Proving security properties of CHERI-MIPS

Kyndylan Nienhuis Alexandre Joannou Peter Sewell

Computer Laboratory, University of Cambridge
{first.last}@cl.cam.ac.uk

Abstract: CHERI-MIPS is an instruction set architecture that provides hardware support for secure encapsulation and fine-grained memory protection. The guarantees it intends to offer are described in high-level prose, which makes it difficult to understand what they precisely are, whether they are true, and whether they indeed provide memory protection. We describe ongoing work on proposing formal definitions of these guarantees and proving that they are true.

Introduction

CHERI [6, 1, 7] is an extension to instruction set architectures (ISAs) that enables fine-grained memory protection by adding hardware support for capabilities. A capability contains a cursor, bounds information and permission bits. Memory accesses that dereference a capability throw an exception if its cursor points outside its bounds or if the type of access is not allowed according to its permission bits. To prevent tampering with the bounds and permissions of in-memory capabilities, each capability-sized region of memory has a *tag* associated with it. The tag is preserved if a capability is changed through special instructions that the CHERI ISA provides, but it is cleared if the in-memory byte representation of the capability is directly overwritten, which makes the capability invalid.

The memory protection CHERI offers is meant to enable secure software compartmentalisation and to prevent a large class of attacks that exploit memory safety bugs. Since the smallest mistake has the potential to be an exploitable security flaw it is crucial that there are no errors in the capability mechanism. To have more confidence in the mechanism the authors of CHERI have created a formal model [2] of the most developed version of CHERI, namely CHERI-MIPS. The formal model is written in the L3 specification language [3], which exports to Isabelle/HOL [5] and HOL4 [4]. The model is executable and is complete enough to boot FreeBSD, totaling around 10K non-comment non-blank lines of specification; it has been used as one of the principal design tools during CHERI development.

The authors have also defined several prose security properties. Unfortunately, these properties are not detailed enough to actually validate the formal model. For example, *capability integrity* states that invalid capabilities cannot be "dereferenced" [6, §2.3.1], while the formal model allows one to copy the cursor of an invalid capability to a data register and then dereference it, if the cursor is within the bounds of the Default Data Capability. Does that mean the formal model violates the security property, or does it mean that copying the cursor is not considered "dereferencing the capability"? Another example is *capability provenance* which states that "valid capabilities can only be constructed by deriving them from existing valid capabilities" [6, §2.3.1]. The formal model allows one to create a valid capability cap_{new} equal to an old valid capability cap_{old1} but with the bounds of an invalid capability cap_{old2}, if the bounds of cap_{old2} are smaller than those of cap_{old1}. Does the formal model violate the security property, or did the authors mean that cap_{new} only derives from cap_{old1} and not from both cap_{old1} and cap_{old2}?

To solve these ambiguities we define formal security properties in Isabelle/HOL and we prove that these properties hold in the L3 model. While constructing these proofs we uncovered several bugs in the L3 model which were consequently patched. This is work in progress, though much has been finished.

Capability derivations

Before we define our security properties we introduce a type that describes capability derivations:

- $Restricted\ r\ r'$ means that the bounds and/or permissions of the capability in register r have been restricted (or remain equal) and the result has been copied to register r'.
- $Loaded\ auth\ a\ r$ means that under the authority of the capability in register $auth$ the capability at address a has been loaded to register r.
- $Stored\ auth\ r\ a$ is similar to the previous case.
- $Sealed\ auth\ r\ r'$ means that under the authority of the capability in register $auth$ the capability in register r has been sealed and copied to register r'. A sealed capability is unusable until it is unsealed. Sealed capabilities contain an *object type*, which allows multiple sealed capabilities to be linked.
- $Unsealed\ auth\ r\ r'$ is similar to the previous case.
- $Invoked\ r\ r'$ means that the capabilities in registers r and r' have been invoked: they are unsealed and the execution jumped to the cursor of the first. Mutually untrusting compartments can use this mechanism to communicate without exposing their own capabilities.

Then we define a function $DerivationsOfStep$ that describes which derivations happen during an execution step. Its definition is based on the instruction that is executed. For example, if the *CMOVZ* instruction is executed with parameters cd, cb and rt then $DerivationsOfStep\ s$ equals $\{Restricted\ cb\ cd\}$ if $rt = 0$ and $\emptyset$ otherwise. As another

example, if the *CSeal* instruction is executed with parameters cd, cs and ct and does not throw an exception, then $DerivationsOfStep\ s$ equals $\{Sealed\ ct\ cs\ cd\}$.

Basic security properties

We define *derivation correctness* that describes the intended results of each type of derivation. Below is an export from Isabelle/HOL for derivations of type $Sealed$. It reads as follows: for all states s and s' and registers $auth$, r and r' (Line 1), if s' is a successor state of s (Line 2) and if the capability in register r has been sealed and copied to register r' under the authority of the capability in register $auth$ (Line 3), then the capability that is used as authority has the permission to seal capabilities (Line 6), is valid (Line 7) and not sealed (Line 8); the object type t (defined in Line 4-5) is contained in the memory segment of the capability that is used as the authority (Line 9); the original capability is not sealed (Line 10); and the resulting capability is equal to the original capability, except that the resulting capability is sealed and has object type t (Line 11-12).

```
(1)  for all  s s' auth r r'.
(2)  if s' ∈ NextStates s
(3)    and Sealed auth r r' ∈ DerivationsOfStep s
(4)  then let t = Cast (Base (CapReg auth s)) +
(5)                Cast (Offset (CapReg auth s)) in
(6)       PermitSeal (CapReg auth s)
(7)       and Tag (CapReg auth s)
(8)       and not IsSealed (CapReg auth s)
(9)       and Cast t ∈ Segment (CapReg auth s)
(10)      and not IsSealed (CapReg r s)
(11)      and CapReg r' s' = CapReg r s with
(12)            IsSealed ← True, ObjectType ← t
```

Then we define *capability nonforgeability* that states that $DerivationsOfStep$ captures all derivations of capabilities. The formal definition reads as follows: for all states s and s' and locations loc (which can be registers or memory addresses) (Line 1), if s' is a successor state of s (Line 2), if there has not been a derivation of which loc is the destination (Line 3-4) and if the capability at location loc in the new state is valid (Line 5), then that capability remained unchanged during the execution step (Line 6).

```
(1) for all s s' loc.
(2) if s' ∈ NextStates s
(3)   and not exists prov. prov ∈ DerivationsOfStep s
(4)                        and loc ∈ Destinations prov
(5)   and Tag (Cap loc s')
(6) then Cap loc s' = Cap loc s
```

Theorem. The L3 model satisfies *capability nonforgeability* and *derivation correctness* for all types of derivations.

Our proof in Isabelle/HOL consists of 20k non-comment non-blank lines. A large part of this proof (9k) has been generated, for example commutativity lemmas for all the auxiliary functions defined in the L3 model and lemmas for the ∼160 instructions that do not derive new capabilities.

Conclusion

The prose security properties described in the CHERI ISA are not precise enough to be used as validation. To overcome this we have formally defined two security properties in terms of the L3 model of CHERI-MIPS, namely *capability nonforgeability* and *derivation correctness*. Furthermore, we proved that the L3 model satisfies these properties increasing our confidence that the capability mechanism is correct.

As future work we plan to formally define properties that describe when data can be stored to and loaded from memory, when capability registers are accessible and when address translation can be changed. Then we plan to formally define the *permissions of a compartment* which is roughly speaking the union of the permissions of all the capabilities that are transitively reachable from that compartment. The permissions of a compartment should be monotonic and we believe we can prove this from our basic security properties. We then plan to use compartment monotonicity to prove memory isolation in specific examples.

Acknowledgements This work is part of the REMS and CTSRD projects and was funded by a Gates studentship (Nienhuis), EPSRC (EP/K008528/1), DARPA and AFRL (FA8750-10-C-0237). The views, opinions, and/or findings contained in this paper are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the Department of Defense or the U.S. Government. We thank the CHERI team for extensive discussions.

References

[1] CHERI. https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/.

[2] The L3 model of CHERI/MIPS. https://github.com/acjf3/l3mips.

[3] A. C. Fox. Directions in ISA specification. In *ITP*, pages 338–344, 2012.

[4] M. Gordon and A. Pitts. The HOL logic and system. In *Real-Time Safety Critical Systems*, volume 2, pages 49–70. Elsevier, 1994.

[5] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2012.

[6] R. N. Watson, P. G. Neumann, J. Woodruff, M. Roe, J. Anderson, J. Baldwin, D. Chisnall, B. Davis, A. Joannou, B. Laurie, S. W. Moore, S. J. Murdoch, R. Norton, S. Son, and H. Xia. Capability hardware enhanced RISC instructions: CHERI instruction-set architecture (version 6). Technical report, University of Cambridge, Computer Laboratory, 2017.

[7] J. Woodruff, R. N. M. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton, and M. Roe. The CHERI capability model: Revisiting RISC in an age of risk. In *Proc. ISCA*, 2014.