Capstone: An Architecture Design for Expressive Security

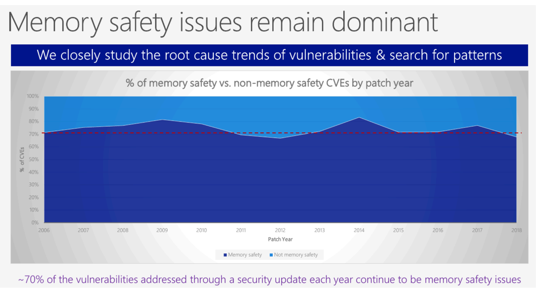
Jason Zhijingcheng Yu†, Prateek Saxena

School of Computing, National University of Singapore

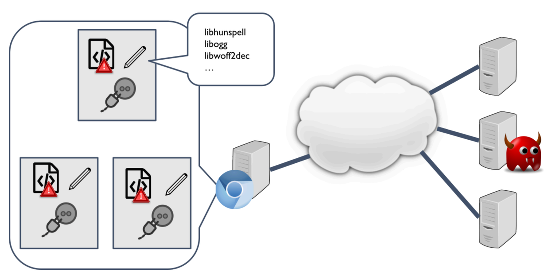
† final-year PhD student on the job market

Motivation: Patchwork of Security Extensions

Security Challenges



Memory Safety



Fine-grained Isolation

	Product Name	Vendor Name	Product Type	Number of Vulnerabilities
1	Debian Linux	Debian	OS	8161
2	Android	Google	OS	5136
3	Fedora	Fedoraproject	OS	4497
4	Ubuntu Linux	Canonical	OS	3961
5	Linux Kernel	Linux	OS	3186
6	Mac Os X	Apple	OS	3103
7	Windows 10	Microsoft	OS	3054
8	Windows Server 2016	Microsoft	OS	3042
9	Iphone Os	Apple	OS	2941
10	Chrome	Google	Application	2677

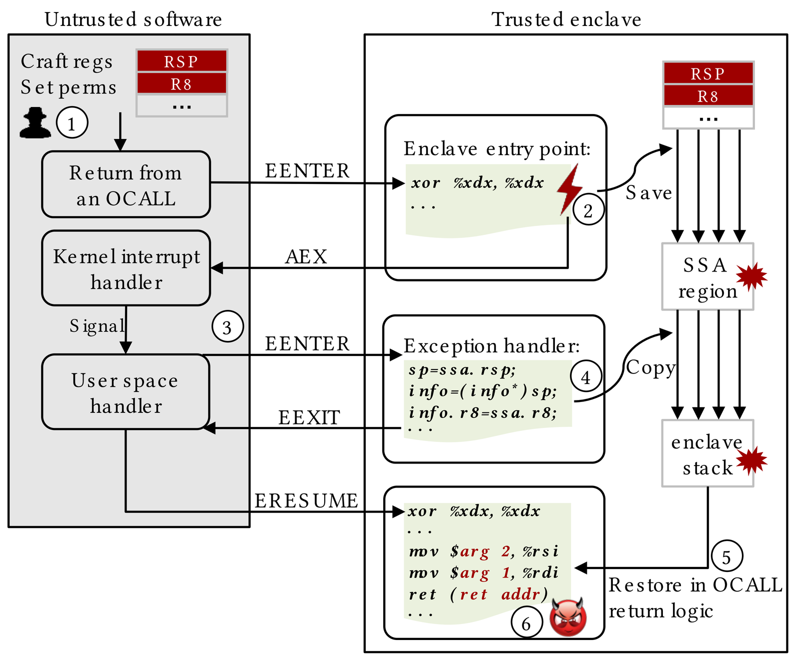
Confidential Computing

Patchwork of Security Extensions

Spatial Memory Safety	[Intel MPK, x86/64 DEP/NX][Intel MPX, RISC-V/ARM CHERI]
Temporal Memory Safety	[ARM MTE]
Concurrent Thread Safety	[Intel TSX] [ARM TME]
Intra-process Sandboxing	[Intel SGX] [Intel MPK]
Process Sandboxing	[x86/64 Privilege Rings]
Virtualization	[AMD SEV] [Intel VT-x] [Intel TDX] [ARM CCA]
Red-Green Secure Worlds	[ARM TZ] [Intel TXT]
Nested / App Virtualization	[Intel VT-x] [Intel SGX]

Problem: Compose Security Extensions?

+ Exception handling (Cui et al., 2021)

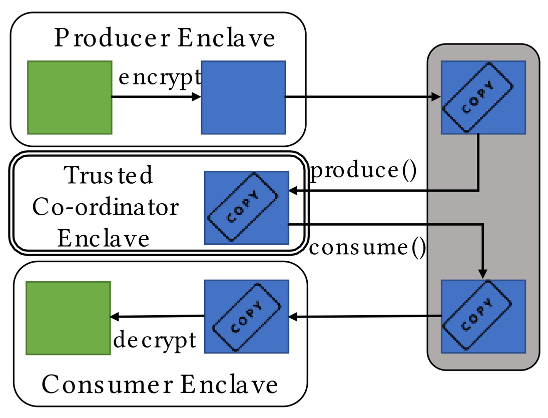


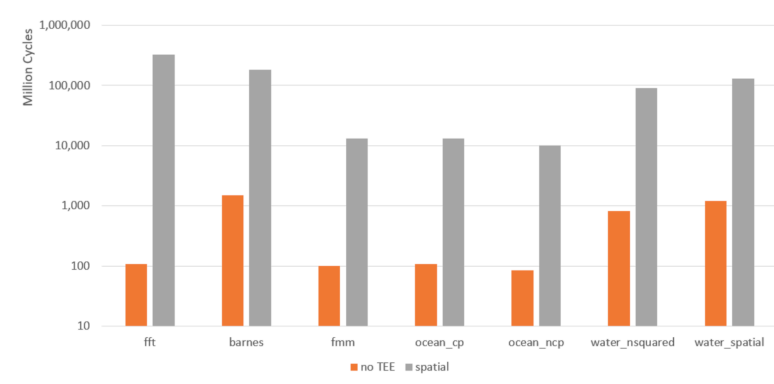
Arbitrary code execution

Affecting 9 SGX runtimes

CVE-2021-0186, CVE-2021-33767

+ Memory sharing (Yu et al., 2022)





2–3 orders of magnitude overhead

Goal

Can one design a unified foundation for multiple security goals? (Yu et al., 2023)

Our answer:

Capstone

an ISA (instruction set architecture) based on RISC-V (RV64IZicsr)

Minimal set of properties

- *P1:* **Exclusive Access**
- *P2:* **Revocable Delegation**
- *P3:* **Extensible Hierarchy**
- *P4:* **Secure Domain Switching**

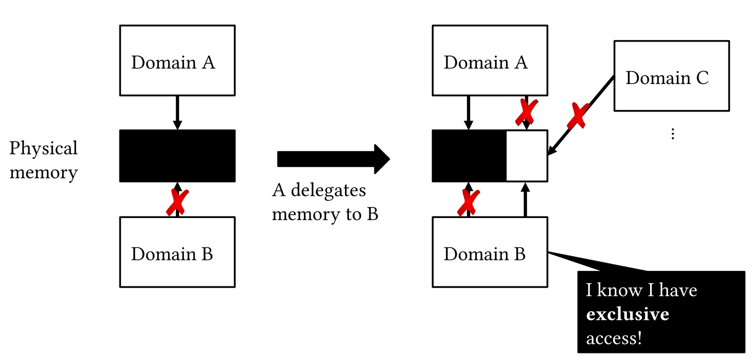
Capstone

Capstone (USENIX '23)

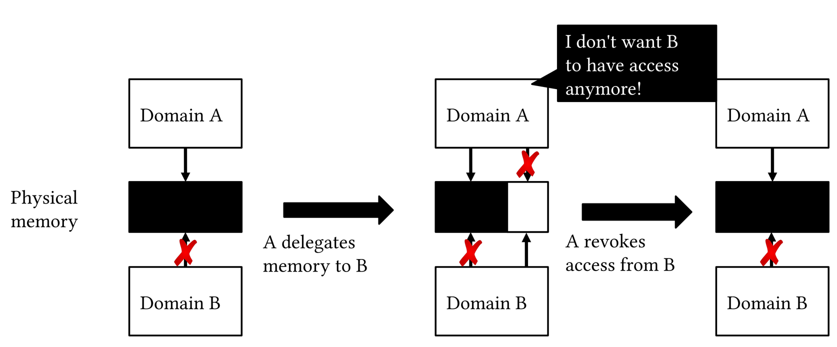
- Spatial Memory Safety
- Temporal Memory Safety
- Concurrent Thread Safety
- Intra-process Sandboxing
- Process Sandboxing
- Virtualization
- Red-Green Secure Worlds
- Nested / App Virtualization

Desired Properties in Capstone

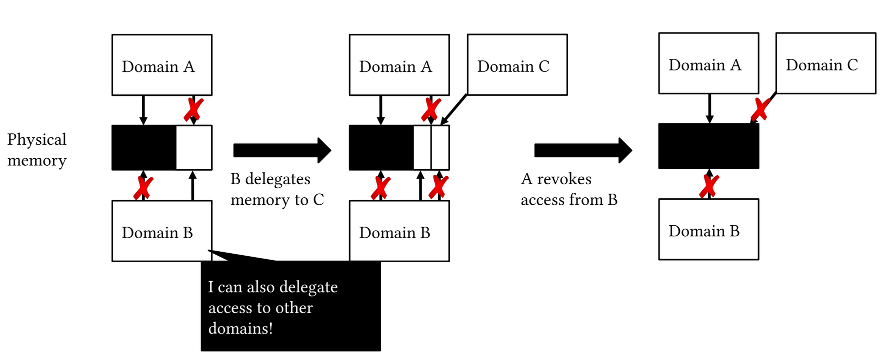
P1: Exclusive Access



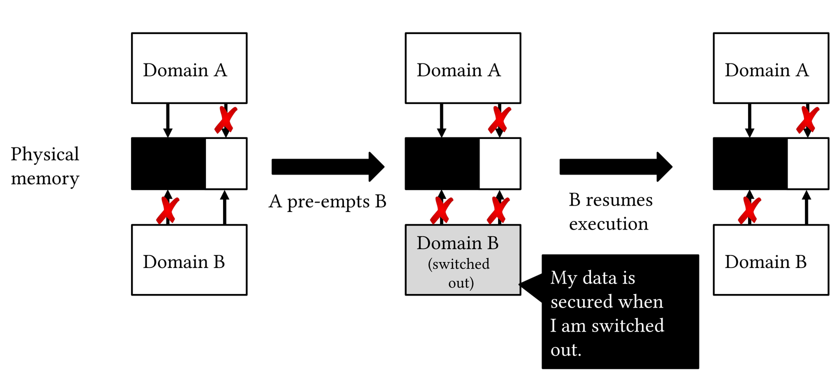
P2: Revocable Delegation



P3: Extensible Hierarchy



P4: Secure Domain Switching

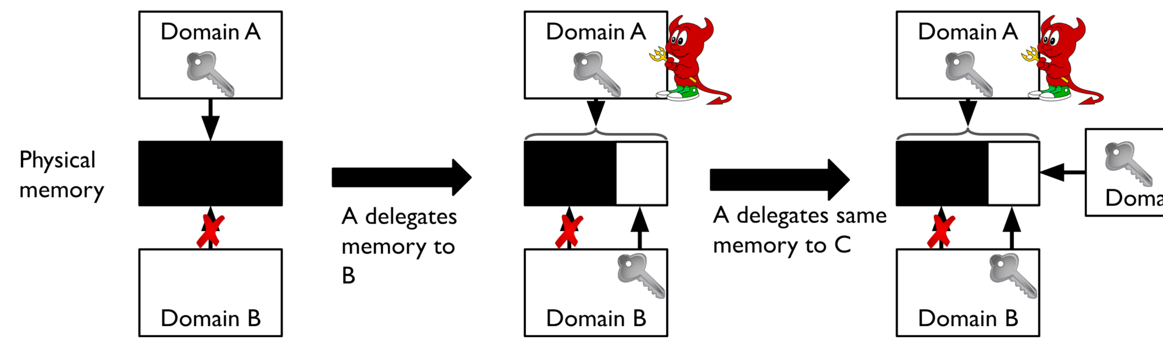


Starting Point: Hardware Capabilities

- A **Hardware Capability** is a (pointer, metadata) tuple
  - Created or modified only by querying the hardware
  - Sufficient and necessary to access the corresponding memory
  - Has the associated permissions embedded in it and is enforced by hardware
- Capability machines existed in '80s, but had challenges scaling securely

Base Capability-based Model Is Insufficient

Example: P1 (Exclusive Access) cannot be achieved

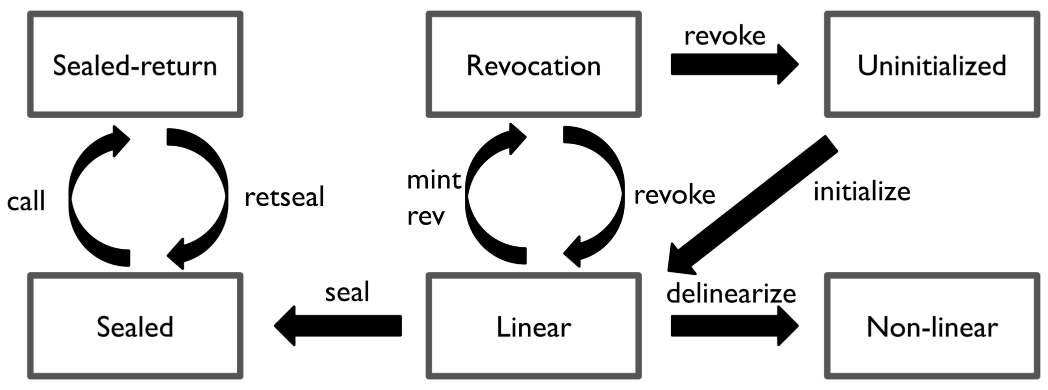


Capabilities could do more (Watson et al., 2024)

- "Continuing to refine our understanding of memory safety"
- "Pushing beyond memory safety to … software compartmentalization… for malicious programmers"
- "Exploring potential opportunities to compose … memory- and type-safe … languages, such as Rust"

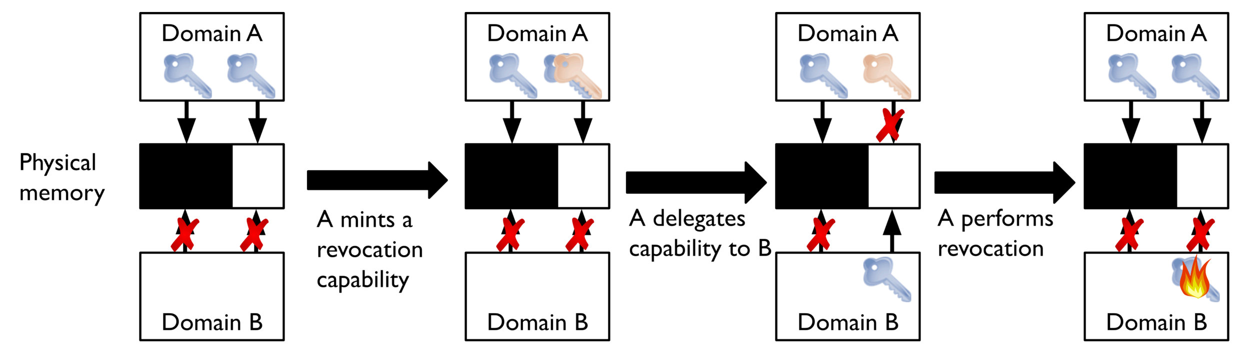
Capability-based Model in Capstone

Capability Types

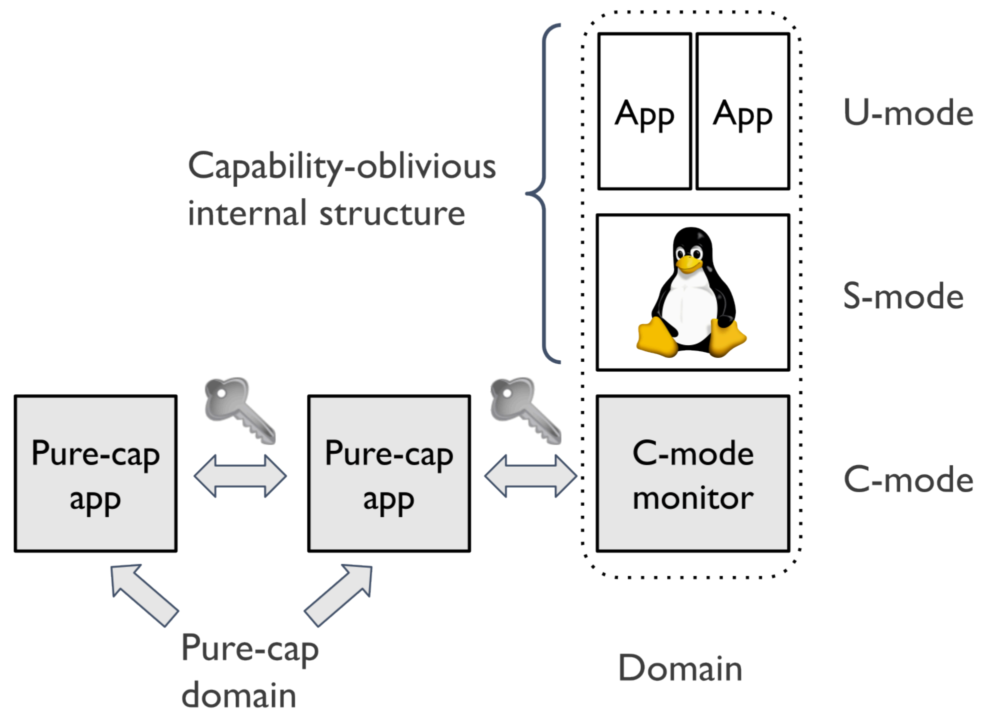


Examples

- **Linear capability:** Non-duplicable
- **Revocation capability:** A capability "snapshot", usable only for revocation



Nestable Two-Way Domain Isolation

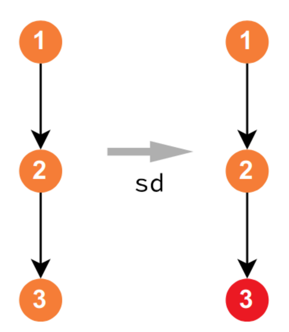


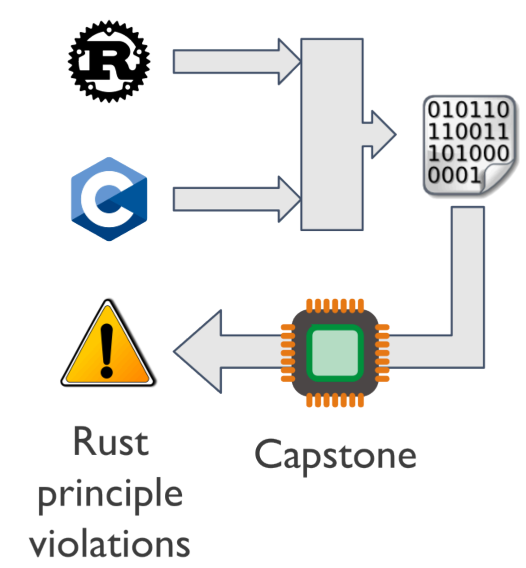
- Isolated software components in **domains**
- Domains share memory exclusively through capabilities
- A domain can be *pure-cap*, i.e., it uses capabilities explicitly for all memory accesses
- Alternatively, a domain can have an *internal structure* with a compatible *capability-oblivious* software stack in S-mode and U-mode. Memory accesses in such software ultimately translate to accesses with capabilities the domain holds

Expressive Program Safety

- **Spatial memory safety** enforced through bounds checking
- **Temporal memory safety** enforced through the revocation mechanism
- Capstone model is similar to **Rust's ownership, borrowing, and AXM principles,** and **detects their violations in mixed Rust code**

```
use std::arch::asm;
fn use_p(p : *mut u64) {
  unsafe {
    asm!(
      "sd x0, 0({addr})",
      addr = in(reg) p
    );
  }
}
fn main() {
  let mut v = Box::new(0u64);
  let v_raw = &mut *v as *mut u64;
  let v_ref = unsafe { &mut *v_raw };
  use_p(v_raw);
  *v_ref = 42;
}
```





Publications

[1] J. Cui, J. Z. Yu, S. Shinde, P. Saxena, and Z. Cai, "SmashEx: Smashing SGX Enclaves Using Exceptions," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, Virtual Event Republic of Korea: ACM, Nov. 2021, pp. 779–793. doi: 10.1145/3460120.3484821.

[2] J. Z. Yu, S. Shinde, T. E. Carlson, and P. Saxena, "Elasticlave: An Efficient Memory Model for Enclaves," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds., USENIX Association, 2022, pp. 4111–4128. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/yu-jason

[3] J. Z. Yu, C. Watt, A. Badole, T. E. Carlson, and P. Saxena, "Capstone: A Capability-based Foundation for Trustless Secure Memory Access," in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 787–804. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/yu-jason

Capstone Documentation Jason

NUS National University of Singapore