Security in Era of Persistent Memory

Naveed Ul Mustafa [1], Yuanchao Xu [2], Xipeng Shen [2] Yan Solihhin [1]

[1]*University of Central Florida, Orlando, FL* [2]North Carolina State University, NC

Introduction

Persistent Memory:

- Higher density
- Non-volatility
- Lower static power consumption
- Byte addressability
- Access latencies are not much slower

Persistent Memory Object (PMO):

- An abstraction to hold persistent pointer-rich data structures in memory without file-backing.
- Managed by OS (namespace and permissions)

Motivation

- PMO programming model allows PMO sharing over time.
- This breaks inter-process isolation, making shared PMOs a new security vulnerability.
- Previous work focuses on making unauthorized accesses to PMOs difficult *for the process accessing the PMO.*

Research Problem

A vulnerable process with legit PMO access can be used by an attacker to launch a cross-process attack on a victim via shared PMO. In doing so, a shared PMO becomes security vulnerability.

Contributions

1. Discuss security implications of PMOs.
2. Present new threat model and sample attacks stemming from PMO programming model.
3. Potential defenses against PMO based cross-process attacks.

Background

PMO Programming Interface:

- *attach()/detach()* maps/unmaps a PMO to/from address space of a user process.
- Once attached, PMO data is accessible via load/store instructions.
- *psync()* persists a modified PMO in crash-consistent way.

PMO Sharing:

- A PMO can outlive its creator process.
- A PMO can be attached by multiple processes over time.
- Simultaneous multiple readers.
- Write attach must be exclusive to other readers/writers.

Security Implications of PMOs

1. PMO corruption is persistent.
2. Relaunching a process does not erase effect of PMO corruption on it's execution.
3. PMO Corruption by one process can affect any sharing process.
4. Attacker can incrementally determine target data location for corruption across different runs.
5. Attractive target for manipulation due to pointer-rich nature of PMOs.
6. PMO accesses (via load/store) are not trapped by OS and hence not protected.

Threat Model

- Two user processes, *payload* and *victim*, share a PMO over time.
- Victim has no memory safety vulnerabilities but the payload does.
- **Adversary's goal:** Use payload to compromise victim process.
- **Adversary's knowledge:**
  1. A PMO is shared.
  2. Data structure type.
  3. PMO layout
- Trusted system software (i.e., OS).

Pointer Classification

Direction-based classification:

1. **VM2PM:** Necessary for normal operation of PMO (e.g., reading PMO data). Such pointers can be dereferenced only when PMO is attached.
2. **PM2VM:** Should not be permitted as they are not valid across process runs.
3. **PM2PM:**
   - **Intra-PMO Pointer:** Points within the same PMO and are essential to build data structures.
   - **Inter-PMO pointer:** Points to location in a different PMO. Permits simultaneous access and reduction of exposure window.

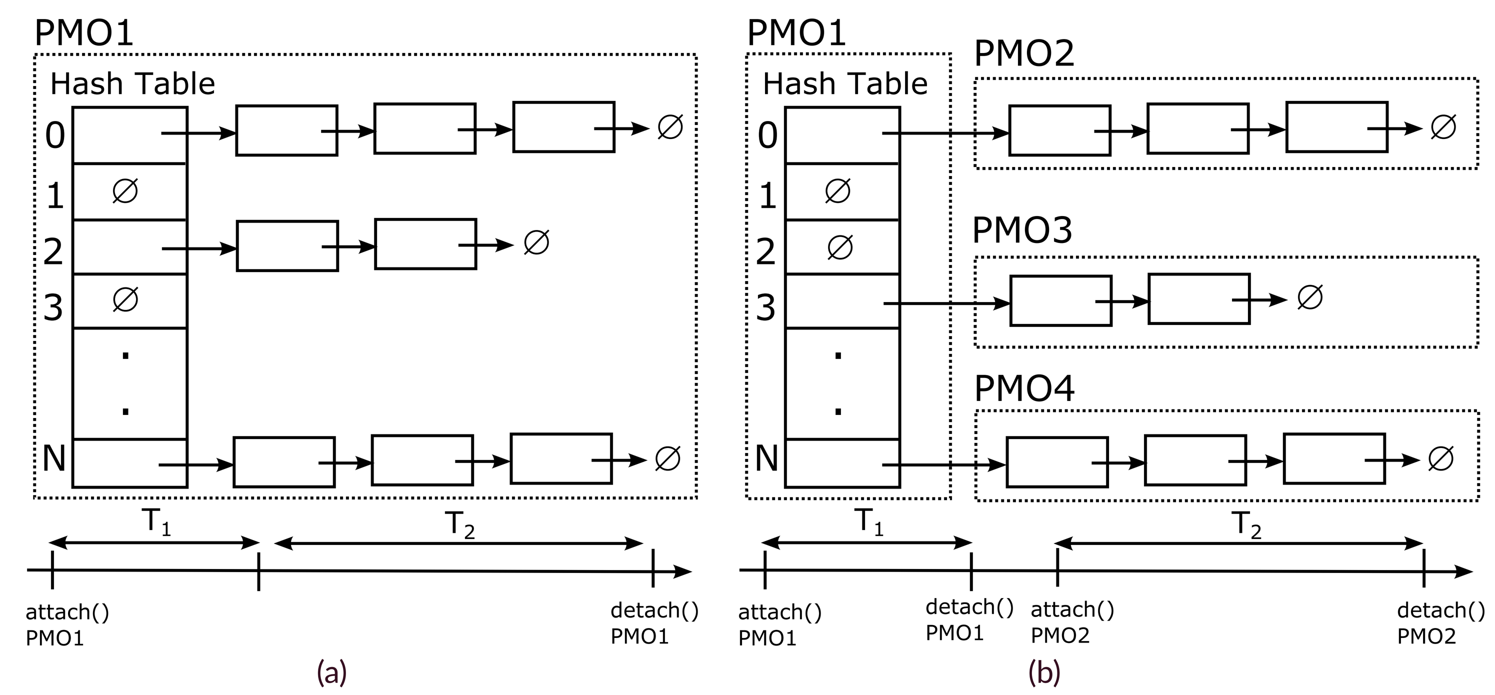

Figure 1. Hash table and linked list placed in a) same PMO and b) different PMOs.

Address-based pointer classification:

1. **Absolute pointer:** Contains virtual address
   - Fast to dereference
   - Costly space layout randomization.
   - Shared PMOs must be mapped to same virtual address range in all sharing processes.
2. **Relative Pointers:** Contains combination of PMO ID and offset
   - Translation table lookup is needed.
   - PMO relocation is less expensive.

Attack Types

Control-data attacks: alter target program's control data (e.g., return address and function pointer) to execute injected malicious code or stitched gadgets.

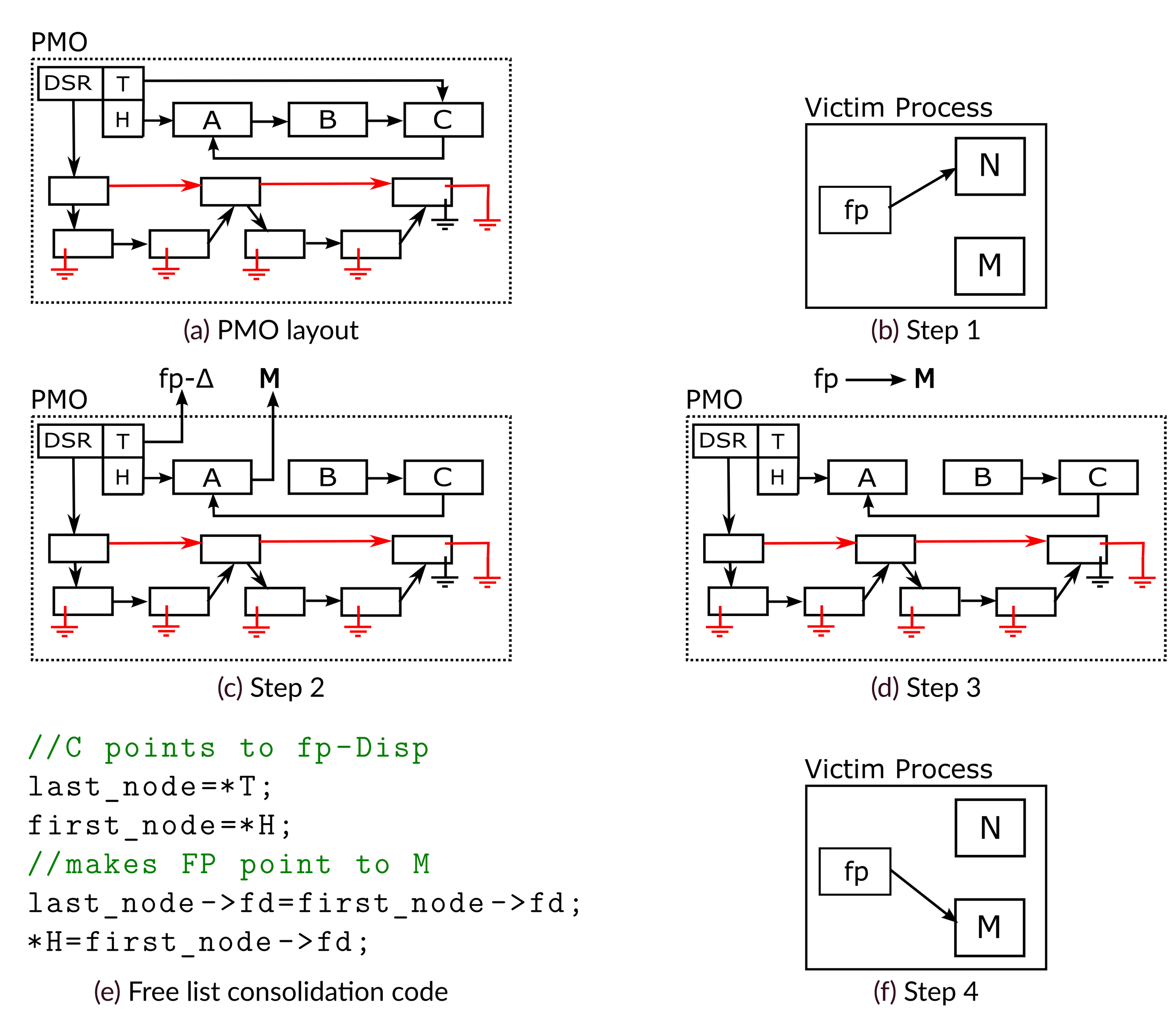

Figure 2. Steps of PMO-based cross-process/run pointer redirection attack.

Non-control-data attacks: depend on specific semantics of target application and the source code to corrupt variety of application data such as configuration data, user identification data and decision-making data.

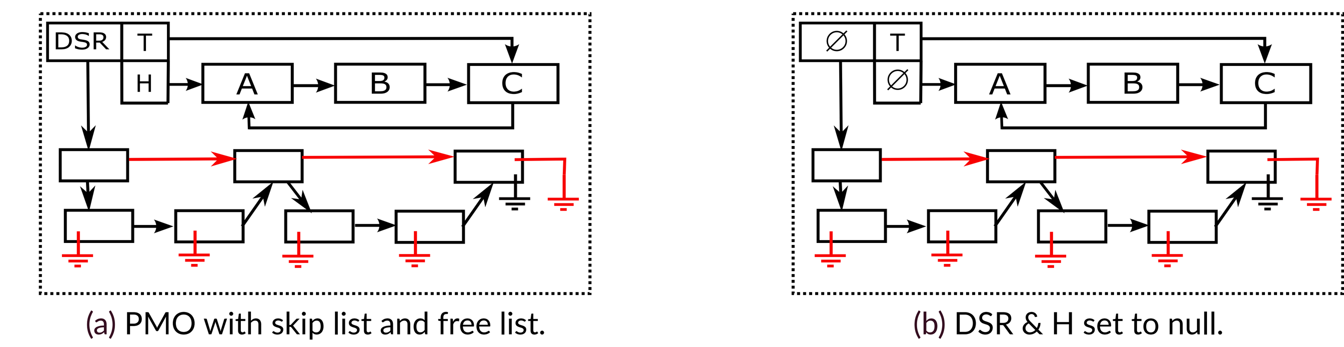

Figure 3. PMO-based cross-process/run denial of service attack.

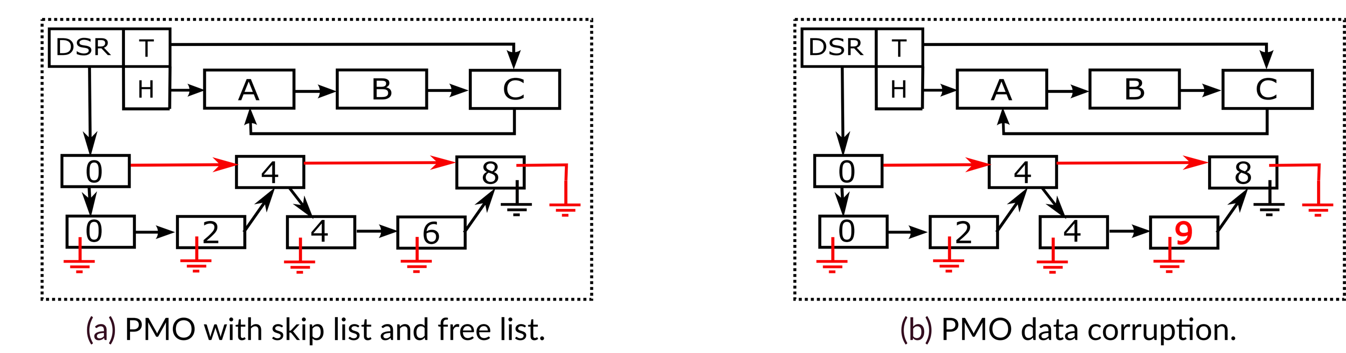

Figure 4. PMO-based attack to corrupt decision-making data.

Possible Defenses

- **PMO Space Layout Randomization (PSLR):** If enabled, randomizes the addresses (e.g., of fp and M in Fig. 2) making it hard for attacks to succeed.
- **Data Execution Prevention (DEP)** prevents code injection i.e. M in Fig. 2.
- Stronger defense is needed if PSLR or DEP are breached.

Detection and Foiling the PMO-based attacks:

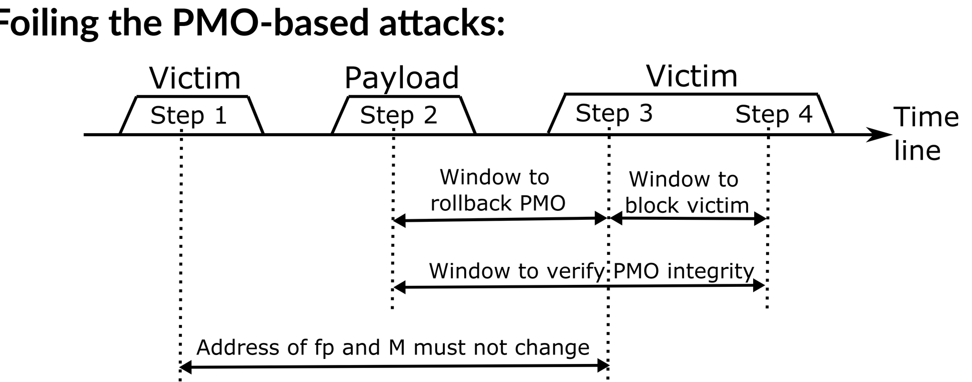

Figure 5. Opportunities for detecting and foiling an attack

Table 1. Summary of PMO attacks

Attack	Assumptions	Detection strategy
Pointer redirection to out-of-context code	No PSLR PM2NVM pointers are permitted	Topology verification
Pointer redirection to injected code	No PSLR No DEP PM2NVM pointers are permitted	Topology verification
Denial of service		Hash re-computation and comparison
Corrupting decision-making data		Data invariance checking

References

[1] Naveed Ul Mustafa, Yuanchao Xu, Xipeng Shen, and Yan Solihin. Seeds of seed: New security challenges for persistent memory. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 83–88. IEEE, 2021.

Acknowledgement

This work is supported in part by ONR through award N00014-20-1-2750 and NSF through award CNS-1717425.