Secure Object Stores (SOS): Non-Volatile Memory Architecture for Secure Computing

Rodel Felipe Miguel*, Sivaraman Sundaram, K. M. M. Aung
Data Storage Institute, A*STAR (Agency for Science, Technology and Research), DSI Building, 5 Engineering Drive 1, Singapore 117608.

* Corresponding author. Email: rodel_fm@dsi.a-star.edu.sg
Manuscript submitted October 7, 2014; accepted March 10, 2015.
doi: 10.17706/jcp.11.3.189-194

Abstract: Next generation non-volatile memory (NVM) technologies will change the design of major operating system components and how applications will be written because it deviates from the volatility and capacity assumptions of primary memory in the conventional computer systems design. A program's persistent data and run-time objects that contain sensitive information and without proper security mechanisms in place, it exposes to critical attacks. In this paper, we will introduce Secure Object Stores (SOS), which is an object-based mechanism to access the NVM. We also illustrate different use-cases that can take advantage of the SOS.

Key words: Secure object stores, non-volatile memory, memory management.

1. Introduction

New promising, main memory technology (NVM), non-volatile memories such as PCM, STT-MRAM, memristor etc. are not only scalable in-terms of memory density, but are also less power consuming and most importantly persistent. This allows us to perceive this technology in 2 different ways. Firstly, we can view it as a direct way to achieve persistence to in-memory data structures accessible directly to programmers. Secondly, as a byte addressable storage class memory, accessible through a file system interface allowing discretionary access control or mandatory access control (e.g. SELinux, AppArmor).

Our technology focuses on providing a clean and consistent object based interface to program objects, files and I/O. This allows us to view memory data structures, files etc as application accessible objects, while providing common access control mechanism for them.

SOS addresses two challenges: (1) what could be the logical architecture when NVM acts as both primary and secondary memory, and (2) how to provide consistent access control mechanisms for program objects and files.

The objective of this paper is to discuss the challenges and opportunities that the new NVM technology brings concerning security. More specifically, we would like to propose an architecture that will address the access control and privacy of data stored in the NVM. The next section will discuss the background technologies in SOS. We will discuss the motivation of the design, NVM, security issues, and the current state-of-the-art. Section 3 will discuss the details of the design of SOS. Section 4 will list the advantages of SOS. In section 5, we will compare SOS with existing memory management solutions for NVM and we will discuss why it is important to use SOS. Finally, we will conclude the paper by summarizing the contributions and novelty of the SOS.

2. Related Works

There have been many recent systems that have come-up that provide memory transactional and file-based access control to non-volatile memory that makes use of its byte-addressability.

Mnemosyne [1] exposes NVM as persistent memory abstraction, and makes it simple for programmers to declare and use persistent data. This is done by providing low-level programming primitives as well as system call APIs to programmers. This allows programmers to persist data structures without having the need to serialize them for storage. Further, Mnemosyne [1] provides durability, atomicity and consistency to memory transactions. Further, it provides implicit access control by making use of a backing file for dynamic and static regions of processes and a persistent mapping table consisting of NVM frame number and page offset information.

NVHeaps [2], similar to Mnemosyne, provides programmers with abstractions to create persistent objects, pointers etc. Further, NVHeaps provides consistent, atomic transactions on persistent objects and pointer safety by preventing programmers from misusing them and hence corrupting the data. Objects are accessed through file system like interfaces to open, close, rename, copy and transmit. Access control for NVHeaps is done through the use of files. Set of NVHeap objects contain file references that provides naming and access control.

BPFS [3] and SCMFS [4] implement file systems for NVMs. BPFS file system for NVM makes use of a tree-based hierarchical pointers. By using 64-bit pointers to data blocks or pointer blocks the system allows fast and reliable access to file data and also offers mechanisms to provide transactional consistency. As BPFS is a file system it uses file based access controls at the operating system layer.

SCMFS [4] is also a file system for non-volatile memories attached to the memory bus of the system. It uses allocation system calls such as nvmalloc and nvfree to manage file system allocations / de-allocations at the kernel level. In a way the system is similar to memory file systems available with the Linux kernel such as tmpfs and ramfs. As in the case of all file systems SCMFS access control is discretionary.

CDDS [5] describes a consistent and durable data structure versioning system, which can be used to create storage systems without the need to do logging or shadow paging to move non-volatile data from one consistent state to another. CDDS can be used by storage systems such as file-systems, key-value stores etc. for consistent data updates, fast access with limited data copying.

Essentially systems such as [1], [2], [6] exposes non-volatile memory through programming primitives for creating pointers, objects, data structures and make them persistent. Each varies in the way it manages consistency, durability, atomicity etc. Access control in these systems are either implicit through the use of processor protection boundaries [1] or file based access control [2].

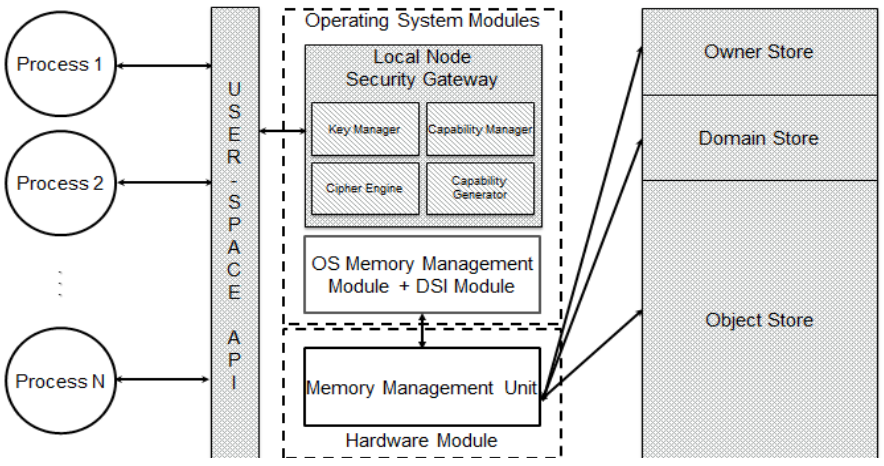

Fig. 1. Secure object stores architecture.

3. A Proposed: Secure Object Stores

The logical architecture of SOS is illustrated in Fig. 1. SOS has 2 key features for NVM acting as both primary and secondary memory. First, the NVM is divided into three logical regions: Owners, Domains, and Objects. The Owners region stores the list of owners with unique names. Each Owner data structure contains metadata for all the Domains it owns. Domain region stores domain information created by various applications run by various Owners. Domain data structure is a collection of Object metadata which belongs to that particular domain. Object region is the main repository of objects needed by applications. Operating System Memory Management Module and the processor MMU together provide for the management of these regions.

The second key feature/component of SOS is the Local Node Security Gateway (LNSG). This component is an additional module to the operating system kernel. The LNSG contains 4 sub-modules:

1) Capability Manager: Maintains a list of capabilities for specific processes. Essentially Capability Manager manages the owner, domain and object regions as well as integrates other parts of the system such as capability generator, key manager etc. to generate tokens, store tokens and authenticate tokens. Capability tokens and application capability list are stored under a capability domain created by the Capability Manager.
2) Capability Generator: Generates unique tokens based on object / domain name, physical location addresses, cipher information and access rights. Further maps capabilities to offset from start address of the Domain or Object region
3) Encryption Engine: The encryption engine provides encryption/decryption for domain capabilities, object capabilities and objects. Internally contains standard cipher modules such as random number generator, message digest and so on.
4) Key Manager: Generates keys for the encryption engine. It also acts as repository of keys for symmetric encryption and manages the lifecycle of keys. Essentially it is the key life cycle manager. All Keys are further stored under Key Manager owned domain on the NVM.

3.1. Process Flows and Secure NVM API's

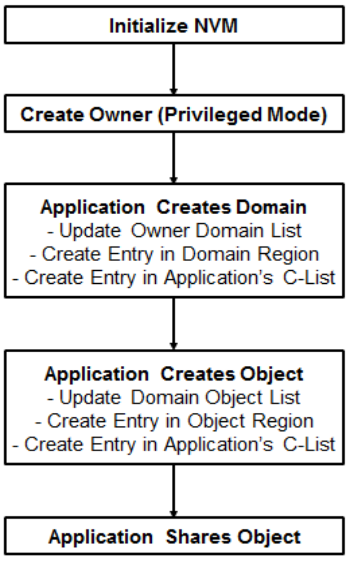
Initialize NVM
Create Owner (Privileged Mode)
Application Creates Domain
- Update Owner Domain List
- Create Entry in Domain Region
- Create Entry in Application's C-List
Application Creates Object
- Update Domain Object List
- Create Entry in Object Region
- Create Entry in Application's C-List
Application Shares Object

Fig. 2. Generic process flow of SOS.

Fig. 2 above shows a high level chronology of steps that need to be done for an application to create and share objects. First, during system initialization, for e.g. during installation of the operating system, the capability manager and key manager components of the LNSG create their corresponding domains and objects. The capability manager creates a capability domain under which it creates a capability object. The capability object can be used as a key-value or an index store containing the application id as key and capability lists as values.

An application ID is a unique number assigned to each application installed on the system. This application ID contains a capability list (C-List). C-List is a data-structure containing all active capability tokens for the application. Essentially the capability object is a secure container that is used to persist access control information for installed applications.

Similar to capability manager, the key manager also creates a key domain and key object(s) on the NVM upon initialization. The key manager uses these key object(s) as repository for keys used for encrypting meta-data, data or both.

Further, for every user created in the system an owner entry is created in the owner region of the NVM using the create owner API.

As can be seen from the flow charts, the application that creates a domain or an object initially owns the full access rights to that domain. The access rights being Read, Write, Delete, and Share. Before proceeding further let us take a look into each of the above privileges and what they mean in the context of domain and object.

Table 1. Domain and Object Privilege List

Context	Read	Write	Delete	Share
Domain	Allowed to open the domain and access capability tokens for objects within	Allowed to open the domain and create objects within the domain	Allowed to delete the domain and individual objects within	Allowed to share the domain and individual objects within to other applications
Object	Allowed to read objects	Allowed to write objects	Allowed to delete objects alone	Allowed to share the objects

4. Advantages of SOS

In this section we highlight the advantages of our architecture for managing non-volatile memory data, which provides fast and efficient access-control capabilities at the operating system layer, over existing systems.

SOS' architecture makes memory protection more efficient and robust than page boundaries offered by traditional memory protection mechanism. Owners, Objects, and Domains are the primary memory protection boundary. This makes the isolation of data clearer and makes access control a seamless security feature that can be integrated. Additionally, Domains allow administrators or system users to provide explicit access control mechanisms to the system. Furthermore, Objects or all Objects under a Domain can be encrypted. Encrypting the Objects will protect the critical or sensitive information of the Owners. It can be allowed by developing relevant API's or system calls to support Object encryption.

SOS is designed for scalability. Object stores using capability tokens and a namespace allow easy expansion of a local store to global/distributed stores, e.g. IMC. Capability tokens can be easily expanded to encode compute/data node identifier along with others such as object/domain name, physical address, cipher information and access rights. Such tokens can have a global reference, which can be used for distributed in-memory data stores.

SOS is portable. Objects provide sufficient abstraction to allow any mechanism to be used to store and access them. For example, Objects can be stored as a byte addressable data blocks with pointers addressing them, which is similar to BPFS. The logical architecture of SOS can be applied to any programming language to support storage of Objects managed/maintained by the applications written in that programming language, like Mnemosyne. Objects can be stored using journaling based systems like CDDS, providing transactional properties such as consistency, durability, and atomicity.

5. Conclusion

Our technology provides methodologies to organize, provide access control and memory protection boundaries for data created and stored in non-volatile memories. We achieve this end by considering non-volatile memories, sitting on the memory bus, to be primary as well as secondary memory. An application programmer can create objects within the non-volatile region and make use of such objects to store application data, file data etc. while at the same time consider them as named entities upon which access control mechanisms can be implemented. With such a simple organization of NVM data, we can achieve multiple objectives. One, application programmers can be provided with a clean and consistent interface for creating, writing and accessing data. Second, users can also apply security mechanisms such as discretionary access control, capability based access control etc. This will ensure that right users / applications accesses the persistent data even after system reboots. Finally, processes storing application data on NVM can ensure memory protection during execution through domain boundaries.

Acknowledgement

This work is supported by A*STAR's Future Data Centre Technologies (FDCT) grant, the project entitled "Secured Large Scale Shared Storage System" (Grant No. 1122804006).

References

[1] Volos, H., Tack, A. J., & Swift, M. M. (2011). Mnemosyne: Lightweight persistent memory. *Proceedings of the 16th International Conference on Architectural support for Programming Languages and Operating Systems*. New York, NY, USA. ACM.

[2] Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R., & Swanson, S. (2011, March 5). NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems.* Newport Beach, CA, USA.

[3] Condit, J., *et al.* (2009, October 11-14). Better I/O through byte-addressable, persistent memory. *Proceedings of the ACM SIGOPS 22nd Symposium on Operating systems Principles.* Big Sky, MT, USA.

[4] Wu, X. J., & Reddy, A. L. (2011, November 12-18). SCMFS: A file system for storage class memory. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. Seattle, WA, USA.

[5] Venkataraman, S., Tolia, N., Ranganathan, P., & Campbell, R. H. (2011, February 15-17). Consistent and durable data structures for non-volatile byte-addressable memory. *FAST'11 Proceedings of the 9th USENIX Conference on File and Storage Technologies* (pp. 61-75). San Jose, California, USA.

[6] Lowell, D. E., & Chen, P. M. (1997, October). Free transactions with Rio vista*. Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles* (pp. 92-101). Saint-Malo, France.

Rodel Felipe Miguel hails from Laguna, Philippines. He received his bachelor of science degree in computer engineering (cum laude) from AMA Computer College, Makati City, Philippines.

He has more than 12 years of software engineering experience focused on embedded systems, device drivers, and network applications. He led teams from the Philippines and Singapore and has worked for industry leaders like Agilent Technologies and JDSU. He is currently a senior research engineer in Data Storage Institute — A*STAR, Singapore. His research interests include storage and network security.

Sundaram Sivaraman received his B.E. degree in engineering (1st Class Honors) from Anna University Chennai and M.Sc. degree from Nanyang Technological University, Singapore. He is currently a research engineer in Data Storage Institute, Singapore. His research interests include storage and network security.

Khin M. M. Aung received a PhD degree of computer engineering from Korea Aerospace University, in 2006. She is currently a scientist with A*STAR, Data Storage Institute, Singapore. Her research interests include data center and network storage technologies.