

Formally proving nonforgeability of capabilities (work in progress)

Kyndylan Nienhuis and Peter Sewell

April 23, 2016

Security properties of CHERI


CHERI guarantees certain properties such as:¹

- ▶ “Nonbypassability of the capability mechanism”
- ▶ “Nonforgeability of capabilities”
- ▶ “Integrity of the tags”

These properties allow higher level guarantees:

- ▶ “The separation kernel will provide controlled sharing among different virtual machine partitions, and otherwise strict isolation.”

We are investigating how to formally prove them.

¹From CHERI Formal Methods Report §2.2, §3.2 and §10.3 

Formally stating the properties

What does “controlled violation” mean in the following?

- ▶ The specifications of *CCall* and *CReturn* allow “controlled violation of inductive preservation of capability monotonicity.”

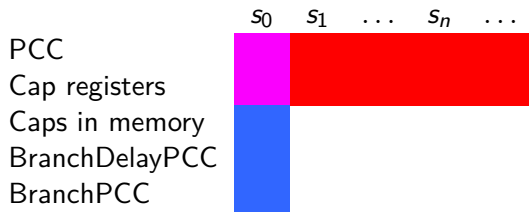
What does “controlled sharing” mean in the following?

- ▶ “The separation kernel will provide controlled sharing among different virtual machine partitions, and otherwise strict isolation.”

To formally prove them, we need to make these statements precise.

Nonforgeability of capabilities

Let s_{i+1} be the state obtained by executing one instruction at state s_i . The capabilities in the *PCC* or registers of any of those states are *reachable* from s_0 (the red or purple cells).



We want to abstract away from executions, and define the reachable capabilities just in terms of the capabilities in s_0 (the blue or purple cells).

Nonforgeability of capabilities - 2

We recursively define the set A of *available* capabilities of s_0 .

- ▶ $PCC, BranchPCC, BranchDelayPCC, CapReg(i) \in A$ for all register indices i ,
- ▶ If $cap \in A$ and $cap' \leq cap$ then $cap' \in A$ (e.g. cap' has less permissions, or specifies a smaller region of memory than cap),
- ▶ If $cap \in A$ and cap allows us to load cap' from memory, then $cap' \in A$ (we have not yet formally defined this).

Our goal is to prove *reachable caps* \subseteq *available caps*.

State of proof

We prove with induction to i that the *PCC* and cap registers of s_i are *available* in s_0 . We have proved the induction step for

- ▶ All the 122 instructions that do *not* change the *PCC* or cap registers,
- ▶ 10 of the 32 instructions that *do* change the *PCC* or cap registers,
- ▶ Exception handling for all instructions.

All proofs are in Isabelle/HOL using the Isabelle export of the L3 model of CHERI. There are 2708 lines of general lemmas and tactics, and 1720 specifically about this proof.

State of proof - 2

Of the remaining 22 instructions

- ▶ 18 change the memory; here we need to precisely define which capabilities are available in the memory.
- ▶ 2 seal and unseal caps; here we need to precisely define under what conditions a sealed capability is available.
- ▶ 2 decrease the permissions of caps; we think these are easy to prove.

Open questions

We are not sure whether the following is intentional:

- ▶ The specified region of a capability can overflow: a capability with base 2^{63} and length $2^{64} - 1$ can be turned into one with base 0 and length $2^{63} - 1$.
- ▶ If there are multiple TLB hits, then the behaviour is undefined (according to the L3 model), which might mean that the *CLC* instruction can load any capability.