ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

VeriSMo: A Verified Security Module for Confidential VMs

Ziqiao Zhou[*], Anjali[†], Weiteng Chen[*], Sishuai Gong[‡], Chris Hawblitzel[*], Weidong Cui[*]

[*]Microsoft Research, [†]University of Wisconsin-Madison, [‡]Purdue University

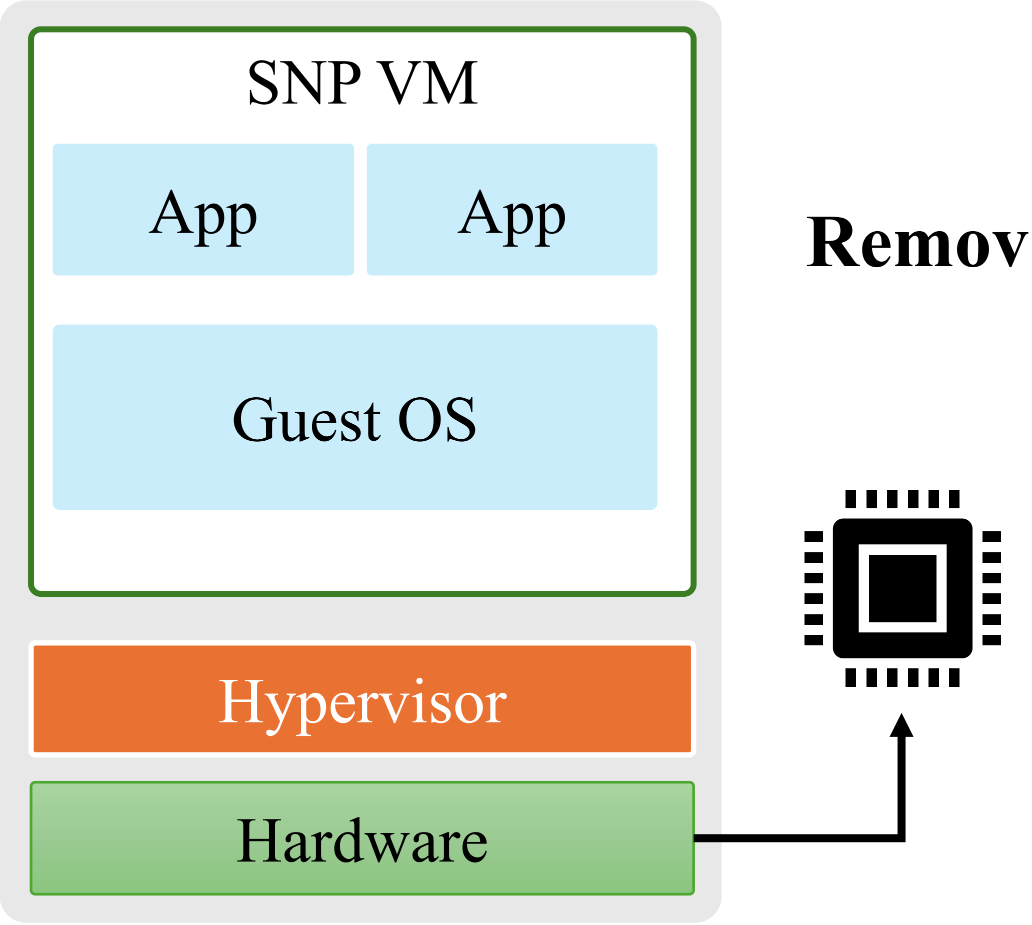
Microsoft Research

WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

PURDUE UNIVERSITY

https://github.com/microsoft/verismo

Confidential VMs



SNP VM

App

App

Guest OS

Hypervisor

Hardware

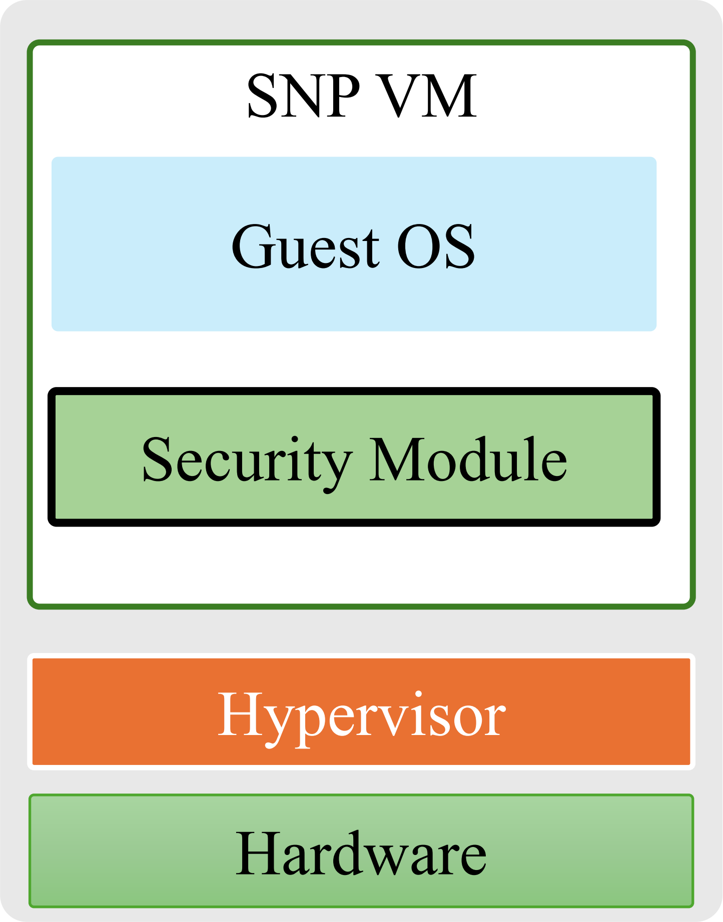
Remove the trust in the hypervisor

AMD Secure Encrypted Virtualization (SEV) - Secure Nested Paging (SNP)

Intel Trusted Domain Extension

Arm Confidential Computing Architecture (CCA)

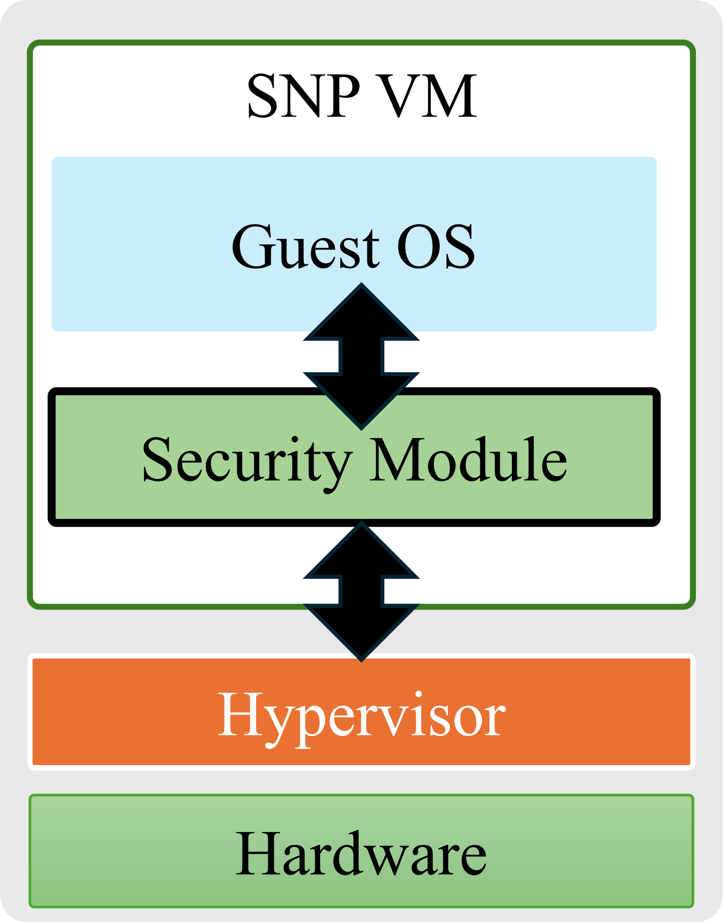
Why do we need a security module?



Untrusted hypervisor-based security features

- ✓ Hypervisor-based code integrity protection
- ✓ Virtual Trusted Platform Module (vTPM) for extended runtime attestation

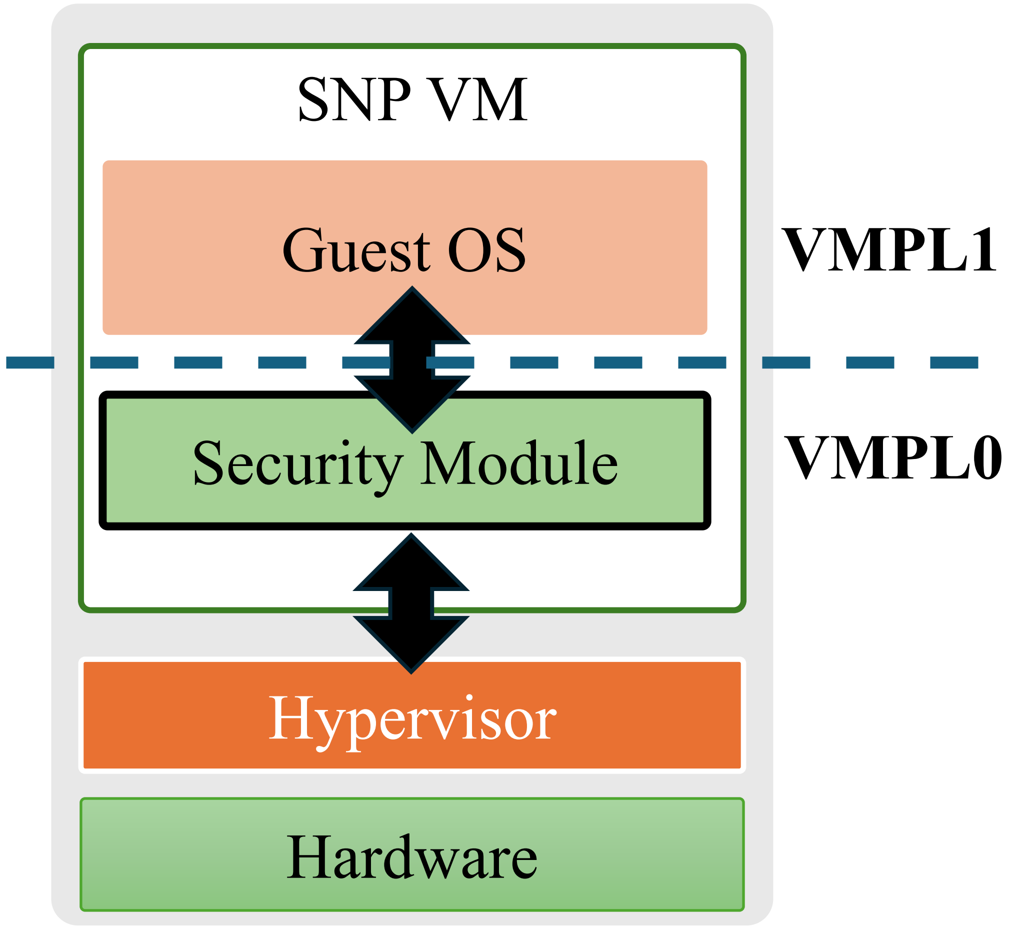
What does the security module do?



A VM firmware at highest privilege level provides APIs to the guest OS

- Replace hypervisor-based security features
  - ✓ Code integrity protection
  - ✓ vTPM
- Manage security-sensitive changes
  - ✓ Setup CPU contexts for Guest OS.
  - ✓ Manage SNP guest memory:
    access permissions, private/shared

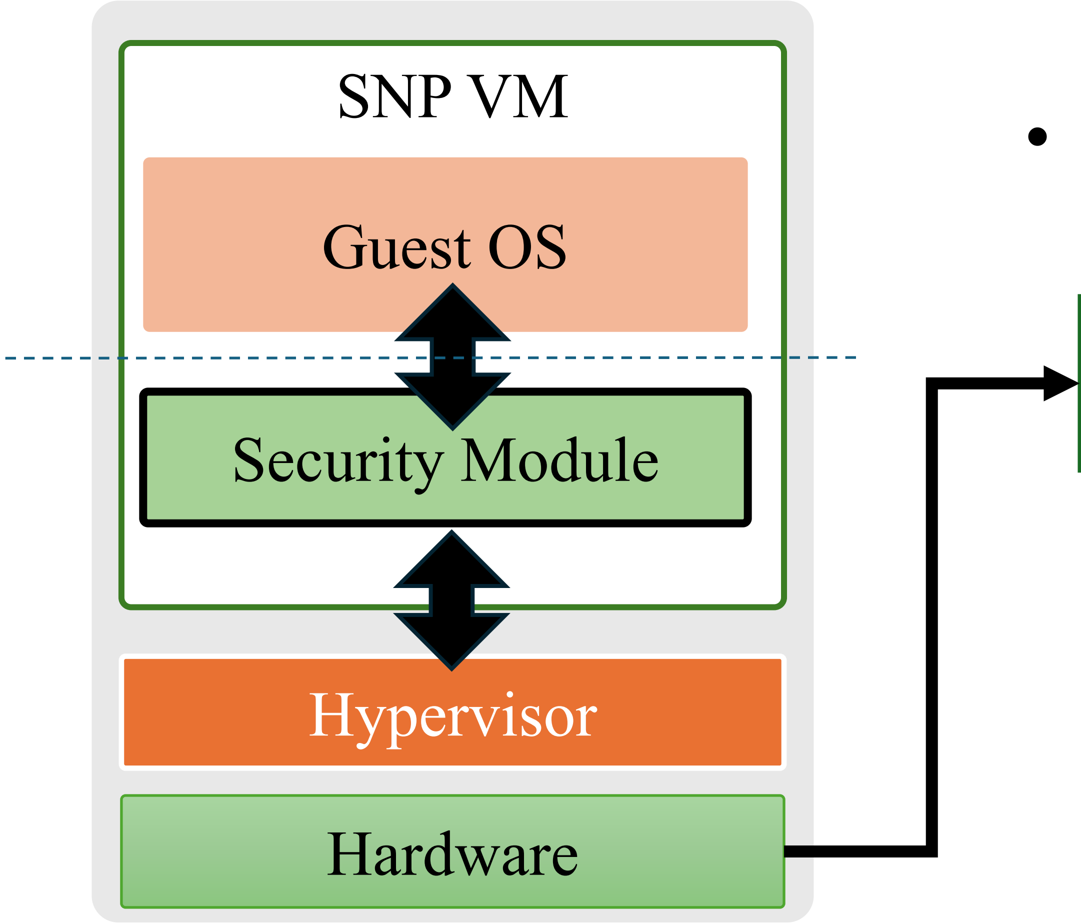
The security module should be isolated



Runs at VM Privilege Level 0 (VMPL0)

- A hardware-based isolation
- Isolated memory with different access permission
- Isolated CPU context
- Only share the VM memory encryption

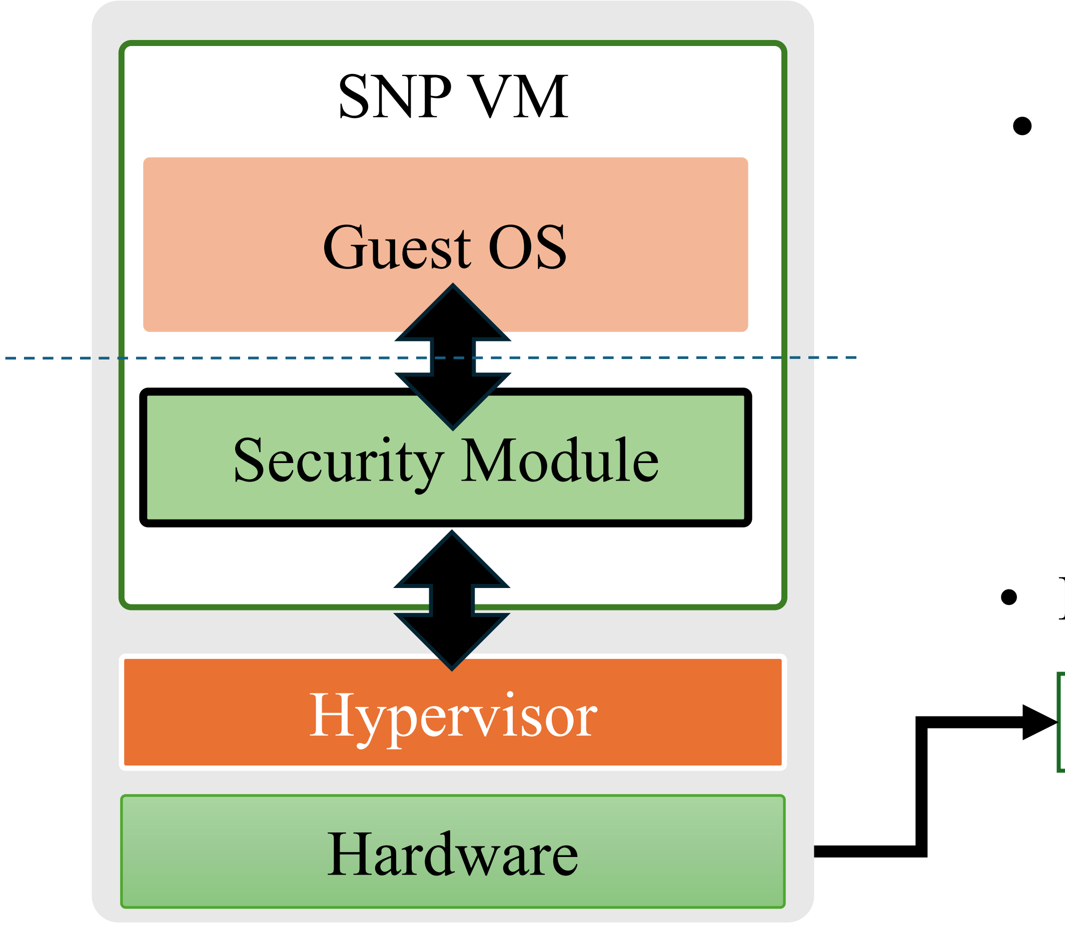
Security Property: Confidentiality + Integrity



- Hypervisor and guest cannot read sensitive data in security module

✓ Memory encryption
✓ VMPL-based memory isolation

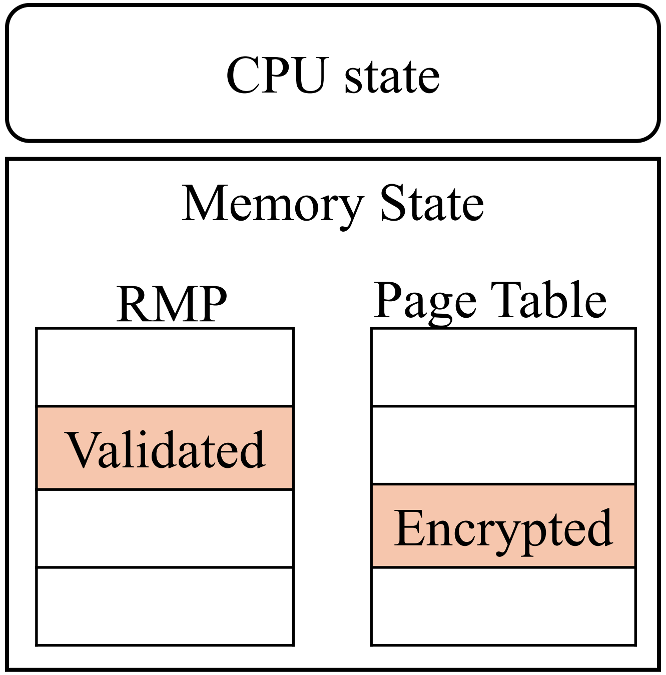
Security Property: Confidentiality + Integrity



- Hypervisor and guest cannot read sensitive data in security module
  - ✓ Memory encryption
  - ✓ VMPL-based memory isolation

- Hypervisor and guest cannot change the code/data
  - ✓ Reverse Map Table (RMP)

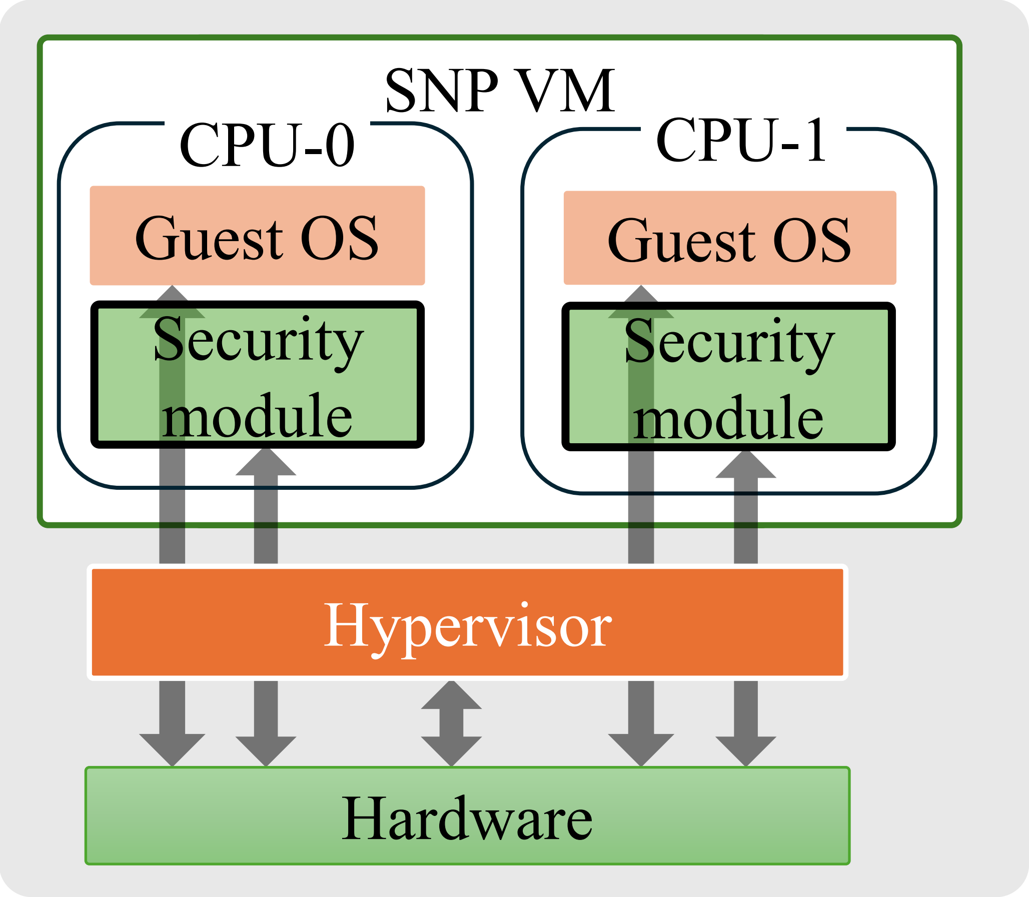
Security depends on correctness of security module



- Security module needs to
  - ✓ Validate or invalidate a page in RMP
  - ✓ Control memory encryption in page table
  - ✓ Setup guest vCPU context in VM save area page.
  - ✓ …

The correct application of those sensitive changes is critical for security.

Two types of concurrency in the security module



Multi-CPU concurrency

Multi-entity concurrency

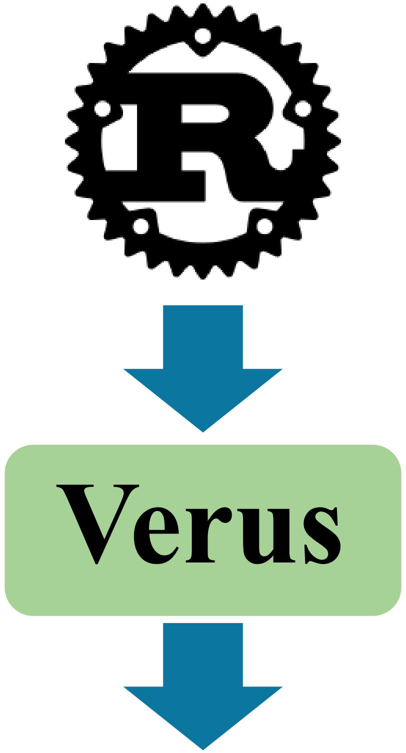
Untrusted Hypervisor and Guest OS

Existing open-sourced security modules

- AMD Linux SVSM (Secure VM Security Module)
- Coconut SVSM

- They are written in Rust but with *unsafe* Rust.
- They are **not formally verified** to be correct.

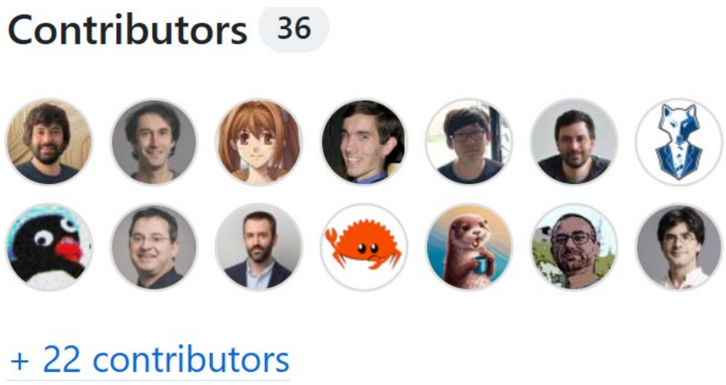
VeriSMo: A formally verified security module



Verus

```
>> verus verismo/src
verification results: xxx verified, 0 errors
```

Verus: a state-of-art verification tool



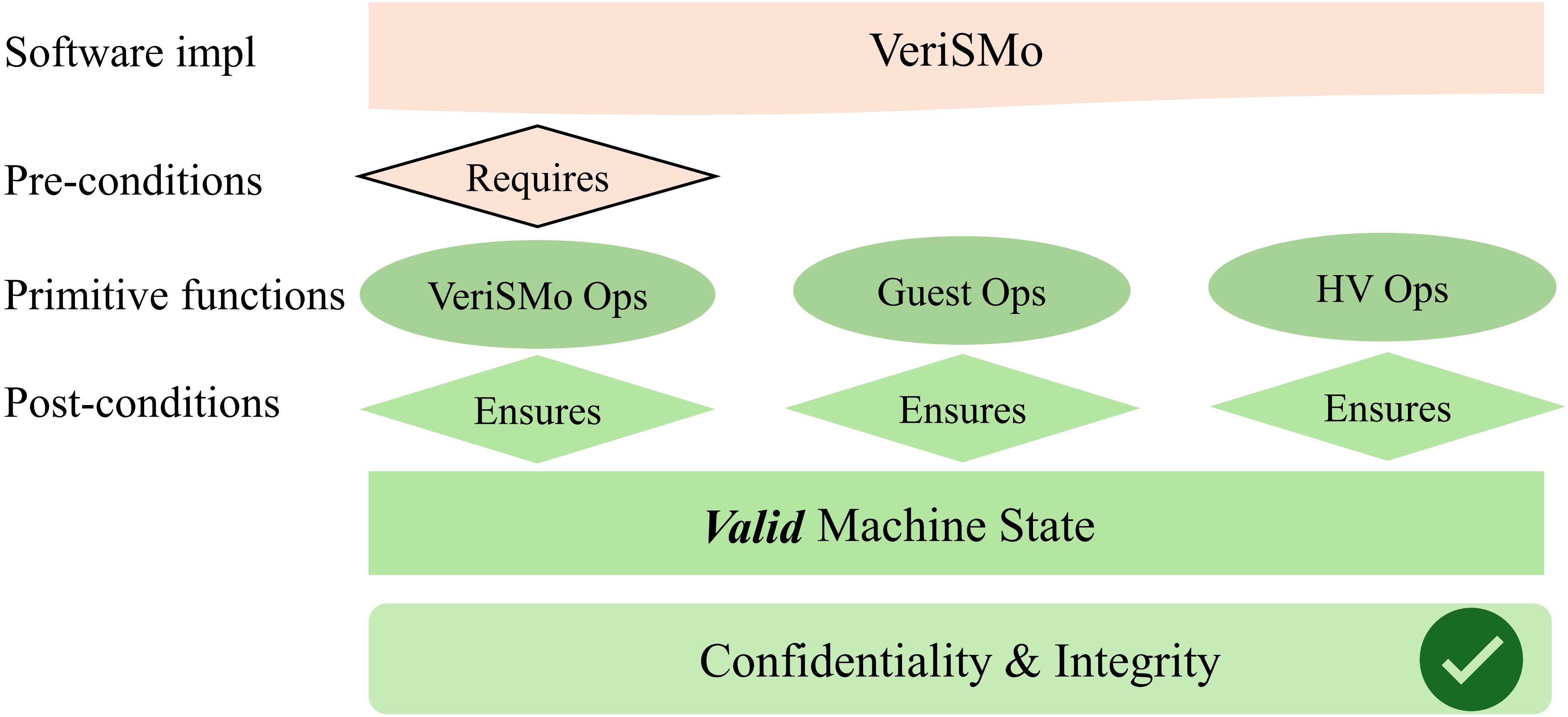
Contributors 36

+ 22 contributors

- **Rust-based verification**
  - ✓Builds on Rust ownership, borrow, and type checker.
  - ✓Ownership-based *tracked permissions* are similar to separation logic.

- **Optimized performance**
  - ✓Utilizes the SMT solver more efficiently.

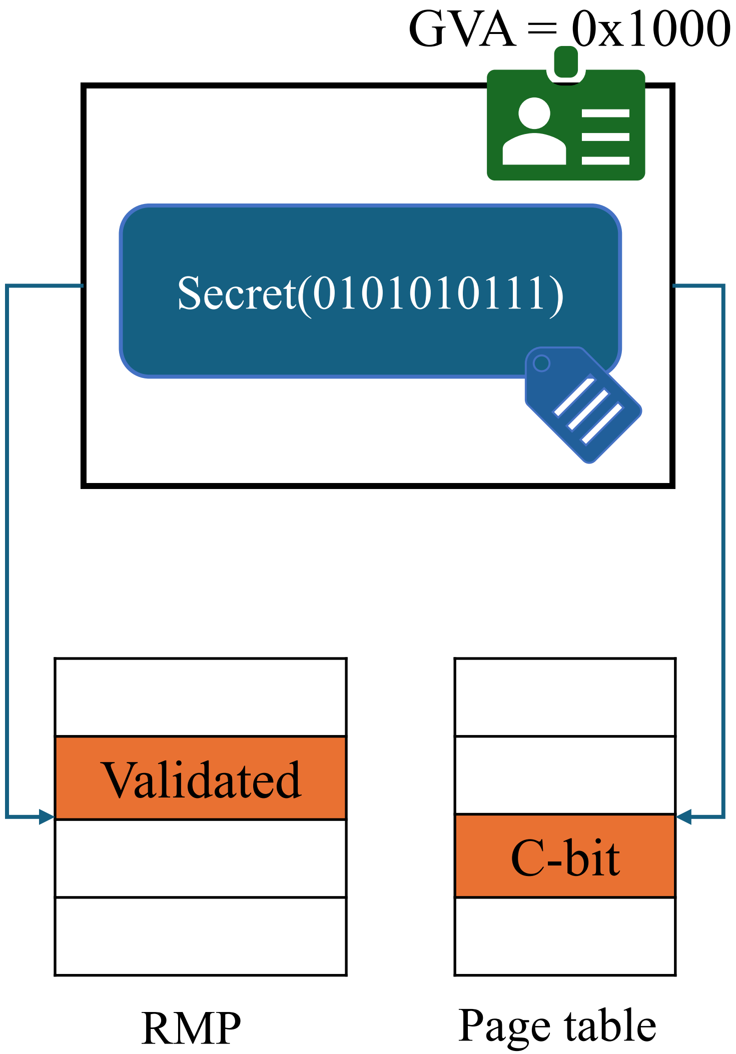
VeriSMo's verification design



Permission-based verification

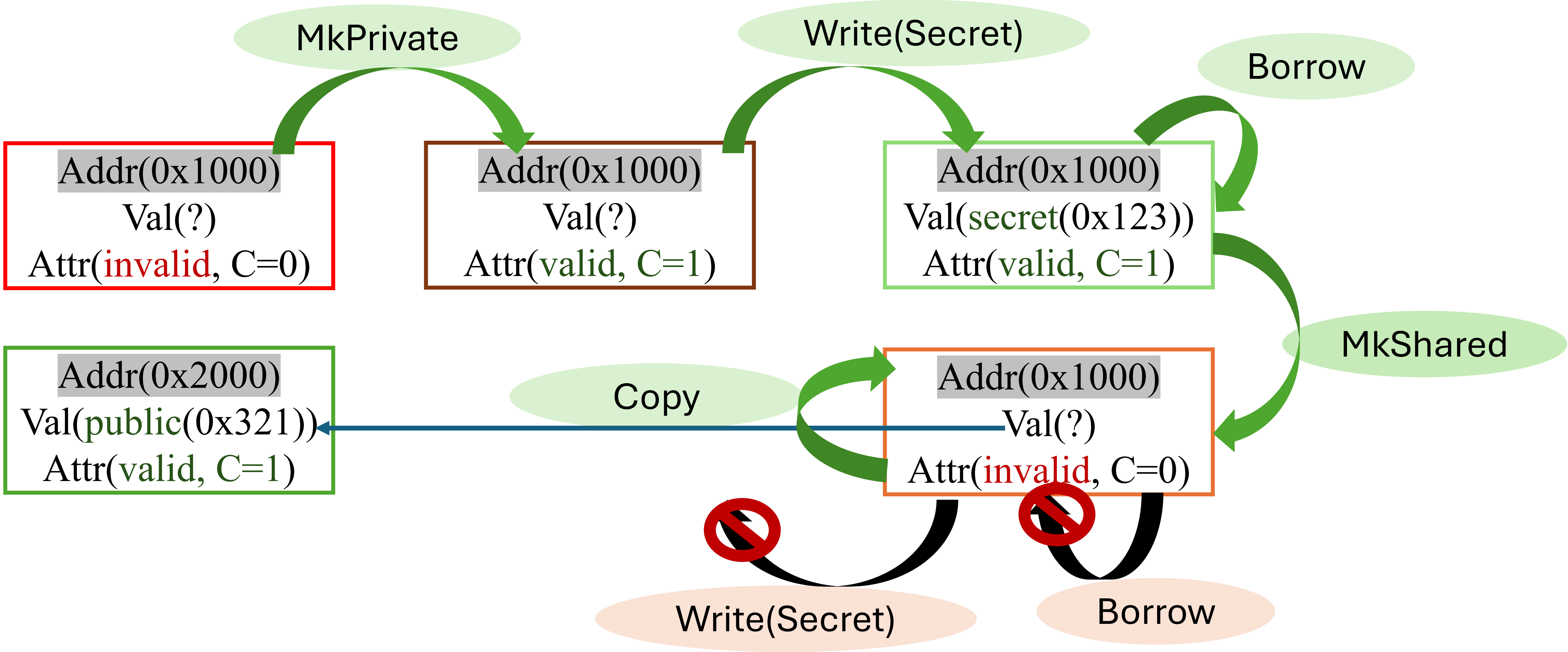
- Uses the ***tracked resource permission*** to protect raw resource access
    - ✓Raw memory
    - ✓Page table
    - ✓RMP
    - ✓Lock
    - ✓Control registers

Tracking a memory state



- Memory identity (Fixed)
  - ✓ Guest virtual address
- Memory content
  - ✓ Data
  - ✓ Security label of the data: secret/public
- Memory attributes
  - ✓ RMP entry: validated, RWX, etc.
  - ✓ Guest page table entry: encryption-bit, etc.

An example of safe access to raw memory



MkPrivate

Write(Secret)

Borrow

Addr(0x1000)
Val(?)
Attr(invalid, C=0)

Addr(0x1000)
Val(?)
Attr(valid, C=1)

Addr(0x1000)
Val(secret(0x123))
Attr(valid, C=1)

MkShared

Addr(0x2000)
Val(public(0x321))
Attr(valid, C=1)

Copy

Addr(0x1000)
Val(?)
Attr(invalid, C=0)

Write(Secret)

Borrow

Protect raw memory access in VeriSMo

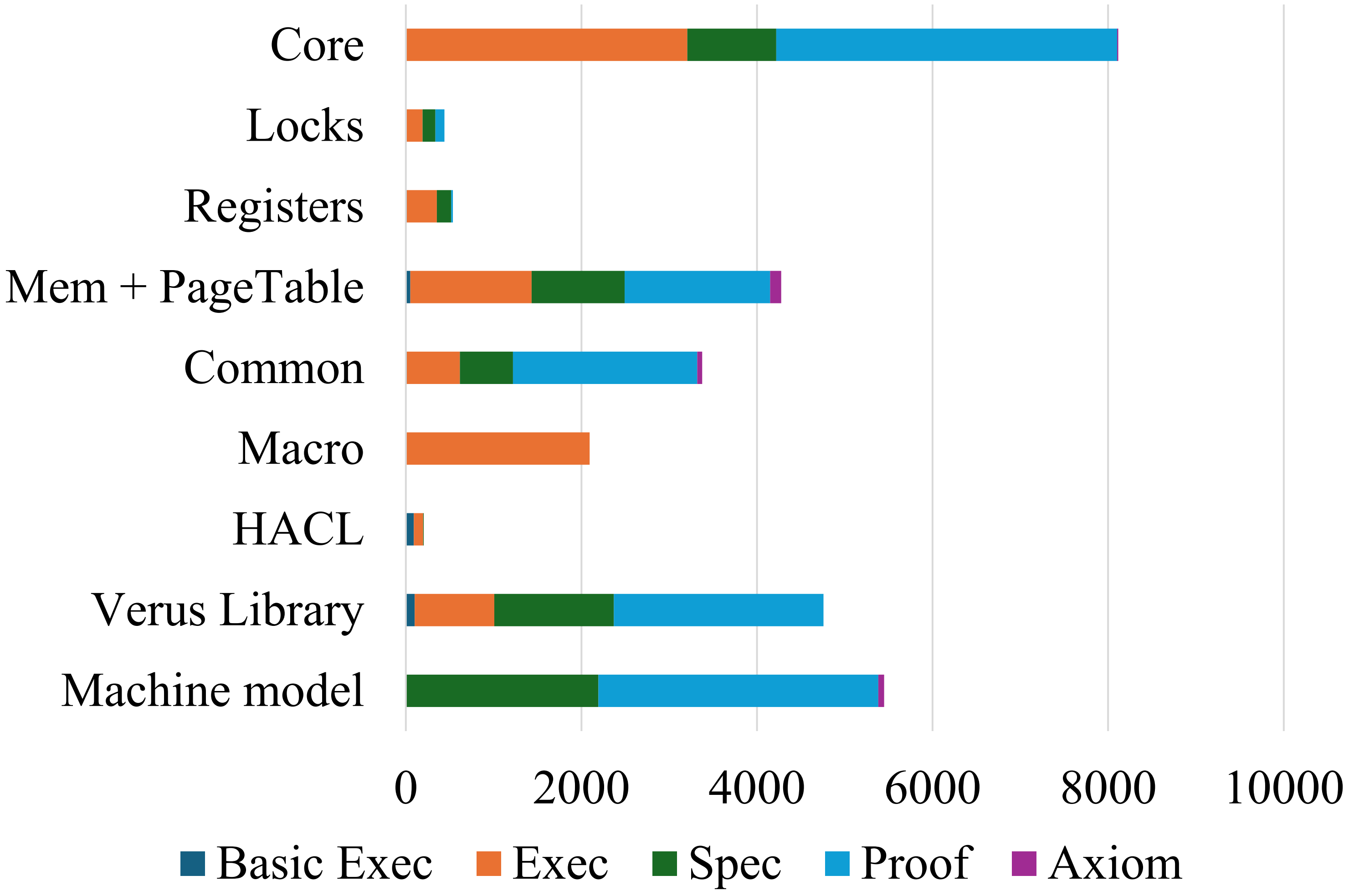
```
unsafe fn ptr_borrow<T>(
        addr: u64,
) → &T
{
        …
}
```

Protect raw memory access in VeriSMo

```
fn ptr_borrow<T>(
     addr: u64, Tracked(mperm): Tracked<&MemPerm<T>>
) → (ret: &T)
requires
    mperm.id == addr,
    mperm.attr_valid_borrow(),
ensures
    *ret == mperm.value
{
     unsafe {…}
}
```

An unforgeable object
w/o runtime overhead

VeriSMo implementation



Verification results

- Over 8k lines of executable codes
- Verified in 3 mins with 128 cores

Runtime performance

- Nearly zero performance overhead from verification

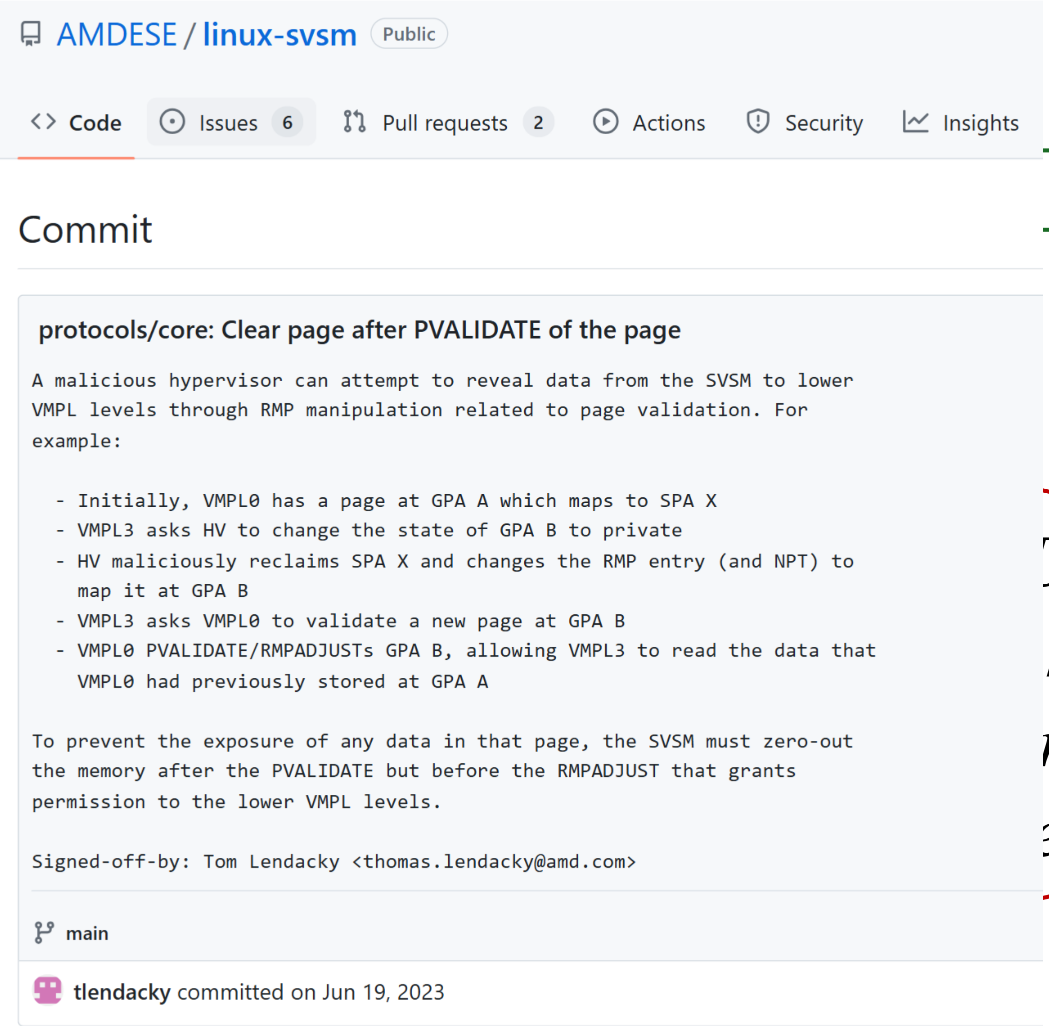
A bug in security module

```
fn mk_guest_priv(page: usize) -> bool
{
  // Reject if the page is not guest page and is not shared.
  if !is_guest_os_page(page) || !is_shared(page) {return false;}
  …
  validate_page(page, true);

  rmpadjust_page(page, rmpattr_rw);
  …
}
```

I checked that the page belongs to guest and is shared. It seems that it should not contains ***security module's secret****.*

A bug in security module

```
fn mk_guest_priv(page: usize) -> b
{
  // Reject if the page is not gue
  if !is_guest_os_page(page) || !i
  …
  validate_page(page, true);

  rmpadjust_page(page, rmpattr_rw)
  …
}
```



protocols/core: Clear page after PVALIDATE of the page

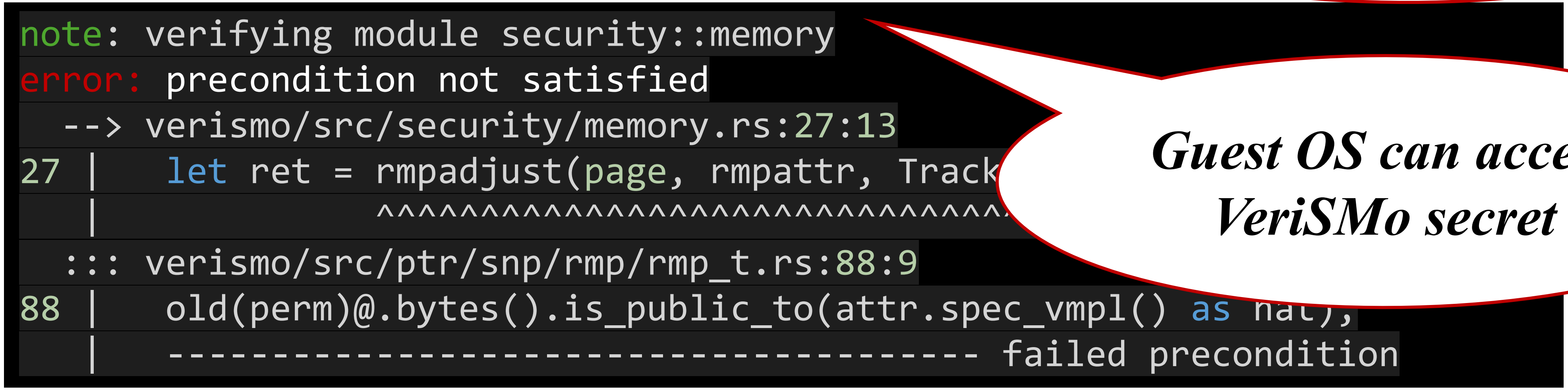
```
A malicious hypervisor can attempt to reveal data from the SVSM to lower
VMPL levels through RMP manipulation related to page validation. For
example:

  - Initially, VMPL0 has a page at GPA A which maps to SPA X
  - VMPL3 asks HV to change the state of GPA B to private
  - HV maliciously reclaims SPA X and changes the RMP entry (and NPT) to
    map it at GPA B
  - VMPL3 asks VMPL0 to validate a new page at GPA B
  - VMPL0 PVALIDATE/RMPADJUSTs GPA B, allowing VMPL3 to read the data that
    VMPL0 had previously stored at GPA A

To prevent the exposure of any data in that page, the SVSM must zero-out
the memory after the PVALIDATE but before the RMPADJUST that grants
permission to the lower VMPL levels.

Signed-off-by: Tom Lendacky <thomas.lendacky@amd.com>
```

main

tlendacky committed on Jun 19, 2023

A bug in security module

```
fn mk_guest_priv(page: usize, Tracked(mperm): Tracked<&MemPerm<T>>) -> bool
{
  // Reject if the page is not guest page or is not shared.
  if !is_guest_page(page) || !is_shared(page) {return false;}
  …
  validate_page(page, true, Tracked(mperm));

✕ rmpadjust_page(page, rmpattr_rw, Tracked(mperm));
  …
}
```



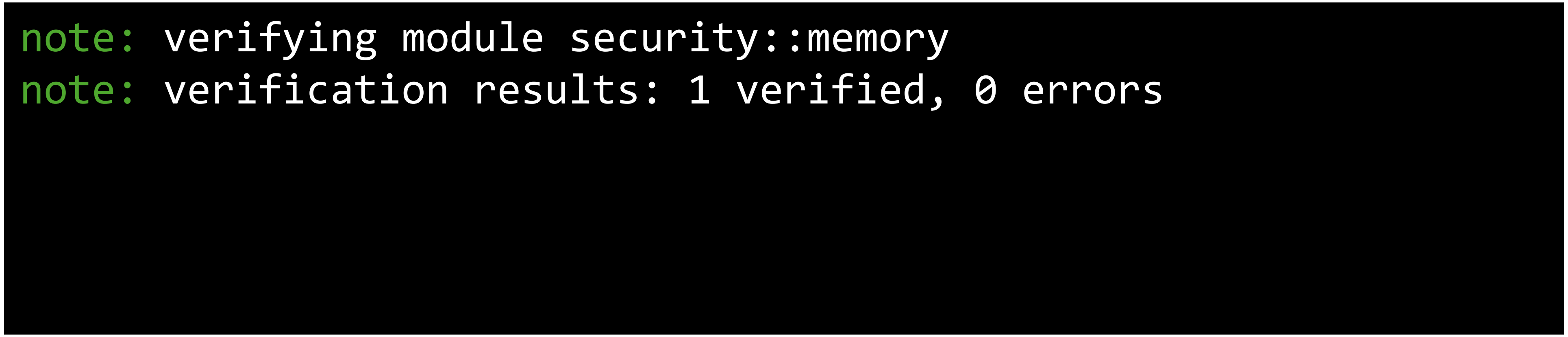
```
note: verifying module security::memory
error: precondition not satisfied
  --> verismo/src/security/memory.rs:27:13
27 |    let ret = rmpadjust(page, rmpattr, Track
   |              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  ::: verismo/src/ptr/snp/rmp/rmp_t.rs:88:9
88 |    old(perm)@.bytes().is_public_to(attr.spec_vmpl() as nat);
   |    --------------------------------------- failed precondition
```

Hypervisor sets the page mapping to a secret physical page

Guest OS can access VeriSMo secret

A bug in security module

```
fn mk_guest_priv(page: usize, Tracked(mperm): Tracked<&MemPerm<T>>) -> bool
{
  // Reject if the page is not guest page or is not shared.
  if !is_guest_page(page) || !is_shared(vpage) {return false;}
  …
  validate_page(page, true, Tracked(mperm));
  memset(page, 0, Tracked(mperm));
  rmpadjust_page(page, rmpattr_rw, Tracked(mperm));
  …
}
```



```
note: verifying module security::memory
note: verification results: 1 verified, 0 errors
```

Summary

- VeriSMo is a formally verified Rust-based security from machine model to implementation layer.
- Permission-based verification ensures correct accesses to raw resources.
- We found a security bug in existing implementations (SVSM).
- Outstanding verification and runtime performance (see paper for details).
- Code is available at https://github.com/microsoft/verismo