Safe Speculation for CHERI

Franz A. Fuchs*, Jonathan Woodruff*, Peter Rugg*, Alexandre Joannou*,
Jessica Clarke*, John Baldwin‡, Brooks Davis†,
Peter G. Neumann†, Robert N. M. Watson*, Simon W. Moore*

*Department of Computer Science and Technology, University of Cambridge, Cambridge, UK
†SRI International, Menlo Park, CA, USA ‡Ararat River Consulting, Ashland, VA, USA
{franz.fuchs, jonathan.woodruff, peter.rugg, alexandre.joannou}@cl.cam.ac.uk
{jessica.clarke, robert.watson, simon.moore}@cl.cam.ac.uk
{neumann, brooks}@csl.sri.com john@araratriver.com

***Abstract*—We present an architectural Capability Speculation Contract (CSC) for CHERI implementations, test for violations in the CHERI-Toooba microarchitecture, and develop and evaluate a conforming implementation. The CHERI capability instruction-set extension promises proven architectural guarantees for memory safety and pointer provenance. However, superscalar and out-of-order CHERI implementations will need to contend with microarchitectural transient-execution side-channel attacks. To ensure the safety of all CHERI implementations, we articulate CSC: a universal architectural speculation contract for the CHERI architecture that maintains key capability invariants in speculation. We then develop tests against sub-classes of CSC, and discover violations in CHERI-Toooba that lead to a new class of transient-execution attacks, Meltdown-CF (Capability Forgery) for which we develop a user-mode exploit that allows reads of secret data. We then develop strategies to fully enforce CSC in CHERI-Toooba. We find that simplistic, strong enforcement incurs a low performance overhead of only 3.43% in SPECint2006 benchmarks, with promise for more optimal designs in the future. Our architectural recommendations to mitigate Meltdown-CF have been accepted by the upstream CHERI architecture and are included in current CHERI-RISC-V drafts for ratification.**

***Index Terms*—transient-execution attacks, instruction-set architectures, testing, guarantees, microarchitecture, CHERI**

I. Introduction

CHERI (Capability Hardware Enhanced RISC Instructions) extends instruction sets with unforgeable, bounded pointers to augment ring and page-table memory protection [1]. CHERI instruction sets constrain each memory access to the intended object, and have been formally proven to compartmentalize a program within the set of pointers it possesses [2]. The promise of CHERI protection has inspired experimental Arm and RISC-V extensions. Arm's Morello is a CHERI research prototype SoC that with a 4-way superscalar, out-of-order pipeline enables research and evaluation [3]. CHERI-RISC-V is in the process of being ratified, and has been announced in a commercial product [4].

Since 2017, researchers have repeatedly demonstrated transient-execution attacks bypassing traditional architectural security mechanisms such as privilege rings and address space separation [5]–[13], and also specialized security architectures such as pointer authentication [14] and enclaves [15]. It is abundantly clear that superscalar, out-of-order CHERI implementations will need to consider how to maintain CHERI guarantees in the face of transient-execution attacks.

Prior work on Hardware-software contracts [16] and Architectural Speculation Contracts [17] have laid the groundwork for specifying how hardware should be constrained to allow safe reasoning about speculation without unduly limiting performance. In this paper, we define the Capability Speculation Contract (CSC) to provide similar protections for CHERI implementations.

In this paper, we investigate how the contract can be tested and enforced using the CHERI-Toooba implementation. This process demonstrates the utility of an architecturally defined contract to constrain the limits of speculative execution; the CSC specification with the test generator was sufficient to expose serious transient-execution violations and support the development of a safe implementation.

We make the following contributions:

- We define the Capability Speculation Contract (CSC) for CHERI capability enforcement in speculation, while allowing high-performance implementations.
- We develop a suite of test generators in the TestRIG framework [18] to test critical aspects of CSC, and discover violations in CHERI-Toooba.
- We describe Meltdown-CF, a new family of transient-execution vulnerabilities for CHERI implementations.
- We develop a working Meltdown-CF exploit in a user process in CheriBSD, breaking CHERI's spatial memory safety guarantees in CHERI-Toooba.
- We design a mitigation strategy for Meltdown-CF that includes adjusting the CHERI-RISC-V architecture to clear tags on illegal capability manipulations rather than throw exceptions. This change is included in drafts for ratification.
- We implement this protection in CHERI-Toooba.

This project was supported by the NCSC under the UK RISE Initiative. This work was supported in part by the Engineering and Physical Sciences Research Council EP/S030867/1. The authors gratefully acknowledge support from Arm Limited. This work was sponsored by the Air Force Research Laboratory (AFRL) and by the Defense Advanced Research Projects Agency (DARPA) under Contracts No. FA8750-24-C-B047 ("DEC"), HR0011-18-C-0016 ("ECATS"), and HR0011-22-C-0110 ("ETC"). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Air Force Research Laboratory (AFRL) or the Defense Advanced Research Projects Agency (DARPA). Distribution Statement A: Approved for public release: distribution is unlimited. For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

- We further implement *SinglePCC*, enforcing safe bounds for the program counter. This provides complete enforcement of CSC in CHERI-Toooba.
- We compare both safety properties and performance cost of the modified CHERI-Toooba implementation against Arm's Morello prototype microarchitecture. Despite being a simplistic implementation that reduces logic complexity, our modification incurs only 3.43% cycle overhead for full CSC enforcement compared with Morello's 10.7% cycle overhead for (unintentional) partial CSC enforcement.

II. Background

A. CHERI

CHERI has been recognized in a recent White House report "Back to Building Blocks: A Path Toward Secure and Measurable Software" [19], which highlights memory safety as a key tenet to improve security. CHERI executables rely on hardware-supported spatial safety for stack, heap, and global objects, with heap temporal safety also built on top of the capability guarantees [20].

CHERI capabilities double pointer size; for a 64-bit address, CHERI adds 64 bits of compressed bounds and permission metadata as shown in Figure 1. Capabilities not only *describe* a region of address space, but also *authorize* access according to the permissions field. Each 128-bit word has a hidden validity tag (a 129th bit) to distinguish a valid capability from data; these tag bits are stored atomically with capabilities, both in registers and in memory. CHERI supports *sealed* capabilities, which are immutable and are used for software compartmentalization among other use cases.

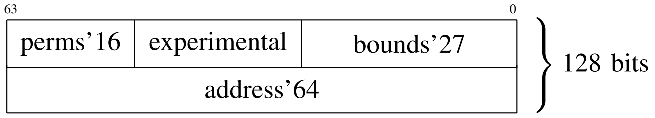

Fig. 1: Bit representation of a 128-bit capability.

Memory cannot be addressed without a valid capability. Furthermore, capability register-to-register operations never increase the privilege of a capability, and would-be violations must either clear the validity tag or throw an exception. These two properties imply that execution is entirely constrained to the memory addresses authorized by the capabilities in the register file and to those transitively reachable through them.

CHERI defines general-purpose capability registers used for data access, and also the *Program Counter Capability* (PCC) used for instruction access. A *pure-capability* program represents all pointers using capabilities; each access is constrained to its intended object, providing intra-process protection. Programs can also be constrained to a subset of the address space by limiting the capabilities provided to them. This natural compartmentalization offers promise for isolating untrusted components in both the kernel (e.g., drivers or Berkeley Packet Filter programs) or userspace programs (e.g., JavaScript or application plugins). The promise of CHERI protection has inspired experimental MIPS, RISC-V, and Arm extensions, with the latter being developed into an ASIC in the Arm Morello research project [3]. While there have been formal proofs of the CHERI protection mechanism [2], these proofs are confined to the architectural level and explicitly do not consider information leaks via timing channels from transient execution or otherwise.

B. Hardware-Software Contracts

In Hardware-Software Contracts for Secure Speculation, Guarnieri et al. present a framework for specifying contracts that guarantee memory safety properties for programs running on a simple speculation-aware model of a processor [16]. *Revizor* [21] tested Hardware-Software contracts using an expanded speculation model on sophisticated out-of-order x86 processors which were able to detect a set of transient-execution vulnerabilities. One of the greatest challenges of this work was to sanitize the test bench such that execution approximated their simplified model of execution, exposing that the model used in their work was missing many of the possibilities of an actual implementation. This work produced a promising demonstration that the safety of programs can be proven with respect to a model of speculative execution. However, we believe that the full potential of this approach requires a model of speculative execution that is collaboratively developed with processor implementors and enshrined in the architecture.

C. Architectural Contracts for Safe Speculation

Fuchs et al. present a framework for contracts for safe speculation on the instruction-set architecture level [17]. They research the feasibility of testing and enforcing contracts for the most common forms of speculation in general-purpose ISAs. These Architectural Contracts architecturally formalize the computation models required to reason about Hardware-Software Contracts so that implementations can target a concrete set of guarantees. These contracts are carefully designed to align with the common case of current high-performance microarchitectures; performance is not generally impacted, although certain corner cases are disallowed in speculation if their security implications are difficult to reason about.

This style of contract need not reason about side-channels, which are perhaps impossible to fully enumerate. Instead, it prevents private information from entering the core where computation might leave traces for a side-channel attack.

III. Threat Model

Conventional architectures usually define their security model over two primitives: privilege rings and address spaces. CHERI additionally defines in-memory capabilities equipped with permissions and bounds information.

The microarchitecture of a processor should enforce the memory-protection mechanisms of its ISA specification. Threat models for such processors assume an attacker who can execute arbitrary bit patterns, but is nonetheless expected to be entirely constrained to reading and writing memory mapped in its address space and accessible in its ring. While side-channels were previously excluded from the de-facto mainstream threat model, Spectre and Meltdown combined side-channel attacks and *transient-execution* [5], [6] to achieve arbitrary read access. These prompted a large-scale response from both software and hardware, demonstrating clearly that the threat model for processor vendors has shifted to include transient-execution attacks [22].

Throughout this paper, we therefore assume a threat model where an attacker has arbitrary code execution and can observe timing information, e.g., through the cache hierarchy, and other side channels to exfiltrate information. The attacker and the victim are architecturally isolated using CHERI capabilities, preventing explicit reads and writes of code or data. We assume that the victim is not collaborating with the attacker, as in the case of a covert channel. Given these restrictions, we do not expect any secret of the victim to be observable to the attacker through transient execution; no visibility of failed speculation in the attacker can reveal a secret from the victim.

We recommend addressing this threat model by adding contracts to the instruction-set architecture that constrain capability speculation (in the style of Architectural Speculation Contracts [17]) to allow reasoning about CHERI program safety on all implementations. Such capability architectural contracts both enable microarchitectural threat models and support enforcement of software threat models. For microarchitectures, the speculation contract embodies the transient-execution threat model, enabling testing and even formal verification against clear properties. For software, the speculation contract provides basic building blocks that allow reasoning about software security against transient-execution attacks with respect to their higher-level threat model.

Definitions

We will use the following terminology in our contracts:

a) **Committed Register State:** Our contract for capability speculation argues about the *committed register state* in a processor. While determining the committed register state in an in-order processor is easy, this is not as straightforward in an out-of-order processor. We define a value part of the *committed register state* if the instruction producing this value is *non-squashable*. A non-squashable instruction is guaranteed to commit, but might not be at the Commit pipeline stage yet. This definition allows us to construct a capability contract with security guarantees, but equally permits for as much microarchitectural design freedom as possible.

b) **Issued Memory Accesses:** An *issued* memory access is allowed to return data at any level. For example, memory accesses might find data in the load queue, the store buffer, or the L1 data cache. If the address selects data on any of these levels, we consider the memory access *issued*. For example, in an out-of-order processor, when a cached memory address is delivered to the load queue, the operation is allowed to proceed to index various structures, including the store queue, the store buffer, and the L1 data cache.

IV. A Capability Speculation Contract (CSC)

To ensure safe, high-performance CHERI implementations, we must define an architectural contract for safe speculation with CHERI capabilities. In CHERI, bounds and permissions checks must not only be safe in non-faulting, in-order execution, but also in transient, faulting execution [23], [24]. CHERI invariants include:

- CHERI capabilities are unforgeable; capabilities are derived only from capabilities of greater or equal privilege.
- Memory can be addressed only through a capability describing and authorizing access to that address.

The first requirement is naturally enforceable in speculation, as pipelines generally forward values that are legitimately calculated from register state. The second requirement is also naturally enforceable, as capability metadata is bundled with the address and can be verified before issuing requests to memory. These two requirements together give rise to a powerful emergent property we call the *Capability Speculation Contract* (CSC):

> *Capability Speculation Contract* (CSC)
>
> All instruction and data-memory accesses issued in speculation must be authorized by capabilities either:
>
> 1) in the committed register file;
> 2) in memory transitively reachable through 1.

In other words, a CHERI processor should act – even in speculation – only with rights transitively reachable from its architecturally committed register file. CSC does *not* forbid speculation on capabilities, but it does forbid using speculatively manipulated capabilities that cannot be found in the architectural register file and its transitive closure. As with previous speculation contracts [17], CSC obviates side-channel concerns by preventing memory accesses to illegal addresses from being issued. This approach prevents illegal data from entering the core before it might be exfiltrated by a side-channel.

CHERI capabilities separately authorize data access and instruction fetch, so we may distinguish between *data-CSC* and *instruction-CSC*.

Data-CSC requires aggressive enforcement of both memory bounds and capability provenance (i.e., the valid derivation of capabilities). Checking bounds before data-memory access is reasonable and can be done in parallel to memory translation, which must also succeed before the access is issued [17]. Capability provenance is also reasonable to enforce, as data values are generally forwarded results of a valid data flow from the committed register state.

Instruction-CSC enforcement is challenging, as instruction addresses are generally predicted with no dependency on committed register state. Nevertheless, instruction-CSC is highly desirable, as it leverages the PC bounds metadata provided by CHERI executables to constrain execution to the current compartment.

Instruction-CSC and data-CSC are closely related, and a violation of one can lead to a subsequent violation of the other. For example, code capabilities usually also allow loading data with an example demonstrated in Section VII-B. Therefore, enforcement of both forms of CSC is highly desirable in high-performance CHERI implementations.

V. Security Evaluation

We must evaluate our contracts ability to enforce CHERI's security guarantees in the face of transient execution. Published CHERI guarantees comprise: *bounds*, *permissions*, *encapsulation*, *provenance validity*, *monotonicity*, and *integrity* [1].

The Capability Speculation Contract (CSC) does not argue directly about speculative capability rights, but rather about speculative memory accesses. CSC states that memory accesses must be authorized by architectural capabilities (i.e., capabilities in the committed register state), but does not otherwise limit the existence of illegal capabilities in speculation. While this freedom is likely to be appreciated by microarchitects who are optimizing for performance, we must be certain that CSC is sufficient. Consider the following example:

```
cincoffset ca0, ca1, a0
clb a0, 0(ca0)
```

This CHERI-RISC-V assembly code loads a byte from an array at an offset (i.e., `arr[i]` in C). Following our contract, `cincoffset ca0, ca1, a0` is permitted to speculatively produce an illegal capability, e.g., using sophisticated value prediction [9]. This speculation, however, must be resolved before `clb a0, 0(ca0)` (a load byte instruction through a CHERI capability) accesses memory in order to avoid violating CSC by issuing an illegal memory access. This level of constraint must be sufficient to enforce all expected CHERI security properties.

CHERI *Bounds* and *permissions* primarily relate to memory, and direct violations of these cases (e.g., an out-of-bounds load) are trivially excluded by CSC. The remaining category of permissions and bounds control types and sealing. It could be possible to transiently manipulate permissions to allow unsealing without causing an illegal memory access. However, the only advantage of an unsealed capability is to access memory, which is prevented by CSC. Therefore, even an illegally unsealed capability cannot expose new data to the core unless it can by used to access memory.

Provenance validity ensures that valid capabilities are always derived from other valid capabilities; *integrity* ensures that all such derivations are valid, in particular enforcing *monotonicity* such that derived capabilities never have greater privilege than their forbears. Similarly to above, if no memory access is allowed through fabricated or expanded capabilities, then no advantage can be gained from transiently possessing provenance-breaking or non-monotonic capabilities.

Encapsulation is the mechanism CHERI provides for software compartmentalization, and relies on sealing permissions mentioned above. Changing compartments enables both executing instructions and accessing data from the new compartment, both of which require memory access. Thus, an attacker cannot gain any advantage by unsealing code and data transiently without violating CSC.

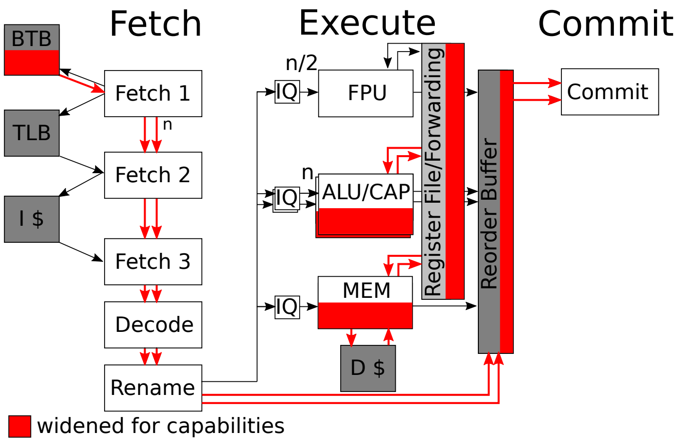

Fig. 2: Pipeline diagram of CHERI-Toooba with n=2.

Thus, the memory-centric specification of CSC is sufficient to uphold CHERI security guarantees while allowing transient register-to-register violations. Incidentally, however, the path we took for microarchitectural enforcement of data-CSC was to guard all capability transformations as discussed in Section IX. This is not likely to be true for all microarchitectures, as extra performance can be gained from optimistically forwarding a result before all checks at the cost of the complexity of validating checks before issuing the memory request.

VI. CHERI-RISC-V Evaluation Platform

In this work, we use CHERI-Toooba [25] for our experiments and evaluation. Toooba is a branch of RiscyOO – a parameterizable superscalar out-of-order RISC-V core written in Bluespec SystemVerilog [26] – that has added compressed instructions, a debug unit, and prefetching. CHERI-Toooba adds support for the CHERI-RISC-V instruction-set extension [1]. All general-purpose registers and datapaths have been extended by Rugg et al. to support full 128-bit CHERI capabilities [27]. We configure CHERI-Toooba with a commit-width of two instructions per cycle with an out-of-order window of 64 instructions. In this configuration, CHERI-Toooba employs two integer pipelines, one floating-point pipeline, and one memory pipeline. The L1 instruction and data cache are each 32KiB and 8-way associative. The L2 last-level cache has a capacity of 1MiB and is 16-way associative. The CHERI-Toooba pipeline diagram is depicted in Figure 2. The CHERI-RISC-V project currently supports two FGPA platforms for research: the VCU118 board as well as the DE10Pro board.

VII. Hardware Testing

For any speculation contract to be useful, it must be verifiable during hardware development. We evaluate the testability of CSC using generators in the TestRIG framework [18], which injects instruction sequences into CHERI-Toooba and collects traces of the resulting execution, allowing assertions on its behavior. The assertions used in this testing approach are asserting that architecturally visible state does not change during the execution of a test. If the architecturally visible state changes, a violation of a constraint has been found.

A. Data-CSC Testing

To test data-CSC, we developed a single TestRIG generator that was able to produce examples of all known data-CSC violations in the CHERI-Toooba core. This generator arranges for the data-cache miss counter to indicate accesses not authorized by capabilities in the committed register file. Each sequence starts with a full reset, which clears all caches. A prelude then prepares a capability granting access to a single word of memory, and loads that word. Henceforth, any cache misses will indicate a memory access not allowed by this capability. A random stream of capability instructions is then fed to the processor, followed by a read of the data-cache miss counter. If the counter shows unexpected misses, a violation is reported.

The three initialized operand registers used in the examples below are:

- **rOneWord**: The original, 1-word capability
- **rInvalid**: A capability pointing to another location, but with the tag cleared
- **rDest**: The destination register of all instructions, and the address of any loads

A register operand used as a capability will be prepended with "c", e.g. **cDest**. One counterexample our generator discovered was:

CBuildCap

```
lb.cap rDest, cDest[0]
cbuildcap cDest, rInvalid, cInvalid
lb.cap rDest, cDest[0]
```

While the first `lb.cap` is waiting to commit, *CBuildCap* transiently constructs a valid capability from an untagged value *and loads through that capability*. The load through the forged capability will miss the L1 data cache and thus increase the miss counter indicating a violation.

In addition, this generator produces counterexamples for variants of *CSetBounds* as well as *CUnseal* and *CInvoke* that illegally dereference sealed capabilities in transient execution.

Analysis of Data-CSC Violations

These data-CSC violations allow trivial circumvention of capability protection using transient execution. Allowing faulting addresses to proceed to issue memory requests is a classic Meltdown-style vulnerability, similar to forwarding data during a page permissions fault in the original Meltdown attack [5]. Thus, we dub this new family of transient-execution vulnerabilities Meltdown-CF: Meltdown Capability Forgery.

The data-CSC violations in CHERI-Toooba are due to loading through forwarded capabilities without accounting for exception checks. Several crucial CHERI exceptions require a full bounds check, which is not performed until the cycle after the execution result has been forwarded, as shown in Figure 3.

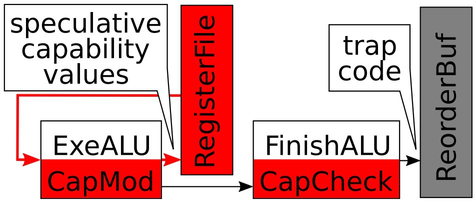

Fig. 3: The CHERI-Toooba ALU with late exception checks.

To enforce data-CSC, an implementation must either perform all capability checks before forwarding the register result, or must develop a new mechanism to prevent issuing memory operations that depend on a pending capability check. We evaluate the seriousness Meltdown-CF with a user-level exploit in a pure-capability process on CheriBSD in Section VIII, and present as well as evaluate a solution in Section IX-A.

B. Instruction-CSC Testing

Instruction-CSC is violated if we execute instructions not allowed by the committed PCC and register state. Our instruction-CSC counterexample generator trains the BTB with a sequence of jumps within a PCC with permissive bounds. It then restricts the bounds of PCC and executes a similar sequence to trigger mispredictions with permissive bounds.

Instruction-CSC is defined to allow any instruction to be *fetched*, but we must detect if CSC-violating instructions are *executed*. In our generator, we used the `auipcc` instruction, which copies the current PCC into a register, as a portable mechanism for observing PCC in Execute without requiring special-purpose counters. We found this counterexample:

```
cjalr x0, x23
auipcc x25, 0
lb.cap x24, x25[0]
```

This counterexample triggers a misprediction with `cjalr`, and then with `auipcc` loads the current PCC into a register. This capability is then used to perform a load (`lb.cap`) – allowing us to count unexpected data cache misses. This strategy violates data-CSC using an instruction-CSC violation.

Analysis of Instruction-CSC Violations

CHERI-Toooba violates instruction-CSC due to predicting the entirety of PCC, including the bounds and permissions. While this design minimizes performance overhead relative to the base Toooba microarchitecture, it allows transient execution into foreign compartments. Two microarchitectural solutions for solving instruction-CSC violations are presented with implementations and performance overheads in Section IX-B.

C. Evaluation of Instruction Generators

Our test generators are reasonably efficient at discovering CSC counterexamples. As described in Section VII-A,

cbuildcap	*csetbounds*	*csetbounds exact*	*csetbounds immediate*	*cunseal*	*cinvoke*
0.10%	0.05%	0.20%	0.20%	0.20%	1.80%

TABLE I: Distribution of sequences that produce counterexamples for the data-CSC generator during 2000 runs. Overall, 2.55% of runs produced a counterexample.

the data-CSC generator produces the six classes of counterexamples listed in Table I, with an overall probability of 2.55% of discovering a counterexample for any sequence in 2000 runs. The skew towards the *CInvoke* counterexamples can be explained by a bias in the shrinking mechanism of TestRIG [18]. The instruction-CSC generator produced one class of counterexamples with a chance of 0.35% measured in 2000 runs.

In this section, we presented a guided testing approach for the CSC. We have validated our findings with a full transient-execution attack presented in Section VIII. Furthermore, we have conducted manual inspection of the HDL code and code to better understand our findings and to discover whether there could be other sources of CSC violations. While we did not find additional sources of violations, our testing is not exhaustive and might have potentially missed some cases. This is future work, which has to formally specify contracts and then can exhaustively test them.

VIII. Meltdown-CF Attack

In this section, we present a successful Meltdown-CF attack on the CHERI-RISC-V platform we used for the previous evaluation.

A. Attack Setup

We ran all our experiments on the CHERI-Toooba platform presented in Section VI. Our goal in this attack is to show how Meltdown-CF can break spatial memory safety on CHERI [1] in a vulnerable implementation. We execute two protection domains in the same address space, which is a common use case for CHERI processors [28]. In this attack, one domain attempts to access the memory of the other domain without having a valid capability to it.

B. Conducting the Attack

We selected the `cbuildcap` variant of Meltdown-CF for this demonstration, though we could have used any other variant presented in Section VII to demonstrate that we can break CHERI's spatial memory safety in speculation. Our approach is to illegally build a capability from bits in speculation, i.e., without deriving it legally from any valid capability. We then use this capability to illegally access memory through this forged capability and encode it in the cache side channel.

First, we assemble the bit pattern of the capability by simply writing two 64 bit data words to memory to produce an invalid, untagged capability. We then load these 128-bits into a capability register, and feed it to `cbuildcap` with an insufficient authorizing capability. When running on CHERI-Toooba, the hardware optimistically sets the tag and forwards this capability in the pipeline, as explained in Section VII-A. We then use this forwarded capability and access the memory. Next, we encode the data through the cache by accessing an array and probing it later.

Normally, executing the illegal `cbuildcap` instruction non-speculatively would lead to an exception that terminates the process. In order to sustain a long-running attack, we take inspiration from previous Meltdown-style attacks and embed the code in an `if` statement [29]. This way, the `cbuildcap` exception occurs only in control-flow misspeculation, and thus will not lead to process termination on an exception. The core of the attack code is depicted below:

```
meltdown_cf:
  clb t0, 0(ca1) // load variable to branch on
  blt x0, t0, end // branch mispredicted to next instruction
  cbuildcap ct1, ca1, ca0 // generate speculative capability
  clb t2, 0(ct1) // load secret
  and t2, t2, a6 // apply bit mask to get subset of secret
  sll t2, t2, a7 // shift secret to achieve cache line granularity
  cincoffset ct2, ca5, t2 // add offset to probing buffer
  clb t2, 0(ct2) // transmit secret via load to buffer
end:
  cret // return to caller
```

We are conducting a classical flush+reload attack. First, we need to have a high signal-to-noise ratio for the cache side channel. This required evicting all cache lines of our side-channel buffer, which needed particular care due to lazy allocation of pages by the operating systems as well as compiler optimizations. Second, we need to ensure that the misprediction of the `if` statement guarding the `cbuildcap` instruction is resolved as late as possible. Thus, we make the `if` statement dependent on a load that will miss all caches.

In Figure 4, the steps of the Meltdown-CF attack presented in this section are summarized.

C. Evaluation

We ran our attack on CHERI-Toooba with the CheriBSD operating system. We measured a reading rate of approximately 1.4 bytes/second for our attack, with an error rate of 0%, which means that we read out the correct secret in every attack run. On an industry-scale processor, we expect our attack to significantly scale up in speed compared to our FPGA platform. Our attack demonstrates that Meltdown-CF vulnerabilities can be applied to running CHERI systems in the field. Our attack or similar ones can be used to considerably harm CHERI applications running on vulnerable processors.

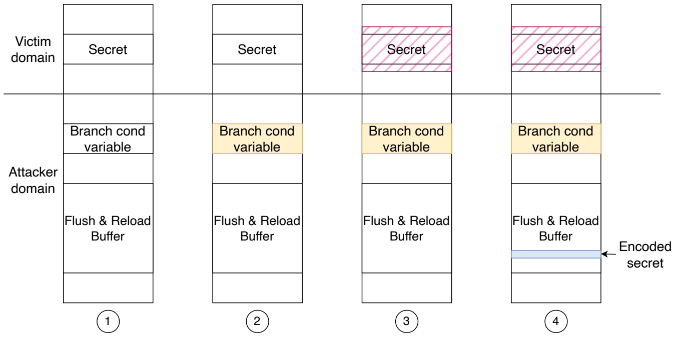

Fig. 4: This figure summarizes the steps for the Meltdown-CF attack. The first diagram shows the system in the start state, where the attacker and victim are in two different protection domains. In the second diagram, the access to the branch condition variable will miss all caches to delay resolution of misspeculation. In the third diagram, the attacker creates a speculative capability covering the memory where the secret is stored. In the final diagram, the attacker accesses the secret and transmits it by loading a cache line.

IX. CSC Enforcement

Our TestRIG generators found that CHERI-Toooba had multiple violations of both data-CSC and instruction-CSC. In this section, we demonstrate that enforcing CSC in CHERI-Toooba is possible at full performance.

A. *Data-CSC Enforcement*

CHERI-Toooba is vulnerable to Meltdown-CF due to forwarding capabilities before exception checks. The CHERI-Toooba exception check is split into two pipeline stages: ExeALU and FinishALU as depicted in Figure 3. Result values are forwarded from ExeALU, accelerating the common case, but the exception is reported in FinishALU, marking the instruction for cancellation at Commit. While a safer implementation could be imagined, faulting instruction semantics generally imply a speculative result with a late flush, so it is likely that other implementations would follow the same dangerous pattern.

We propose to change the CHERI-RISC-V specification to clear tags on capability violations to suggest microarchitectures that conform to data-CSC, preventing Meltdown-CF vulnerabilities. This implies that implementations should reflect failure atomically in the forwarded result, removing the opportunity for transient, illegal capability. To implement tag-clearing semantics in CHERI-Toooba, we moved the entire exception check to the ExeALU stage so that forwarded tag of the result could reflect any error conditions. We verified using our data-CSC TestRIG generator that this change eliminated all known Meltdown-CF violations. This modified design did not change resource utilization, e.g., the VCU118 implementation, had a mean increase of +0.86% for look-up table (LUT) usage and a mean decrease -0.5% in FF usage for the cores within the range of synthesis variation. Our implementation of this change did not introduce cycle delays under any circumstances, and therefore does not affect performance. While this design passing timing at 25MHz on the VCU118 prototype and at 50MHz on the DE10 prototype, a high-performance ASIC implementation may need to implement rare capability operations using multiple cycles.

We responsibly disclosed our discovery of Meltdown-CF to the CHERI-RISC-V team and the Arm Morello team. The CHERI-RISC-V team responded by transitioning the CHERI-RISC-V specification to tag-clearing semantics to increase robustness against transient-execution attacks. A version of CHERI-RISC-V with our proposed tag-clearing semantics is now in the process of being ratified by RISC-V International. The Morello team had inadvertently already taken this approach in their specification for consistency with the Arm architecture, but reviewed their in-progress microarchitectural design.

B. *Instruction-CSC Enforcement*

Instruction-CSC constrains what instructions can be executed to those that lie within capabilities in the committed state of the register file, e.g., the program-counter capability (PCC). However, CHERI-Toooba stores the bounds of previous PCC targets in the Branch Target Buffer (BTB) and freely predicts PCC bounds when the Fetch stage encounters a jump. This provides ideal performance, but leads to violations of instruction-CSC, as foreign targets with their own bounds are reintroduced from the BTB. Another solution to this problem in an in-order core is predicting only the address of PCC and *forwarding* the bounds from the last-executed branch. This is possible because the bounds of PCC are not needed in the early stages of the pipeline, such that bounds can be read and checked in the Execute stage of the pipeline when all previous jumps have executed. Thus, an in-order core can forward PCC bounds for efficiency and performance, while incidentally enforcing instruction-CSC.

For superscalar out-of-order pipelines such as CHERI-Toooba, a straightforward, safe, and efficient strategy to enforce instruction-CSC is to simply predict that the bounds of the PCC did not change. If we allow only those instructions to execute that lie within the PCC written by the latest-executed branch (which in turn must be derived legally according to data-CSC), then an implementation will comply with instruction-CSC.

We call this implementation strategy *SinglePCC*, illustrated in Figure 5. We removed bounds from all program-counter state everywhere in the pipeline and replaced them with a single PCC register. Any logic in the pipeline that needs the complete PCC simply appends the bounds from the PCC register to the current instruction address, thus speculating that the bounds of PCC have not changed. Any jump to a capability with different bounds will trigger a flush at the Commit stage to ensure all older instructions commit with the correct bounds. We verified that this design eliminates known instruction-CSC vulnerabilities using our TestRIG generators.

We synthesized the SinglePCC design for two FPGA boards: the VCU118 board and the DE10Pro board. Sin-

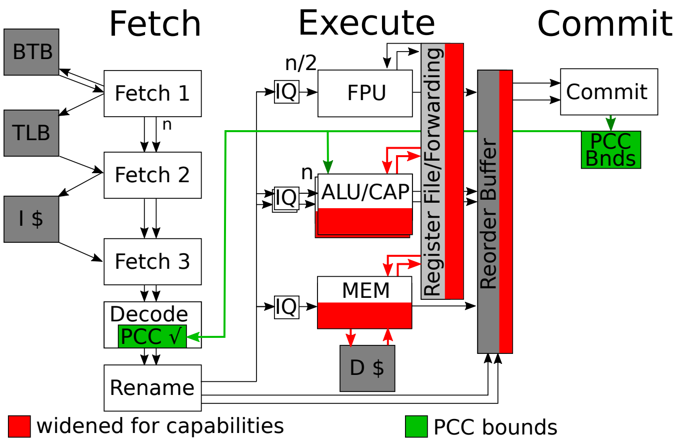

Fig. 5: CHERI-Toooba with SinglePCC prediction, n=2.

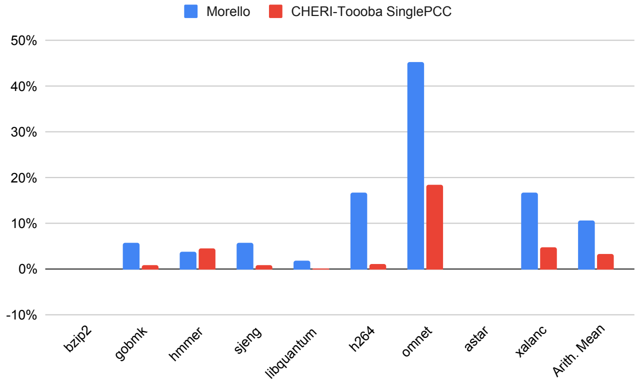

Fig. 6: SPECint2006 cycle overhead of Morello bounds forwarding and CHERI-Toooba with SinglePCC over full-prediction baseline.

glePCC simplifies the design by deriving all instances of PCC bounds in the pipeline from a single register, reducing register and LUT elements used. We measure a reduction of approximately 4.89% for LUTs on both boards as well as 5.48% and 6.67% fewer registers on the DE10Pro board and VCU118 board respectively. This constitutes a significant reduction in area, eliminating over 15% of the logic required to add CHERI support to the Toooba core [27].

We compare with Arm's Morello implementation of CHERI, which takes a version of the bounds-forwarding approach, but in a superscalar out-of-order pipeline. Unfortunately, PCC readers in Morello must block until the previous jump is executed, causing delay in many common cases. While transient-execution vulnerability mitigation was not a design goal for Morello, this design partially enforces instruction-CSC by preventing wild PCCs from being used for data access, but still allows wild execution.

Figure 6 compares the cycle overhead of SinglePCC, which fully enforces instruction-CSC, against Morello's bounds forwarding. The overhead for Morello's bounds forwarding is measured using an unsafe compiler workaround that uses

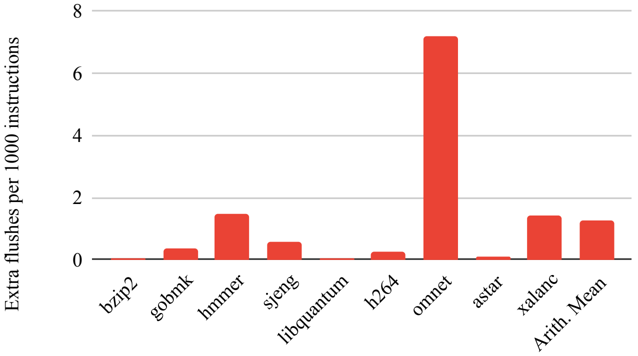

Fig. 7: Extra pipeline flushes due to capability jumps that change bounds when using SinglePCC in SPECint2006 with distinct bounds for each dynamically loaded library.

legacy integer jumps in place of pure capability control flow to eliminate stalls on PCC bounds dependencies [30]. Despite flushing on every cross-library call, SinglePCC incurs only 3.43% overhead on average, versus 10.7% for Morello's bounds forwarding. While a pipeline flush is much more expensive than a single pipeline dependency stall, reads of PCC values prove far more common in dynamic execution than cross-library jumps. Figure 7 shows the number of flushes per 1000 instructions in the benchmarks; only Omnet sees more than 2 flushes per 1000 instructions, and is the only benchmark with a cycle overhead greater than 5%. On average, our results show less than 1.2% extra flushes per 1000 instructions in SPECint2006.

Future implementations have several options for achieving greater performance than SinglePCC while still respecting instruction-CSC. For example, rather than a full pipeline flush on every change of PCC bounds, we could predict jumps that would change PCC, and pause the following instructions in Rename until the jump has committed. Assuming effective prediction, this should cut cycle overhead in half. Beyond this, more sophisticated solutions might develop a full bounds prediction engine that respects committed register state.

Conclusion

We have articulated the Capability Speculation Contract (CSC) that precludes transient-execution attacks against CHERI protection, and have demonstrated complete enforcement at a 3.43% performance loss in a superscalar, out-of-order implementation, with hope for further optimizations in the future. Our discovery of CSC violations in the CHERI-Toooba implementation, resulting in the Meltdown-CF vulnerability, is ample proof that such a clearly defined and testable contract is necessary to develop safe superscalar out-of-order CHERI processors. This work has paved the way for a standardized CHERI-RISC-V extension (ratification pending) with tag-clearing semantics, to encourage implementations that are safe from Meltdown-CF.

References

[1] R. N. M. Watson, P. G. Neumann, J. Woodruff, M. Roe, H. Almatary, J. Anderson, J. Baldwin, G. Barnes, D. Chisnall, J. Clarke, B. Davis, L. Eisen, N. W. Filardo, F. A. Fuchs, R. Grisenthwaite, A. Joannou, B. Laurie, A. T. Markettos, S. W. Moore, S. J. Murdoch, K. Nienhuis, R. Norton, A. Richardson, P. Rugg, P. Sewell, S. Son, and H. Xia, "Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 9)," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-987, Sep. 2023.

[2] K. Nienhuis, A. Joannou, T. Bauereiss, A. Fox, M. Roe, B. Campbell, M. Naylor, R. M. Norton, S. W. Moore, P. G. Neumann *et al.*, "Rigorous engineering for hardware security: Formal modelling and proof in the CHERI design and implementation process," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2020, pp. 1003–1020.

[3] R. Grisenthwaite, G. Barnes, R. N. Watson, S. W. Moore, P. Sewell, and J. Woodruff, "The Arm Morello evaluation platform—validating CHERI-based security in a high-performance system," *IEEE Micro*, vol. 43, no. 3, pp. 50–57, 2023.

[4] "CHERI security technology," https://codasip.com/solutions/riscv-processor-safety-security/cheri/, November 2023.

[5] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium*, August 2018.

[6] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2019, pp. 1–19.

[7] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. B. Abu-Ghazaleh, "Spectre returns! Speculation attacks using the return stack buffer," in *12th USENIX Workshop on Offensive Technologies, WOOT 2018*, C. Rossow and Y. Younan, Eds. USENIX Association, 2018. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/koruyeh

[8] G. Maisuradze and C. Rossow, "ret2spec: Speculative execution using return stack buffers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, January 2018, pp. 2109–2122. [Online]. Available: https://doi.org/10.1145/3243734.3243761

[9] J. V. Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, and F. Piessens, "LVI: Hijacking transient execution through microarchitectural load value injection," in *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2020, pp. 54–72.

[10] J. Stecklina and T. Prescher, "LazyFP: Leaking FPU register state using microarchitectural side-channels," *CoRR*, vol. abs/1806.07480, 2018. [Online]. Available: http://arxiv.org/abs/1806.07480

[11] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. V. Bulck, and Y. Yarom, "Fallout: Leaking data on meltdown-resistant CPUs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 769–784. [Online]. Available: https://doi.org/10.1145/3319535.3363219

[12] M. Schwarz, M. Lipp, D. Moghimi, J. V. Bulck, J. Stecklina, T. Prescher, and D. Gruss, "ZombieLoad: Cross-privilege-boundary data sampling," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 753–768. [Online]. Available: https://doi.org/10.1145/3319535.3354252

[13] J. Horn, "speculative execution, variant 4: speculative store bypass," https://bugs.chromium.org/p/project-zero/issues/detail?id=1528, February 2018.

[14] J. Ravichandran, W. T. Na, J. Lang, and M. Yan, "PACMAN: Attacking arm pointer authentication with speculative execution," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 685–698. [Online]. Available: https://doi.org/10.1145/3470496.3527429

[15] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium*, W. Enck and A. P. Felt, Eds. USENIX Association, 2018, pp. 991–1008. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[16] M. Guarnieri, B. Köpf, J. Reineke, and P. Vila, "Hardware-software contracts for secure speculation," in *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2021.

[17] F. A. Fuchs, J. Woodruff, P. Rugg, M. van der Maas, A. Joannou, A. Richardson, J. Clarke, N. W. Filardo, B. Davis, J. Baldwin, P. G. Neumann, S. W. Moore, and R. N. M. Watson, "Architectural contracts for safe speculation," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2023, pp. 578–586.

[18] A. Joannou, P. Rugg, J. Woodruff, F. A. Fuchs, M. van der Maas, M. Naylor, M. Roe, R. N. M. Watson, P. G. Neumann, and S. W. Moore, "Randomized testing of RISC-V CPUs using direct instruction injection," *IEEE Design & Test*, 2023.

[19] The White House, "Back to building blocks: A path toward secure and measurable software," Tech. Rep., February 2024. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf

[20] N. W. Filardo, B. F. Gutstein, J. Woodruff, J. Clarke, P. Rugg, B. Davis, M. Johnston, R. Norton, D. Chisnall, S. W. Moore, P. G. Neumann, and R. N. M. Watson, "Cornucopia reloaded: Load barriers for cheri heap temporal safety," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 251–268.

[21] O. Oleksenko, C. Fetzer, B. Köpf, and M. Silberstein, "Revizor: Testing black-box CPUs against speculation contracts," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2022, pp. 226–239.

[22] Arm Limited, "Cache speculation side-channels," Tech. Rep. 2.5, June 2020. [Online]. Available: https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability

[23] R. N. M. Watson, J. Woodruff, M. Roe, S. W. Moore, and P. G. Neumann, "Capability Hardware Enhanced RISC Instructions (CHERI): Notes on the Meltdown and Spectre Attacks," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-916, February 2018. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-916.pdf

[24] F. A. Fuchs, J. Woodruff, S. W. Moore, P. G. Neumann, and R. N. Watson, "Developing a test suite for transient-execution attacks on RISC-V and CHERI-RISC-V," in *Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2021.

[25] J. Woodruff, P. Rugg, J. Clarke, F. A. Fuchs, R. S. Nikhil, and M. van der Maas, https://github.com/CTSRD-CHERI/Toooba, 2024.

[26] S. Zhang, A. Wright, T. Bourgeat, and Arvind, "Composable building blocks to open up processor design," in *51st Annual IEEE/ACM International Symposium on Microarchitecture, (MICRO)*. IEEE Computer Society, 2018, pp. 68–81. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00015

[27] P. D. Rugg, "Efficient spatial and temporal safety for microcontrollers and application-class processors," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-984, Jul. 2023.

[28] D. Gao and R. N. M. Watson, "Library-based compartmentalisation on CHERI," in *Programming Languages for Architecture 2023*, Orlando, FL, USA, Jun. 2023.

[29] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. Von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC'19. USA: USENIX Association, August 2019, pp. 249–266.

[30] R. N. M. Watson, J. Clarke, P. Sewell, J. Woodruff, S. W. Moore, G. Barnes, R. Grisenthwaite, K. Stacer, S. Baranga, and A. Richardson, "Early performance results from the prototype Morello microarchitecture," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-986, Sep. 2023.