI protection, even though the exploit failure is not caused by CHERI's protection mechanisms. The exploit can be fixed by adjusting the offset to account for the changed layout as shown in Figure 1(d), which allows successful control flow redirection when CHERI enforcement is disabled.

Using the VP to execute code without CHERI enforcement allows the exploit to be evaluated in the absence of CHERI protections. By comparing behavior with and without CHERI enforcement under otherwise identical conditions, an exploit failure can be clearly distinguished from effective CHERI protection. While the validity analysis can be performed manually, the VP allows automating this process by simply checking whether the exploit successfully redirects control flow when CHERI enforcement is disabled.

Applying this methodology to a collection of attack scenarios introduced by Wilander and Kamkar [3], which were later ported to RISC-V by Palmiero et al. [6], allowed to successfully adapt all 9 applicable scenarios into viable exploits on a CHERI-enabled RISC-V architecture. Meaning that these exploits lead to successful control flow redirection on vulnerable implementations, while being prevented by correct CHERI implementations.

Conclusion

By showing that existing memory corruption exploits do not trivially transfer to new architectures, we emphasize the need to distinguish between exploit failures caused by CHERI and those due to invalid exploit assumptions. We introduce a VP-based methodology that temporarily disables CHERI enforcement, enabling direct comparison of exploit behavior with and without protections and clearly separating exploit failure from effective protection by CHERI.

Using this VP-based methodology, we successfully adapted all applicable attack scenarios of Wilander and Kamkar [3] into viable exploits on a CHERI-enabled RISC-V architecture. Moreover, the method allows for easy integration into the development and testing workflow, while providing deeper insights into exploit behavior, offering a powerful tool for (automated) test validation. All our contributions, including the adapted attack scenarios, the testing environment, and all results, are released as open source.

References

[1] M. Gallagher et al. "Morpheus: A Vulnerability-Tolerant Secure Architecture Based on Ensembles of Moving Target Defenses with Churn". In: ASPLOS '19. 2019, pp. 469–484.

[2] T. De Schutter. *Better Software. Faster!: Best Practices in Virtual Prototyping*. Synopsys Press, Mar. 2014.

[3] J. Wilander and M. Kamkar. "A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention." In: *NDSS*. Vol. 3. 2003, pp. 149–162.

[4] M. Schlägl, C. Hazott, and D. Große. "RISC-V VP++: Next Generation Open-Source Virtual Prototype". In: *Workshop on Open-Source Design Automation*. 2024.

[5] M. Schlägl, A. Hinterdorfer, and D. Große. "A RISC-V CHERI VP: Enabling System-Level Evaluation of the Capability-Based CHERI Architecture". In: *ASP-DAC*. 2026.

[6] C. Palmiero et al. "Design and implementation of a dynamic information flow tracking architecture to secure a RISC-V core for IoT applications". In: *IEEE HPEC*. 2018, pp. 1–7.