Storage Networking Industry Association
Technical White Paper

Persistent Memory Hardware Threat Model v1

July 26, 2018

ABSTRACT: This white paper discusses approaches for securing persistent memory (PM); particularly considering unique characteristics of PM. This work includes a threat model and potential responses to threats.

USAGE

DISCLAIMER

The information contained in this publication is subject to change without notice. The SNIA makes no warranty of any kind with regard to this specification, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The SNIA shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this specification.

Suggestions for revisions should be directed to http://www.snia.org/feedback/.

Table of Contents

1 Scope and relationship to NVM Programming Model

This white paper discusses approaches for securing persistent memory (PM); particularly considering unique characteristics of PM. This work includes a threat model, potential responses to threats and recommended security requirements for PM.

Modern IT is generally segregated into private and public cloud infrastructure. For simplicity in this document we will treat traditional private infrastructure and private cloud together. PM can appear in both public and private cloud deployments. Hybrid cloud is not treated separately here as it includes both public and private use cases depending on the infrastructure in use by particular applications. While most threats to PM security are common across public and private cloud, there are a few notable distinctions.

Public cloud infrastructure is maintained and administered by cloud providers and shared by many independent customers. Security measures that may not be necessary in private cloud environments may be essential to protect each customer's data from all other customers, and from the operators of the cloud data center itself. Customers depend on cloud datacenter security measures to make it easier for them to trust the cloud. Unfortunately this dependency is difficult for a customer to validate and may not always be fulfilled. Security concerns may drive customers towards private cloud infrastructure even in cases where public cloud is more cost effective.

Private cloud infrastructure deployments generally support fewer consumers than public clouds. This can enable private clouds to secure infrastructure in ways that are not as feasible in larger scale public cloud use cases. Additional security can be valuable even through private cloud consumers are generally different parts of the same corporation.

2 Multi-Tenancy models

The threat model and requirements developed within this document extend into the context of a multi-tenant public cloud data center. In this section, public and private cloud multi-tenancy are explored separately, although solutions to security requirements may cross over between the two.

2.1 Public Cloud Datacenter Multi-Tenancy

There is always more than one party involved in cloud computing or storage. Multiple customers are sharing infrastructure, which is itself managed by cloud providers. Threats originate from all parties, including the provider and physical infrastructure itself. Figure 1 shows one of many customers using a cloud datacenter.

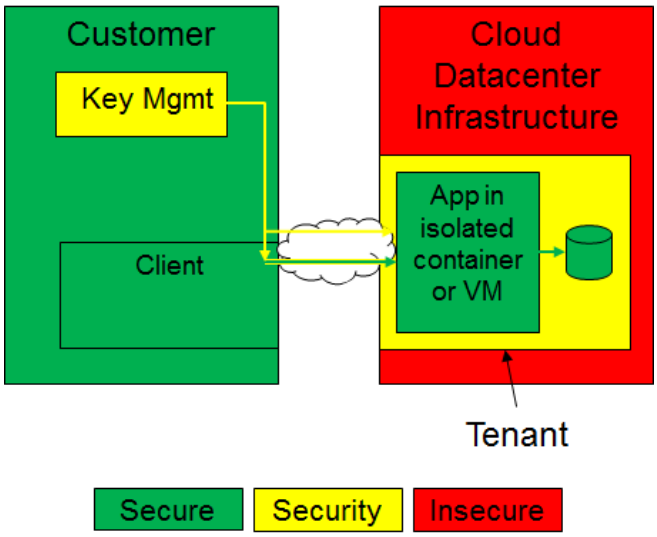

Figure 1 – Cloud datacenter model

The security principal here is the cloud datacenter customer whose data is being stored and manipulated within the cloud datacenter. A given customer can at best trust a subset of the infrastructure (hardware and software) involved in this model. Trusted environments are illustrated in green while untrusted environments are red. The security infrastructure that enables the customer to trust part of the cloud datacenter is illustrated in yellow. Customers always have some infrastructure of their own in order to access the resources of the cloud. Customers must institute security practices such as physical security as well as user and administrator authentication (e.g. passwords) and authorization (e.g. permissions) within their own environment. Most importantly to this model, customers operate or have access to a trusted secure key repository and distribution infrastructure. Customer trusted key management is the basis for maintaining trust outside of the customer's own infrastructure.

In general, cloud datacenter infrastructure is not trusted by customers. To compensate, cloud datacenters must provide isolated containers or virtual machines (VMs) that enable customers to become tenants (temporary residents) of the cloud datacenter. To the extent that customer data outlives the isolated container or VM, the cloud datacenter must also provide durable storage capacity. The contents of both the isolated container and the storage should only be accessed using keys that originate from the customer's secure key management facility.

Customers become tenants of a cloud datacenter through creation of a contract and establishment of identity. Customers authenticate to the cloud datacenter, which provides for their use of keys to access datacenter resources. The tenant identity allows the cloud datacenter to manage additional permissions granted to the customer. Keys are never communicated in the clear outside of a trusted environment. They are themselves encrypted using key encryption keys. The basis for key based authentication in servers and storage devices is generally secured using embedded components such as Trusted Platform Modules. Secure key communication and

storage using Key Encryption Keys and Trusted Compute Modules are standard practice.

Interaction between customers and cloud datacenters should be encrypted using keys that, once again, originate in customer key management infrastructure.

The cloud datacenter operator is responsible for ensuring isolation of the execution contexts and stored data within the cloud datacenter, secured by customer provided keys, including scenarios where storage capacity is reused by a series of tenants. In addition the cloud datacenter operator must insure that no data can be accessed after hardware leaves the datacenter for reuse, recycling or repair.

In many cases the customer shown in Figure 1 is an intermediary running software services in turn for their own customers. The customer to the left might, for example, consume software services from the customer in the middle which are in turn hosted on cloud datacenter infrastructure. The customer of the service remains responsible for managing keys for its own security which propagate through a chain of trust through the service provider to the cloud infrastructure. Although this pattern may be common it is layered atop fundamental storage protection principle so it is not considered any further in this paper.

2.2 Scaling Multi-Tenancy in Storage

Public Cloud datacenters have many tenants that may consume storage capacity. The number of tenants may range from hundreds to millions. This is magnified by the multiple storage containers (volume, file or object sets) which each customer requires, keyed to these individual customers. At this scale, software infrastructure is required to create, delete and secure the containers. Storage solutions have provisions for multi-tenancy they do not come close to the scaling to the number of tenants required by public cloud data centers. For this reason, the multi-tenancy features of storage hardware are more applicable to smaller scale deployments such as private cloud. Figure 2 depicts high level implementation examples of two multi-tenant storage approaches reflecting this dichotomy of scale.

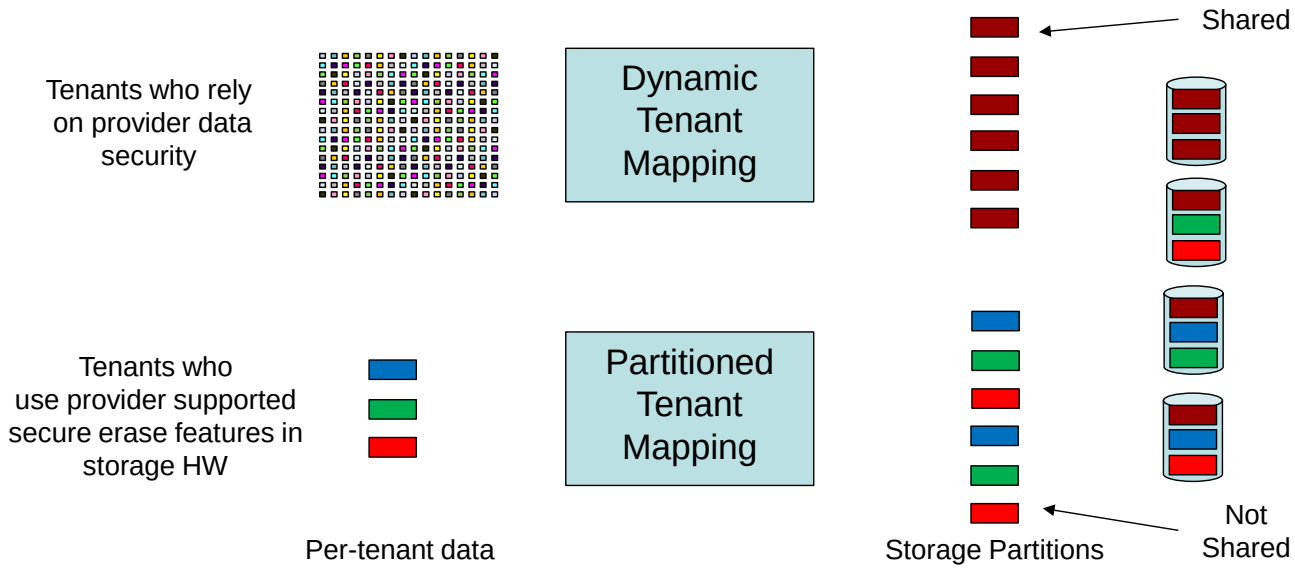

Figure 2 – Scalable Multi-Tenancy

Both of the approaches shown here assume that storage and/or PM devices have a multi-tenant protection capability, for example secure access and secure erase that can handle a limited number of tenants. Cloud datacenters use the upper approach in which large groups of tenants (shown in the per-tenant data column) get secure access to storage using software such as a file system that manages permissions that are keyed to individual customers. These tenants obtain storage space dynamically through the normal operation of the file system (or similar) which causes each partition of storage to contain data from many tenants (the brown boxes in the Storage Partitions column). Storage or PM multi-tenancy features can still be used but they applies to many tenants at once, subject to the layout imposed by the file system (or similar).

The lower part of Figure 2 is more applicable to smaller scale deployments such as private clouds that may still need to protect data consumers from each other. In this approach, constraints must be applied to the mapping of tenant data to partitions in order to assure that no partition contains more than one tenant's data. The number of tenants that can be accommodated by a group of disk or PM devices may be limited by either capacity, or by the number of tenants supported by each device.

If statistics for a private cloud datacenter include tenant data size and physical device capacity then the maximum number of tenants that will fit on a device can be determined. The actual number of tenants the device might encounter must also account for the layout of tenants across devices. As a generalization, many layouts spread a tenant's data across some number of devices. Whenever that occurs there must, at a minimum, be a separate key for the part of a tenant's data that exists on a given device. That way if that tenant is removed, all of its data can be removed from all devices without affecting any other tenant.

To derive a rule of thumb, let DataDevicesPerGroup be the maximum number of devices that can store any part of a given tenant's data given the layout of the data

across devices. Based on that layout characteristic, if the intent is to always have capacity, and not supported device tenant count, be the limiting factor, then the following must be true.

$$DeviceTenants > DataDevicesPerGroup * \left\lceil \frac{DeviceCapacity}{AverageTenantCapacity} \right\rceil$$

This rule of thumb does not account for redundancy, which is layout specific. For many common layouts the additional devices added to provide capacity for redundancy must accommodate the same number of tenants as would have been the case for the devices in a similar non-redundant layout. Therefore DataDevicesPerGroup can often be calculated as the number of devices needed to provide the usable capacity of a group rather than total physical capacity of the group.

If the average tenant capacity is too large to fit across DataDevicesPerGroup, i.e.

$$AverageTenentCapacity > DataDevicesPerGroup * DeviceCapacity$$

then the DeviceTenants calculation above can still serve as an upper bound on the number of device tenants required. Tighter upper bounds may exist depending again on the layout. Also under this condition, additional tenants per device above the upper bound may reduce capacity lost to fragmentation.

In some cases only a subset of the keys are active at a given time, so they can be cached so as to avoid consuming premium hardware resources all the time.

3 Use Cases

Since the purpose of this document is to highlight gaps in security implementations related to PM, it is important to start with the tried and true practices that will continue to be needed. All of these are still required in some form. Many do not need to be modified for use with PM.

3.1 Storage Protection

In general, security enforcement involves authentication and authorization of a principal (data consumer such as a cloud data center customer) to access data. There are two common practices for these.

- Establish identity through an authentication challenge, then succeed with a permission check that indicates whether an access request (e.g. read, write) for specific data or groups of data is allowed. The authentication challenge may involve a password and/or additional functionality such as smart key hardware or secure conveyance of credentials from a prior authentication. Permissions are managed by administrators and users who have security management rights for specific data.

- Establish identity through an authentication challenge in order to obtain a key or keys that enable encryption, decryption or other actions on specific data or groups of data. If encryption and decryption keys do not match then any data accessed is unintelligible. Keys may also be used provide a basis for validation of permissions. For public cloud storage API's it is common to use a certificate (i.e. X.509) for web based authentication and authorization

Authorization may be role based, meaning that principals may be associated with roles they are allowed to take on such as data owner, administrator, etc. Roles imply sets of actions that the principal is enabled to perform. As a result permissions may be granted based in part on role as a way of simplifying security management. See section 4 for an enumeration of the roles that are considered in this document.

In many cases an authentication challenge establishes a root of trust (identity and/or key possession) that confers permissions over a protracted period of interaction such as a login session. In such cases additional measures are taken to insure that the root of trust is still valid after any event that may cause uncertainty as to whether the principal may have changed. Typically events such as resets or session timeouts terminate roots of trust and trigger re-authentication.

If customer controlled physical security is assured and software is trusted to correctly enforce permissions at all times, encryption is not necessary. Under these conditions, correct enforcement of permissions is sufficient. Unfortunately these conditions are never upheld in public cloud environments. In public cloud and other less controlled environments, data travels through domains where physical security is not assured, or if software is not trusted to correctly enforce permissions. In such environments, encryption is the preferred way to maintain data security.

Depending on availability of processing power and time, encryption can be broken through guesswork and/or reverse engineering of keys. This motivates key rotation based on time intervals that place acceptable bounds on the time available to break encryption and on the duration of security violations that may result. Key rotation is a very common IT practice.

Key rotation for encrypted data storage would be very expensive as all data would need to be re-written with every key rotation. Self-encrypting HDD's and SSD's avoid this by encrypting data using one or more device keys that never leave the storage device. The device controller maintains a second key (or set of keys) that is used to ascertain a given principal's right to access part or all of the data in the device. Once a root of trust is established the controller allows access to the associated data using the secret key(s). The second set of keys can then be rotated without changing the secret keys. This method is used in the Trusted Computing Group Self Encrypting Drive (TCG SED) standard which is generally viewed as sufficient for protecting storage devices when physical security is not assured.

These storage security practices are essentially the same whether encryption occurs within a self-encrypting device or in hardware or software on the way to the device. Several caveats should be noted.

- Secure key management techniques must be applied including the use of Key Encryption Keys as described in 3.2.
- Any retention of unencrypted data that is in the process of being encrypted or scheduled for same must guaranteed to be unrecoverable after any event that could compromise security such as power loss, reset or component removal.

Several additional software practices interact with storage security although they are not strictly part of it.

- Applications may encrypt data before it enters storage hardware or software. Depending on key ownership and strength, additional storage encryption may or may not be warranted. Although it is not in the scope of this document it should be noted that early encryption may interfere with value added storage functions such as compression and deduplication.
- Copy on write functionality may be used to create storage or memory images based on a single original. These must be secured in the same manner as complete images based on the identity of the principal consuming the (partial) copy. Such images may also be rendered immutable to provide further protection against tampering.

Some processor architectures now have encryption features built in (e.g. AMD SEV extensions and Intel SGX). This enables memory encryption without self-encrypting NVDIMMs. These processor specific features apply to PM in the same way that they apply to DRAM.Key management – key storage and distribution

Industrial strength security is generally based on a secure key store accessed using protocols such as KMIP. The secure key store must be trusted by the customer of a cloud data center as described in section 2.1. The secure key store is generally managed by the customer's security officer to minimize opportunity for insider attacks from other administrators. Any transmission of keys must be encrypted using a key encryption key which is known only to software or hardware that is also trusted by the customer. All of these practices are already common.

3.2 Security audits of implementations

Certain pieces of software must be designed and implemented using well-known techniques to insure that security is enforced. These include the secure key manager, storage device software that manages encryption, software involved in the use and management of key encryption keys, and software tasked with the isolation of tenants. Generally such software is inspected during and after development by experts in secure software practices using checklists of pitfalls and exposures that must be addressed. For the most critical components, certification such as FIPS may be required.

3.3 Origin and Delivery Protection

For software to be trusted, some assurance that it has not been tampered with is required. This is typically achieved using digital cryptographic signing which provides for authenticated integrity. The same applies to broadly published data. In addition, as software and data elements are packaged together for distribution and installation, there must be some guarantee that the package contains only the digital cryptographically signed components that were intended by a trusted originator.

A related common practice, the use of a Message Authentication Code (MAC), is often applied to any code or data that is broadly shared among readers but can only be updated with special permission. The MAC can be validated by software in the reader's path without special permission, however a writer must use a secret to generate a new valid MAC. This provides additional protection against unauthorized writing with minimal burden on readers.

These common practices also apply to PM security.

3.4 Memory Protection

Current memory protection practices apply to PM. In particular, Memory Management Units (MMU's) enforce memory protection using both virtual address space mapping and physical memory access protection. Details of both of these levels are MMU Implementation specific, and are applied on OS specific ways.

4 Role Definitions

The threat model in this document acknowledges several roles of actors who might pose threats.

- Customer – The data owner whose security and privacy are being protected.
- Tenant – An inhabitant of shared infrastructure. Customers become tenants by establishing accounts with cloud data centers, thus establishing an identity and access to certain resources/services within the cloud.
- Administrator – A person tasked to maintain software and hardware infrastructure. Cloud data centers have administrators who are trusted to keep the data center provisioned, operating and accessible to tenants. Customers may have administrators who are trusted to configure and maintain applications and data running in a cloud datacenter. Neither of these administrators trust each other, and neither is generally allowed access to any data.
- Security Officer – A person trusted by a customer to configure and maintain users, roles and permissions related to data access.
- Developer – A creator or maintainer of hardware or software.
- Deliverer/Repairer – A person who handles components moving into or out of a data-center.

5 Threat Model

The following table enumerates threats to privacy or confidentiality and threats to integrity or availability.

Attack Type	Means of Attack	Attacker	Applicable existing approach	New issues with PM
Privacy/ Confidentiality	Physical manipulation	Administrator, Repair	Encryption at rest.	New NVDIMM Authentication behavior (JEDEC)
	Software access across tenant boundary	Tenant, Administrator	Traditional authorization, authentication. Separation of roles. Memory protection.	NVDIMM does not know principal identity during Ld/St/Mov
	Access to deleted data	Tenant, Administrator	Secure erasure (physical or cryptographic) during deletion.	More rapid free space recycling in memory than disk.
	Access by admin/support	Administrator	Role separation, Authentication/ Authorization	
	Local HW attacks (e.g. DMA)	Tenant, Administrator, Developer	Memory Protection	
	Remote access threats (e.g. RDMA)	Tenant, Administrator, Developer	RDMA security, memory region access protection enforcement	
	Malware	Developer, Delivery, Repair, Administrator	Digital signing, Virus protection to exclude or expunge malware	
Data integrity or accessibility to owner	Data modified/ Destroyed through privacy exposure	All of the above privacy attacks have variations that involve modification, destruction and/or removal of data.		

	Software Defects	Developer, Tenant, Administrator	Traditional authorization, authentication. Separation of roles. Memory protection.	Increased scope of damage due to mismanaged pointers, memory resources.
	Availability – denial of service	Tenant, Developer	Per-tenant QoS	Potential for rapid disruption with limited detection window?

Table 1 – Threat Model

The following sections contain analysis of each of the threats in Table 1.

6 Threats to privacy or confidentiality

6.1 Physical Manipulation

This is the top priority threat to address for PM given the pre-existing security measures that are already in place for other PM related threats. One solution is a self-encrypting NVDIMM, which is precisely analogous to self-encrypting disks. There is one notable difference that stems from the fact that today's NVDIMMs are attached to DDR4. During reads or writes to memory NVDIMMs have no way of identifying the principal initiating the request. Disk drives, on the other hand, receive notification of an initiator with every read or write. Therefore NVDIMMs cannot check per-tenant access permissions during read or write. Self-encrypting NVDIMMs can still be used to ensure that data is unreadable after physical removal, and for cryptographic erasure of all of a tenant's data.

As of the release of this document JEDEC is working on the control path to establish a root of trust prior to making data contained within a self-encrypting NVDIMM component accessible to applications. Self-encrypting NVDIMMs are required to provide some means of securely establishing a root of trust. Existing techniques such as this described in section 3.1 should apply. Since this is a strong analog to disk encryption use case, existing TCG or NIST standards could be applied.

In addition, an NVDIMM communication channel that is logically isolated from data access may be needed. Examples include additional IO control actions through an existing NVDIMM control plane, or an un-encrypted volatile memory region used only for root of trust establishment. Additional system specific firmware and software requirements are also likely in order to enable access to NVDIMM contents early enough in system bootstrap to fulfill all purposes of RAM.

There are various situations wherein the root of trust must be re-established in order to avoid man in the middle or principal substitution attacks. Example situations include power on, reset, hot plug and loss of heartbeat events. Systems containing self-encrypting NVDIMMs are required to ensure that data access is withdrawn and re-authentication required whenever such an event has occurred. In some cases this may be the responsibility of the NVDIMM itself.

Self-encrypting NVDIMMs are required to ensure that no unencrypted data is accessible under any circumstances unless a valid authorization is in place. This includes scenarios where unencrypted data is retained in volatile memory and the NVDIMM is removed from its socket even if the NVDIMM remains operational under auxiliary power.

While encryption of data at rest is a priority, self-encrypting NVDIMMs are not necessarily a requirement if only encrypted data (encrypted by something outside the DIMM) is stored. Encryption deployment use cases generally fall into three groups.

- Self-encrypting NVDIMMs – This is the deployment use case described above. Encryption of data at rest requirements can be met using single or multiple keys within NVDIMMs provided that all data is encrypted. A single key is sufficient to address threats that involve physical removal.
- Encryption in the storage stack or CPU data path – This deployment use case has similar characteristics to self-encrypting NVDIMMs with the additional requirement that data cannot bypass the encrypting component on the way to NVDIMM.
- Encryption by a tenant or application – In this deployment use case, data is encrypted before it reaches any PM specific component (storage stack software, memory controller or NVDIMM). To the extent that all data on the NVDIMM is encrypted by upper level software, encryption of data at rest requirements can be met.

6.2 Software access across tenant boundaries

This section addresses scenarios where different tenants or customers represent threats to each other.

6.2.1 Real time access protection of active data

The DDR4 interface and its predecessors do not enable NVDIMMs to sufficiently protect against cross-tenant access because no form of principal (tenant) identity is communicated to the NVDIMM. Memory address ranges can be encrypted with different keys within NVDIMMs, however once a tenant has established a root of trust with the NVDIMM there is nothing within the NVDIMM to keep other tenants from accessing the protected memory address range.

Therefore all protection against prohibited real time access to PM is based on CPU data paths or software. The type of protection in the CPU data path available today comprises authorization enforcement based on virtual memory systems, not encryption. Virtual memory is already in use pervasively with DRAM and is naturally extended to PM. Virtual memory implementations ensure that a tenant executing outside of the kernel (e.g. in a thread, process, container or virtual machine) can only access memory regions that have been authorized by the kernel. Authorization comprises the creation of page table entries that map tenant accessible virtual memory addresses to physical memory accesses. Only memory regions thus authorized by the kernel can be accessed by a given tenant.

Properly administered, this protection is sufficient to inhibit real time cross-tenant access to active PM, however code running in kernel space must be trusted. There is no protection against rogue kernel code using any physical address to violate privacy.

6.2.2 Access protection of inactive data

Since virtual memory is the primary means of protecting active data, removal of the virtual address space of a tenant can be viewed as the point where data transitions from active to inactive. It is crucial that this transition occur when a tenant program completes or otherwise terminates. While this existing requirement is already met by secure operating systems, it is important that other software such as PM file systems also meet this requirement.

Permission enforcement or encryption of data in storage stacks can also protect against prohibited access to inactive data. For example, storage stack components such as file systems can recognize large numbers of tenants as described in section 2.2, and use authorization or cryptographic methods of enforcement that do not depend on virtual memory.

As with active data, the kernel must be trusted.

6.3 Threats against deleted data

In most systems deletion of data is part of a process that occurs in several stages. In general the process achieves the following results:

- Remove the owner's logical path to the data
- Render the data inaccessible to all software
- Make the storage space occupied by the data available for reuse

6.3.1 Meaning of deletion

Deletion is the removal of a data owner's logical path to the data. This is generally accomplished by removing the data from a namespace such as a file system, at which point the data does not exist by that name any more. Any other paths to the data must be secured before and after deletion. Deletion does not, in itself, imply that the data no

longer exists somewhere in a storage device, nor does it imply that the space previously occupied by the data is immediately free.

6.3.2 Multiple Keys for secure erasure

In order to prevent all threats the data must be erased before the space becomes free. This ensures that deleted data from one principal's point of view is never inadvertently or improperly made visible to another. One approach to this is to over-write the data. The time taken to over-write that data increases with the amount of data and the number of copies. It further assumes that all of the copies can be located.

Another approach is secure erasure through invalidation or destruction of the keys to encrypted data. This is quicker and more secure than over-writing. Still, all copies of the key must be invalidated however this is arguably easier than over-writing all copies of data. Key invalidation is a well understood problem in the secure key management field. While it is not without issues (e.g. copies of keys) customers who require high security must have policies in place to address them to a satisfactory level. Secure erasure of storage can leverage these policies directly.

One approach to secure erasure is the use of multiple keys to isolate tenants from each other. There may be limitations on the number of keys supported in self-encrypting NVDIMMs that lead to special considerations described in section 2.2. In some cases only a subset of the keys are active at a given time, so they can be cached in the hardware and do not need to be there all the time.

When multiple keys are involved, trusted key management is a likely requirement such as that shown in Figure 1. Such solutions also require secure communication of keys using Key Encryption Keys.

6.3.3 Use of the Sanitize command in NVMe, SCSI and SATA

Most disk command sets now have a Sanitize command the removes data from a range of blocks in the drive. Sanitize can also be used as an indication that the consumer of the range has freed the space in the upper layer which allows storage devices or systems capable of thin provisioning to deallocate space. For storage stacks and upper level access methods (i.e. file systems) that can use media aligned sanitization, these functions correspond to the last two steps described above. Additional information on sanitization is available here.

6.3.4 Privacy threats due to loss of re-initialization during reboot or reset

Another privacy threat can arise of volatile data is allowed to survive a reboot or restart and appear within un-initialized data structures thereafter. This can happen with DRAM if power is not removed from the system, or if volatile data is being stored in PM. In either case elimination of this threat requires explicit re-initialization of all memory holding data that is intended to be volatile. This must occur between the event that

triggered the reboot or reset and the first opportunity for any unauthorized software to access the memory. It is the responsibility of system firmware and kernel software to avoid this threat by insuring that volatile data is not accessible after reboot or restart.

6.4 Administrative threats

Referring back to the administrator role definition in section 4, administrators maintain software and hardware infrastructure. Administrative threats through physical manipulation are covered in section 6.1. Cloud data centers have administrators who are trusted to keep the data center provisioned, operating and accessible to tenants. Customers may have administrators who are trusted to configure and maintain applications and data running in a cloud datacenter. Neither of these administrators trust each other, and neither is generally allowed access to any data.

The main requirement related to administration is not new or specific to PM, namely that cloud datacenter and customer administrators must be segregated. They must authenticate separately and have different authorization specific to their roles. Administrative roles may be further refined to separate application management from security management but this practice is once again not specific to PM.

In addition, methods of protecting PM access such as those described in section 6.2 must be securely tied into the administrative chain of trust that establishes access permissions. While there are many ways to model this, the memory mapped file paradigm provides a straightforward model in which access permissions are built into the file system and applied to PM when files are memory mapped. This requires that the granularity of memory protection supports the granularity of permissions.

6.5 Threats exposed through runtime H/W e.g. DMA

In today's systems private user data is virtually always stored in RAM for processing. Runtime hardware such as DMA and other data transformation engines such as encryption or compression already manipulate private user data as it moves in and out of storage or between memory regions. In today's systems physical addresses are often used in DMA control blocks so the protection from the virtual memory address space described in section 6.2 is missing. As a result the avoidance of these threats relies on a trusted kernel being the only code that can generate DMA or other accelerator control blocks. The trusted kernel must ensure that permissions at source and destination are aligned, specifically with respect to the authorized principal.

An alternative to strictly physical access is emerging in interconnects such as open CAPI which enables PCIe attached peripherals (GPU's and possibly HBA's or NICs) to use virtual addresses. This allows virtual address enabled security to be applied provided that the control path that communicated those virtual addresses to the peripheral is also secure. Once again, security of the control path must be established by a trusted kernel even if memory access is self-policing. Still, PM itself does not drive any new privacy related requirements into runtime hardware other than those described in section 6.2.

6.6 Remote access threats (e.g. RDMA)

This is the first use case where PM meets networking. For that reason we need to separate the following concerns.

- Security of data traversing networks
- Security within the endpoints

Both of these are addressed by current practices that are applicable to PM. Encryption is the only effective approach to security of data traversing networks if trust and physical security is not assured throughout the physical network. Since private user data already traverses networks including those with RDMA the need for encryption is not new or specific to PM.

Security within endpoints is addressed in full featured RDMA implementations because virtual address space windows in each endpoint are mapped to a corresponding RDMA address space and securely associated through the use of RDMA Steering Tag properties. Assuming, as always, that the source and communication of RDMA connection information is trusted this allows the virtual memory protection techniques of section 6.2 to be applied to RDMA access. The result is more robust than local DMA using physical addressing.

6.7 Malware threats

Up to this point we have been dealing with eliminating threats as they occur in real time. With malware threats we are trying to detect nascent threats before they become real time threats. This is generally done by scanning files or memory for signatures known to indicate malware. Let's explore PM impact on Malware detection case by case.

2 virus scanner modes: scanning in background or intercepting in the storage stack.

Use Case	Current Approach	Issues if scanner uses file open, read with PM	Issues if scanner uses mmapped PM
Background scan of closed files	Open and read files	None	None
Background scan of open files	Skip or Open file and read shared. Must tolerate modification	Compatibility with active mmapped files Consistency of scanner view during modification	Consistency of scanner view during modification
Scan files during "Open"	Scan data on disk or in memory before open completes.	None	None
Scan files during ingest/download	Open file and read from disk after	None assuming download involves	None assuming download involves

	download but before file can be re-opened	file api activity (close, re-open, unmap) after ingest	file api activity (close, re-open, unmap) after ingest – issue is with ingest access method and scan trigger not scan access method
Scan memory	Scan physical RAM specifically including non-running applications	Memory not accessible via file protocol	NA – current memory scan must already gain direct access to memory
Quarantine non-mmapped file	Remove from visible namespace, refuse to open, mmap	NA	NA
Quarantine mmapped file	Stop process if needed, unmap in addition to above, use memory scan quarantine approach	NA	NA

Table 2 – Virus Protection

Now let's look for potential new requirements that surfaced above.

6.7.1 File open vs. mmap

The distinction between open and closed files is thought to be important to virus scanning because closed files are not changing and can be scanned using normal file system access. With PM there is an additional question as to whether data regions to be scanned are mmapped by an application. As described in section 6.2 there are several situations where regions of closed files may be mmapped.

- Applications and storage stacks sometimes close files after mmapping regions in them. The memory mapped regions survive until unmap is called or the process termination.
- Some file system implementations keep large regions of PM memory mapped as long as the file system is mounted. This practice, and others that allow mmap to survive process termination, are not recommended for security reasons and will not be considered further herein.

The first new requirement that arises from the above is that virus scanners must be able to determine which files contain mmapped regions and treat them as open for the purpose of scan processing. If this requirement is met then closed and non-mmapped files can be scanned use either read/write or ld/st access.

For opened files the question arises as to whether the file is mmapped by an application or not. This leads to the requirements described in sections 6.7.2 and 6.7.3.

6.7.2 Signature detection in changing files

Reliable detection of malware signatures without escapes and misdetections requires some degree of data consistency. For files that are closed or guaranteed not to change this is trivial, but for changing files additional measures may be required. Virus scans of open files may use point in time copies for consistency. The granularity and means of establishing the point in time of the copy depends on virus scan software implementation.

With today's storage technologies the minimum granularity is a storage device block. If virus scanner implementation can be aligned with single storage blocks then the built in atomicity of the storage device may provide the necessary stability. If larger granularities are required by the virus scanner then a heavier weight point in time copy such as a snapshot may be required. Although today's in memory virus scans may have different consistency requirements from storage scans, in memory point in time copies such as those that generally take place during a Linux process fork may be useful. In memory scans of code may benefit from the need for stability during code execution, unless the code is self-modifying which may be more common in viruses.

The above leads to a requirement that files in PM meet the consistency requirements of virus scan implementations. The status quo is sufficient for files that are not memory mapped provided that file system reads and writes to PM use atomic block access through a driver such as BTT. For files that have been memory mapped by applications, in-memory snapshots may be required in order to maintain a consistent view of files being scanned.

Existing memory scans should not be impacted by memory mapping in applications. Some techniques currently used for memory scans may be applicable to mmapped file scans.

6.7.3 Single scanner view of both disk and PM based files

Virus scan providers are not likely to want to provide implementations for PM file systems that are significantly different from those backed of SSD's or HDD's. Barriers to adoption could be most quickly minimalized if file reads and writes can be performed on files that are memory mapped at the time. At least this way the file system API acts as an intercept point on the virus scanner path even if there is no equivalent on the application path.

It remains to be seen whether memory mapping on the virus scan access path is beneficial. It may turn out that the main benefit is copy avoidance or mitigation.

6.7.4 Ingest

Ingest is any scenario wherein data or code moves into trusted storage or memory content from a potentially untrusted or unknown origin. One subtlety occurs if data or code is generated (rather than ingested from outside) by other supposedly trusted code.

In this case the virus generating code is itself malware which should have been detected earlier.

While memory mapping may remove some ability to intercept viruses generated by this type of malware, that is only true if the viruses are stored today using file system (or block or object storage) writes. Existing virus generating malware in today's system may operate as self modifying code meaning that PM did not actually introduce any attack paths that are less detectable than existing threats.

The recommended approach to resisting this type of attack is to carefully govern and monitor the point where the contents of RAM or PM become executable. If that point is explicit such as a permission change or exec call, then there is an opportunity for virus checking of malware generated code. Self modifying code may not have such a transition making it perennially suspect as a virus vehicle with or without PM.

The fundamental ingest requirement is not changed by PM. Ingested data or code must be scanned for viruses before it is used. If the ingesting application is using file access then existing techniques apply including file close after ingest and open before use. Virus scanning can occur during open or close. Explicit virus scanning triggered by ingesting applications such as a download manager are also applicable. Digital signature checking of code is recommended before it becomes executable. These techniques work with block, file or object access to PM regardless of whether files are memory mapped by applications after the virus scan.

If the ingesting application is using memory mapped storage then the recommendation is that the ingested regions be unmapped and containing files closed before use. An explicit virus scan trigger and digital signature check of stable mmapped data is also applicable. An application that ingests data or code and allows it to be used without a virus scan is inherently insecure. Allowing groups of applications to do this by prematurely exposing ingested code or data through shared memory is just as bad. These types of error create vulnerabilities that should be eliminated from the ingesting application implementation regardless of whether the medium is today's RAM or tomorrow's PM.

6.7.5 Scan triggers during writes

In today's systems it is possible to check for viruses or trigger virus scans as a result of writes to SSD's or HDD's. This is more difficult to do when applications are writing directly to memory mapped files since there is no software intervention in that path other than the application itself. If the purpose of these writes is data ingest then the requirements from section 6.7.4 apply. For ingest it is less critical to have a virus intercept in the write access path as long as the application forces a virus scan after ingest and before use as is the norm for applications such as download managers.

The return on investment for triggering virus scanning during memory mapped writes is hampered by a number of factors.

- Applications that create viruses by writing to memory can be viewed as malware themselves, and should ultimately be detected as such.
- Writes to memory at cache line resolution are so fragmented that it may be impossible to detect a virus signature in any one write.
- Intervention during writes, if required, would almost certainly involve page faults. Even if hardware is added to detect viruses, it would be expensive to complete detection during a memory access.
- If the purpose of intervention during writes is to trigger a subsequent virus scan then there is always a window of vulnerability between the write and the scan, during which additional writes may repeatedly create the same trigger. This creates a tradeoff between scan frequency and exposure to nascent viruses. The likely resolution of this tradeoff is to set a minimum time between virus scans of modified files or mmapped regions.

Because of these factors, very few if any customers are likely to demand virus checking during memory mapped writes. More likely, dirty pages should be tracked so that a periodic check can quickly detect pages and files to be scanned at an acceptable rate. This can still be problematic if dirty page bits can be turned off by the processor without notifying software when the processor decides on its own to flush a page to memory.

In a sense this sends us back to square one with the exception that dirty page tracking hardware for PM (as opposed to processor cache) is much less expensive than virus checking during memory access. In fact many processors have support for dirty page tracking in RAM to support caching between RAM and SSD's or HDD's (e.g. the Linux page cache). Even though PM allows pages to be viewed as persistent without writing them to SSD or HDD, processor page cache infrastructure for dirty page tracking in RAM, if available, could remain enabled in PM for the purpose of tracking pages that need virus checking.

7 Threats to availability or data integrity

This section elaborates on the lower part of Table 1.

7.1 Leverage of privacy threat into data integrity threat

Any of the privacy threats in the first part of Table 1 could also result in threats to data availability or integrity. In most cases the defenses described in section 6 also protect against unauthorized writing of data which could lead to data integrity or availability issues.

- Physical manipulation – This attack is resisted using authentication and/or encryption that cannot be completed if the drive is not connected to a system that has the necessary permission. This protection extends to writes because inability to authenticate disables writes. An unauthorized host that is unable to authenticate but cannot correctly decrypt data could still compromise data integrity by writing invalid (unencrypted) data on the drive. For this reason, a challenge protocol to establish access authorization is required to avoid integrity issues involving physical manipulation.

- Software access – the methods described in section 6.2 must be applied to both read and write permission.
- Deleted Data – writing deleted data does not cause any integrity issues once the first step of the deletion process described in section 6.3 is completed. Up to that point the data has not yet been deleted so it must be protected from all other threats.
- Administrative threats – the methods described in section 6.4 must be applied to both read and write permission.
- DMA, RDMA – Access rights must be enforced on both the source and the destination of the data.

Data integrity threats due to malware deserve special attention. In the case of malware, a principal that appears to have access authorization contains malicious code. If the code has not been detected as malware (see section 6.7), the only possible protection is to constrain authorization to the smallest possible granularity. For example if certain structures within larger storage or memory regions should never be written, then writes to those structures should be inhibited regardless of write authorization for the larger region. This requires detailed application specific authorization assertions which may not match the granularity of enforcement supported by the system. Even with such assertions in place, data that is normally writable by the application is at risk due to undetected malware.

7.2 Software defects

Concern has been expressed over the possibility that software defects such as rogue pointers could cause more disruption by writing PM than with volatile RAM. In the end the credible new threats posed by software defects have several categorical root causes.

- Data integrity threats due to the loss of separation between application data structures and permanent storage.
- Denial of service threats due to the loss of complete RAM state re-initialization during reset that can lead to perpetual reboot.

7.2.1 Loss of separation

A software defect may cause an errant write to any PM that is memory mapped to the process performing the write. This has always been the case for volatile RAM. To the extent that data historically written to disk is derived from variables and stored volatile RAM buffers before writing, this is not a new exposure. The difference is that historically the errant write is not made permanent until a disk IO completes. If the error is detected before the write is made permanent, the previously written data is still available for recovery.

With PM the time window of separation between the errant write to memory and the commit to permanent media is much smaller, if not non-existent. One notable activity that frequently occurs in the window of separation is the calculation of a CRC or other digest to protect data integrity. The loss of separation implies that the data and CRC may be inconsistent between the time when the data reaches the PM and the time

when the CRC reaches the PM. Since processors only provide failure atomicity of fundamental data types a transaction like construct is required to manage data and CRC together.

Consider the following timeline as a model of data integrity threats when CRC is in use.

T0 – Data and CRC are consistent. CRC check will succeed.
T1 – Data modified in memory mapped data structure or buffer
T2 – CRC modified in memory mapped data structure or buffer
T3 (HDD/SSD only) – Disk write IO complete
T3 (PM) – data and CRC flush complete to PM (possibly out of order)

Time window specific exposure consequences of corruption. (applies to both disk and PM)

- T0-T1 – new data over-writes corruption unless corruption is in a part of a buffer or data structure that will not be modified at T1. Corruption of data in bytes that were unintentionally modified will propagate to the media with good CRC. With PM there is a chance that the write size more closely matches the new data size so exposure may be smaller.
- T1-T2 – CRC may or may not detect corruption depending on the instantaneous relative position of corruption relative to CRC calculation progress.
- T2-T3 – CRC will detect corruption even with indeterminate PM flush order.

Resisting PM integrity loss due to software defects may involve the following steps listed in increasing order of cost and complexity.

- Implement protection from integrity threats described in section 7.1. This will catch many grossly errant accesses.
- Use CRC to protect data end to end.
- Use transactional constructs and CRC to enable recovery if corruption is detected.
- Constrain authorization to the smallest possible granularity with respect to memory regions within a file or process. This was also mentioned in section 7.1 as a means of early malware detection.
- Constrain authorization to the smallest possible time granularity. This requires dynamic enablement and disablement of writes to memory surrounding the execution of application code that expects to modify PM. This will inhibit writes during time periods when none should occur.

7.2.2 Loss of re-initialization

The concern with this type of threat is that a reset or reboot that, were it not for PM would enable recovery from an error, does not because the error is permanently in PM. This can be factored into several scenarios.

- A software defect has corrupted a file in a way that inhibits reboot. This avenue is covered in section 7.2.1 based on the assumption that the file was intended to be persistent all along. Once integrity has been compromised in either PM or

(historically) storage there is no recourse other than rolling back to earlier versions of files.

- A software defect has corrupted contents of memory that would, without PM, have been discarded during reboot. The way to avoid this threat is to discard the contents of memory that is intended to be volatile even if it is stored in PM.

The root cause of the first exposure above is actually the loss of separation between memory and storage. The fact that such an exposure could compromise reboot increases the importance of the defenses described in section 7.2.1.

The root cause of the second exposure is failure to re-initialize memory containing volatile data during reboot, hence the section title. Dependency on blanket re-initialization during reboot or reset can be viewed as a software defect in and of itself in addition to software defects that may have created the problematic volatile memory state. Loss of re-initialization also causes a privacy threat described in section 6.3.4.

7.3 Denial of service threats

There is some concern that PM might enable new types of attacks in which lower PM latency compared to storage might enable new types of denial of service like attacks. There are two types of scenarios to consider.

- Software is constantly accessing PM and not allowing other software to run. This scenario can already occur in today's systems with DRAM which is lower latency than PM. This may be aggravated by the use of flush and fence instructions to force data to PM. These instructions may take a long time (e.g. up to Seconds) and may affect multiple cores, creating new processor scheduling challenges.
- IO intensive software can now use lower latency PM, eliminating periods of waiting for storage. This scenario is not new either because compute bound software already avoids waiting for IO in today's systems. All that could happen is an IO bound workload could be converted into a compute bound workload.

Both of these scenarios are dealt with today in OS schedulers which are designed to ensure that no process gets more than its intended share of processor or memory resources. Although PM may shift the number of workloads that require scheduler driven pre-emption it does not create any entirely new denial of service threats.