PALM: Security Preserving VM Live Migration for Systems with VMM-enforced Protection

Fengzhe Zhang, Yijian Huang, Huihong Wang, Haibo Chen, Binyu Zang
Parallel Processing Institute
Fudan University
{fzzhang,yjhuang,hhwang,hbchen,byzang}@fudan.edu.cn

Abstract

Live migration of virtual machine (VM) is a desirable feature for distributed computing such as Grid Computing and recent Cloud Computing by facilitating fault tolerance, load balance, and hardware maintenance. Virtual Machine Monitor (VMM) enforced process protection is a newly advocated approach to provide a trustworthy execution environment for processes running on commodity operating systems.

While VMM-enforced protection systems extend protection to the processes in the virtual machine (VM), it also breaks the mobility of VMs since a VM is more closely bound to the VMM. Furthermore, several security vulnerabilities exists in migration, especially live migration of such systems that may degrade the protection strength or even break the protection.

In this paper, we propose a secure migration system that provides live migration capability to VMs in VMM-enforced process protection systems, while not degrading the protection level. We implemented a prototype system base on Xen and GNU Linux to evaluate the design. The results shows that no serious performance degradation is incurred comparing to Xen live migration system.

1 Introduction

System virtualization has been regarded as a technology bearing great potential in improving system security [6]. Many researchers have advocated VMMs as the basis for secure and trust computing by reforming current VMM design and implementation [9, 12, 14, 17]. By providing strong VM isolation or relocating privileged operations into the VMMs, these research efforts have established a secure computing environment to host VMs.

Further steps are taken to improve the security and trustworthiness of applications running inside VMs, especially for VMs running commodity operating systems. Recently, researchers also advocate providing high-assurance execution environments for applications in low-assurance commodity operating systems [4, 5, 7, 23]. By using the VMM to enforce the privacy and integrity protection for the processes, these systems is capable of protecting security-sensitive binary or data at runtime even on the fully compromised operating systems. The VMM is trusted and used to protect the privacy and integrity of specified applications running inside VMs. As the VMM is more privileged than the guest operating system kernel, the protection could not be bypassed.

VM migration, especially live migration [8, 15], is an desirable feature in Grid and Cloud Computing systems [3] for load balancing, elastic scaling, fault tolerance, and hardware maintenance. When VMM-enforced protection is applied to applications running on distributed computing environments, besides the basic live migration support, it is required that the protection strength is not lowered during and after the relocation of a VM.

The VMM-enforced process protection systems work well until VM migration is concerned. The initial design of these systems do not include VM secure migration.

First, a VM cannot be straightforwardly migrated if there are protected processes inside. As the VMM extends its protection to the process granularity, the VM is no longer a blackbox. The VMM should have the knowledge of processes, process groups and their runtime states. Maintenance data for each protected process or even each protected page are stored in the VMM. For example, CHAOS [4, 5] maintains one encryption key for each application and Overshadow [7] stores one randomized encryption key for each protected page. These metadata and their manipulation functions as a whole can be regarded as a kind of *emulated memory device*. Decoupling and reconnecting of this memory device is required for VM migration.

Second, live migration of a VM for these systems may impose critical security issues. Simply using cryptographic means and hashing to protect the sensitive data and meta-

978-0-7695-3363-6/08 $25.00 © 2008 IEEE
DOI 10.1109/APTC.2008.15

IEEE computer society

data is not enough. Critical issues lie in the VM *live* migration, in which the migrating VM is still running while the migration is in process. Time-of-check to time-of-use (TOCTTOU) [2, 13] attack and replay attack could be launched if the protection for migration is not carefully designed. A resumption ordering attack could threat the security on the VM restore phase on the migration target machine.

For security preserving migration, several requirements should be met.

- *Data Integrity and Privacy:* Integrity and privacy protected contents should remain protected in the comparative security level during and after the migration.
- *Metadata Integrity and Privacy:* Security protection metadata should be packed, transferred, unpacked and re-constructed with high strength of security protection.
- *Mobile Security:* The VM live migration should not introduce security holes that do not exist in when VMs are not migrated.
- *Performance:* Performance degradation incurred by secure migration should not be high, so that migration downtime can be within an acceptable range.

The goal of this paper is to preserve integrity and privacy protection during and after VM live migration for VMM-enforced process protection systems. Such secure migration should be achieved within an acceptable performance.

We have implemented a prototype system based on Xen [1] and GNU Linux, called PALM (Protection Aegis for Live Migration of VMs). We evaluate the PALM prototype with typical server application benchmarks. The performance evaluation of the prototype shows that VMs with VMM-enforced security protection can be live migrated with reasonable additional downtime and migration time.

The rest of the paper is organized as follows. The next section introduces the background of VMM-enforced process protection systems. Section 3 describes the design issues for *live* migrating the VMs with protected processes running inside. Section 4 describes the implementation details of our prototype system. Section 5 evaluates the implementation of the PALM prototype. Section 6 discusses related work. Section 7 concludes the paper.

2 Background

In this section, we introduce the background on mechanisms of VMM-enforced process protection systems. We mainly discuss the CHAOS [4, 5] system and focus on the mechanisms that are concerned in the design of secure VM live migration. Other related systems are expected to share similar problems and solutions with the CHAOS system.

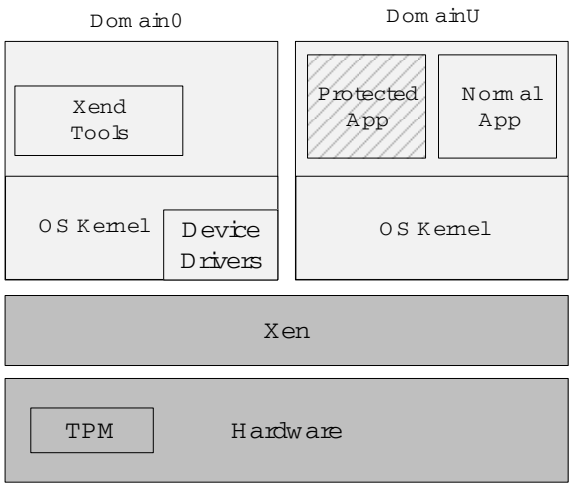

Figure 1. Threat model. Boxes painted in gray are components that are trusted. Boxes painted in white are outside TCB. The box with diagonal lines is the protected application. It is isolated from the operating system, but not a part of TCB.

2.1 Threat Model

The threat model of VMM-enforced process protection systems are slightly different from each other. In common, they all regard the guest operating system as untrusted. The VMM is trusted and used to enforce the protection to specified applications running inside the VMs. The protection could not be broken or bypassed even in the face of a totally compromised operating system.

The threat model of the CHAOS system is even stronger in that the platform owner (people who has full control of the machine, i.e., the administrator) is not trusted. This assumption is realistic in the remote execution environment where jobs are uploaded to the service provider and executed on anonymous servers.

When the two assumptions are combined, the threat model is greatly different from previous research. A commodity operating system is complex and thus error-prone. In addition, due to its rich functionality, it is inevitably vulnerable to deliberate attacks. Commodity operating systems are not an ideal basis for a trustworthy execution environment [7, 21]. In remote execution environment such as in Grid and Cloud Computing, the platform owner can potentially be malicious. In some cases of enterprise computing and scientific collaboration, it is desired that the code and data that the clients uploaded are kept confidential while the services could still be provided.

Figure 1 shows the threat model of the CHAOS system.

The CHAOS is built based on Xen VMM. In the term of Xen, each VM is a *domain*. *Domain 0* is the privileged domain and *Domain U* refers to other unpriivileged VMs. Domain 0 contains a full-fledged Linux kernel with device drivers and *Xend* tools. Xend tools are management tools used to control other VMs via privileged interface provided by Xen. Boxes painted in gray are components that are trusted. As shown in the figure, Domain 0 operating system kernel is outside the TCB.

To establish a certain kind of protection, a trusted computing base must be established. In this threat model, we do not expect the commodity operating system to be trusted. However, we expect the hardware is manufactured by genuine hardware vendors and attacks do not come from the direct compromise of the hardware. Another trusted software layer we assume is the VMM, or the hypervisor. Compared with a commodity operating system, a VMM is much smaller and simpler. A VMM is usually static, which means it does not import other modules at runtime, such as device driver modules. To this point, VMMs could be a more reliable TCB than a commodity operating system [9,16]. In our threat model, we do not trust any component from the VMM layer above, including Domain 0 and privileged tools upon it. In order to guarantee the trusted VMM is working on the remote server, a TPM (Trusted Platform Module) and the TCG remote attestation is required. In this paper, we share the threat model with the CHAOS system.

2.2 VMM-enforced Process Protection

Research efforts are made to build the trustworthy execution environment in the face of a malicious platform owner or a compromised operating system. VMM-based security systems are advocated that provide isolated execution environments for applications [9, 12, 14]. However, though the VMs are isolated from each other, the operating systems running inside the VM is not isolated from the applications, and thus should be trusted. As the commodity operating systems are too complex and vulnerable to be trusted, a practical way is to reduce the size of the operating system so as to increase its trustworthiness. However, this approach limits the functionality of the operating system. The tradeoff of functionality and trustworthy hinderes the prevalence of VMM-based Trusted Computing systems.

Recent research [4, 5, 7, 23] suggests that it is possible to provide a trustworthy execution environment for applications without trusting the operating system. In these systems, the applications are protected directly by the VMM. Even in the face of a totally compromised operating system, the privacy and integrity of the protected binary/data files, memory contents, and CPU contexts are still protected.

2.3 Xen Live Migration

Xen Live Migration system is first introduced in Xen [8]. There are three major phases in the process of live migration, pre-copy, stop-and-copy, and resumption phase.

In pre-copy phase, the migrating VM is still running while the migration manager in the control VM sends its memory content over network in the round basis. The VMM activates the dirty tracking mechanism for the migrating VM so that the dirtied pages can be logged in a bitmap. In each pre-copy round, the migration manager will obtain the dirty bitmap of the previous round, and send the dirtied pages again. This iterative process stops when certain terms are met, e.g., if there are only a few remaining pages to send. In the pre-copy phase, the migration manager in the control VM is capable of issuing a privileged hypercall (system calls from OS kernel to the VMM) to map the memory of the migrating VM. In the implementation of Xen Live Migration, the mapping-and-send operation is done by batch. Each batch, 1024 pages or less are mapped into the memory space of the migration manager. After having finished sending the batch, the manager unmaps these pages and proceeds with the next batch.

In the stop-and-copy phase, the migration manager suspends the migrating VM and sends the rest of the memory contents together with other data to the receiver machine. In this phase, the migrating VM is not active and will not dirty any pages.

In the resumption phase, the migration manager on the receiver machine sets up the execution environment and brings up the VM.

Total migration time is counted from the start of pre-copy phase to the time VM is resumed on the target platform. Migration downtime is counted from the start of stop-and-copy phase to the end of resumption phase. Live migration greatly reduced the migration downtime by pushing most of the work to the pre-copy phase, but at the cost of moderately expanded total migration time.

As the disks and file systems are usually very large, Xen Live Migration system uses network file systems such as NFS and iSCSI to avoid migrating disk contents over network.

3 Security Preserving VM Live Migration

Our design goal is to provide live VM migration capability to systems built with VMM-enforced process protection, while preserving strict protection during and after the migration. Meanwhile, the performance degradation and migration downtime should not be seriously increased. In this section, we describe our design mainly based on Xen and the CHAOS system.

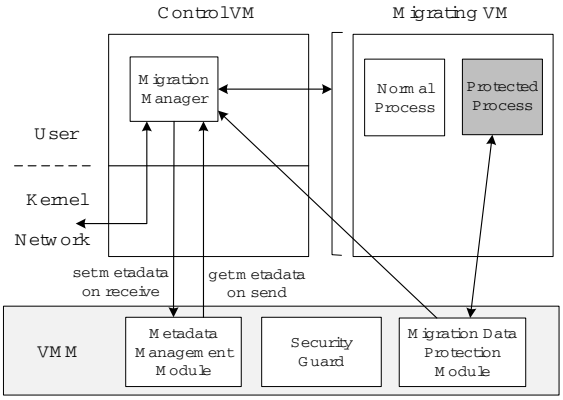

Figure 2. Overall architecture of secure VM migration. The VMM depicted with light grey is the trust computing base. The *Migration Data Protection Module* is responsible for intercepting and protecting contents that belong to the protected processes inside the migrating VM, which is depicted in deep grey. The *Metadata Management Module* is responsible for packing and unpacking the maintenance metadata inside the VMM. The *Security Guard* protects the live migration system from several security vulnerabilities. The *Secure Migration Manager* is the migration tool in the Control VM. It manages the secure migration, but is not necessary to be trusted. The solid lines in the graph show the data flow in the migration.

3.1 Overview Architecture

In security preserving VM live migration, the challenges lie in three aspects:

- Preserving the privacy and integrity of protected contents.
- Packing the maintenance metadata in the VMM, solving the namespace conflict, and re-establishing the protection base on the target platform.
- Eliminating the security vulnerabilities imposed by *live* migration.

Based on the CHAOS process protection system and Xen live migration framework, we propose a security preserving VM live migration architecture, as depicted in Figure 2. Three modules are added in the VMM. The *Migration Data Protection Module* is responsible for intercepting, encrypting and decrypting contents that belong to the protected processes inside the migrating VM. The *Metadata Management Module* is responsible for serializing the metadata for transmission and re-constructing the metadata in the migration target machine. The *Security Guard* protects the live migration system from several security vulnerabilities (discussed in section 3.4).

The *Migration Manager* is the migration tool inside the control VM. This module is not within the trusted computing base. Thus it provides the migration functionality without being trusted. This is possible by encrypting and hashing all the sensitive data before passing to the module.

In the following subsections, we first present the major function modules that provide protected data and metadata migration, and then describe the security issues and our solutions in live migration.

3.2 Protected Data Migration

Before migration, memory pages of secure processes are kept inaccessible from other processes and operating system kernel. This protection should remain valid during migration. In addition to protection for local data privacy and integrity, protection from several network attacks during migration should be addressed, including spoofing, replay and man-in-middle attacks. Denial-of-service attacks are not considered as the platform owner or a malicious kernel can refuse to provide service anyway.

The migration of a memory image is illustrated in Figure 3. In the managed migration scheme, all memory contents are transmitted via the migration manager in the control VM. As the management tools reside outside the trusted computing base, they should not be allowed to access the plain texts of the protected memory pages within the migrating VM. The control VM is capable of mapping memory pages of other VMs via privileged control interface. Through this privileged interface, normal pages (i.e., non-secure pages) are mapped directly in the memory space of the migration manager.

Then the migration manager transfers the content of the pages via network to the target machine. When the migration manager tries to map secure pages, the protection logic embedded in the memory mapping module will make an encrypted copy of the page in the memory buffer of the control VM, and map the encrypted page for the migration manager. This interception, encryption and redirection is done transparently to the control VM kernel and the migration manager.

To encrypt the pages, one can use one global migration session key or assign each page with a random key. Other normal pages are left unprotected. Data integrity can be protected by hashing the memory content at page granularity. These hash values together with the keys should be encrypted again using the private platform key (SK) provided

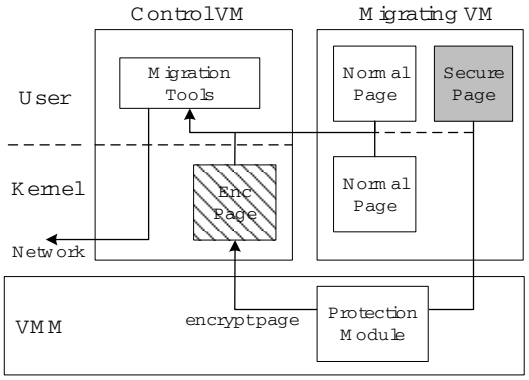

Figure 3. Memory page transmission. Normal pages are directly mapped in the memory space of the migration manager. Mappings requests for the secure pages are intercepted by the protection module in the VMM. The protection module makes an encrypted copy of the secure page and redirect the mapping to this page. The solid lines show the actual data flow of the memory contents. The dash lines show the data flow in the view of migration manager. The interception and redirection is transparent to upper OS kernel and the migration manager.

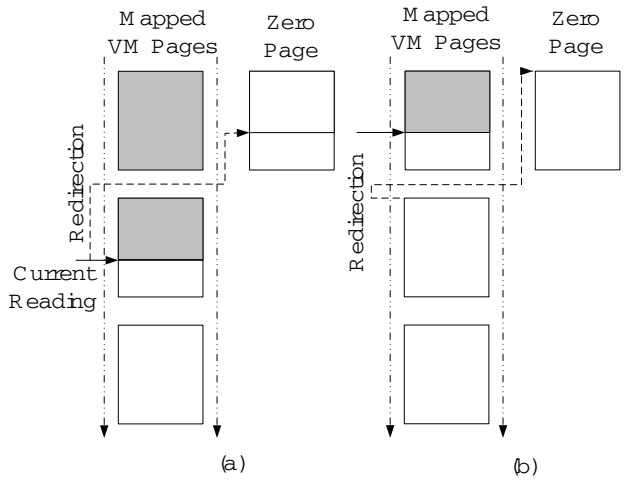

Figure 4. Mapping revocation and redirection. When a page is assigned as a protected page, the existing mappings to this page is checked. If the page is currently mapped in page table of the migration manager, that mapping will be revoked by the VMM and redirected to a prepared zero page. In the left graph, the page is currently being read. In the right graph, the page has not been sent yet. This revocation and redirection is performed no matter the page is currently being read or not.

by the TPM before sending them over network. This additional encryption is essential because the migration tools are not trusted.

On receiving the memory pages, the migration manager simply puts the received contents in the corresponding memory locations. Before resuming the VM on the target machine, the VMM is responsible for locating and decrypting all secure pages. It first decrypts the keys and hashes by the public platform key (PK) of the source machine. Then it compares the hash value of the secure pages before decrypting them. This platform/session key based security protocol is naturally preventing the man-in-middle attack which hijacks the communication by cheating both sides.

After receiving all the memory pages, the VM image is rebuilt on the target machine. As the rebuilding process is under the control of the potentially malicious migration manager, it may switch the content of two protected pages of a protected process to launch a spoofing attack. This is possible as we do not hash the VM image as a whole. To prevent this attack, a kind of binding should be established between the page content and its memory location. A simple way is to bind the hash of each page with the page frame number (PFN) and send them in the metadata transfer phase, which is explained in the following section.

3.3 Metadata Migration

The metadata to be migrated includes encryption keys and hashes, process identities, process CPU contexts, process group info, system call info, and opened file info [4, 5]. The metadata on a platform are structured in arrays, linked lists and hash tables to be manipulated efficiently. In migration, the metadata related to the migrating VM should be peeled off from the data structures and serialized at the source platform, and re-constructured at the target platform. In the pre-copy phase, the metadata are still live and likely to be changed. As the dirty tracking of metadata in the VMM is non-trivial and structured data are not suitable to be sent at the round basis, we pack and send the metadata in the stop-and-copy phase. The metadata receiving and unpacking happens after the target side has received the whole VM memory image.

In migrating the metadata structures, all the internal pointers will be invalidated because the objects will be relocated at the target platform. Some of the inter-reference pointers can be readily re-established at the target side. For example, the linked list of the per-process data structure can

be rebuilt simply by allocating a new linked list and fill them in with the metadata at the target side. Other inter-reference pointers are turned into index numbers and the unpacking function at the target side is responsible for looking up the address of the referred object.

3.4 Secure Live Migration

The protection approach presented in previous subsections works well when the VM is suspended before it is migrated. In the context of live migration [8], the protection is not sufficient. Here, we describe the security problems existing in live migration, including time-of-check to time-of-use (TOCTTOU) attack, resumption ordering problem, and replay attack.

3.4.1 TOCTTOU

There is a TOCTTOU security vulnerability in the map-and-send mechanism when *live* migrating the VM. The key difference is the VM is running when the map-and-send is in process.

When the migration manager issues a hypercall to map a batch of memory pages of the migrating VM, the check is performed to see if the pages are protected. Then the VMM encrypts the protected pages. However, if a page is not protected when the migration manager performs the checking and mapping, and then the page is assigned to a secure process in the migrating VM, that page turns into a sensitive page but is still mapped in the space of the migrating tools.

We fix this security hole by revoking and redirecting the memory mapping to a zero page when the page is assigned as a secure page. As shown in Figure 4, the moment a page is allocated as a secure page, the *Security Guard* will check the page table of migration manager and force any existing mappings to the secure page to be unmapped. To prevent the page fault, the *Security Guard* will temporarily map a zero page to the page table entry. This forced redirection is performed no matter the mapped content has been sent, is being sent, or has not been sent yet. It is possible that part or all of the page may be sent in zero value. This is safe because the page is assigned to another process, which indicates the page content is obsolete. And if the protected process writes the page later, it will be dirtied and sent again.

When a secure page is released and turned into a normal page, the content will not be re-sent as long as it is not dirtied again. This is safe (i.e., content confidentiality is maintained) because the transmitted content is in encryption form and will not be decrypted by the VMM at the target machine because the page is no longer a secure page.

3.4.2 VM Resumption Ordering

Adopting a wrong resuming order may also incur serious security problems. For example, the Xen migration manager will pin all page table pages when a VM is received and about to resume. Page pinning is a process that Xen validates the mappings in the page tables and guarantees the VM has the privilege to map the memory pages. Meanwhile, the metadata that indicate the security level of pages are also set in the resuming phase. If the page tables are pinned before the correct security level of pages are set, it is possible that an unauthorized page table can bypass the security check and illegally map the secure pages. Although we can implement a correct hypercall invocation sequence in the migration manager, it is possible that the migration manager is compromised, and hence breaks the sequence.

We fix this security hole by enforcing the resuming order in the VMM. Specifically, the VMM ensures the pinning operation cannot be done before it has fully received and restored the metadata of page security info.

3.4.3 Replay Attack

As introduced in section 2, pages are sent in a round based fashion in the pre-copy phase. In each round, the migration manager will get the encrypted image of the protected page and send it over network if the page is dirtied in previous round. If a protected page is dirtied multiple times, it is possible for a replay attack that a malicious migration manager sends the old content instead of the new version. To prevent this attack, we hash the content of each protected page in the stop-and-copy phase. This guarantees the hashing values are finalized. The VMM on the target node will confirm that all the secure pages match the final hashing before resuming the VM.

4 Implementation

We have implemented a prototype system called PALM (Protection Aegis for Live Migration of VMs), which is based on Xen VMM and GNU Linux on x86 architecture. The VMM protection system is based on the CHAOS framework [4, 5].

Table 1 shows the coding effort for PALM prototype system. The LOCs is counted without comments. The implementation is not currently optimized. As shown in the table, the major coding effort lies in the *Metadata Migration Module*. Most of its code is to realize the serialization, packing and unpacking logics.

In the following subsections, we will discuss the implementation details in the PALM system.

	Lines of Code
Protected Data Migration	116
Metadata Migration	513
Security Guard	45
Xend Tools	119
Total LOCs	793

Table 1. Coding effort for PALM prototype system.

4.1 Migration Data Protection

The *Migration Data Protection Module* is embedded in the MMU management module of the VMM and CHAOS. This ensures the protection mechanism is activated each time Domain0 (the default control VM in Xen system) performs some privileged operations, like mapping the memory pages of other VMs.

At runtime, the CHAOS base system has the tracking of all secure pages currently being used. The protection module checks all the page table update requests. When the migration manager requests a mapping to a secure page, the protection module will first find an empty buffer page and copy the encrypted content of the secure page to the buffer. Then it alters the page table update request to map the buffer page.

Current Xen Live Migration implementation maps and sends memory in a batched fashion. Each batch operation sends less than 1024 pages. So our redirection buffer can adopt a very simple allocation algorithm that uses a buffer pool with 1024 pages by round robin. As a proof of concept prototype, we assume there are no two live migrations happen concurrently.

4.2 Metadata Management in VMM

The *Metadata Management Module* is implemented in Xen VMM as two hypercall handlers. One handler is responsible for collecting, serializing, and packing the metadata. The other handler is used for unpacking the metadata, solving name conflicts, and re-constructing the metadata.

As it is difficult to dirty tracking data structures inside Xen VMM by itself, and the size of the metadata is relatively small (usually less than several pages), we choose to handle the metadata transmission after the transmission of whole VM image. At that time, the migrating VM is suspended and its maintenance data will not be further tainted.

In collecting the metadata, we remove maintenance data structures from linked lists or hash tables and serialize them in buffers grouped by structure type. In this process, we also sanitize the data structures by replacing the virtual addresses pointing to Xen objects with index numbers, and translating the metadata containing machine addresses to physical addresses. After packing all the metadata in an aggregate buffer, the buffer is encrypted and hashed as a whole.

On the receiver side, the metadata package is handled reversely. It is first decrypted with the migration session key and hashed for integrity check. Then the data in the aggregate buffer is re-constructed. Before the metadata are merged to those on the platform, the names are re-allocated and updated as a whole.

4.3 Security Guard

The *Security Guard* is implemented with little coding effort, but is of great importance. To prevent the TOCTTOU vulnerability, we embed a checking and redirection function in the page allocation module in Xen VMM. In addition, we record each batched mapping requests issued by the migration manager. On the page allocation, the guard is activated when a secure page is to be allocated. It checks if the secure page has already been mapped by the migration manager. If it happens to be true, the guard reversely finds the page table entry that maps the secure page by looking up the mapping request record. Then it replaces the mapped page with a zero page. As discussed in previous section, it is safe to send a part or entire page in zero value.

Another guard function is added in the page validation module in Xen VMM. This guard ensures a resuming VM should first rebuild the security metadata and then start page validation checking.

4.4 Secure Migration Manager

The Live Migration manager is modified to support the secure migration. As the protected pages are migrated the same way as normal pages from the perspective of the migration manager, we only need to add a phase to get and migrate metadata that Xen provides on the sender side, and receive it and pass it to Xen on the receiver side.

As the pre-required key negotiation protocol has been implemented [4], our prototype simply assumes a migration session key that is pre-approved by both sides.

5 Evaluation

In this section, we measure the performance of PALM with server application benchmarks. From the experimental results of two server application benchmarks, the security preserving live migration mechanism incurs at most 3.94 times slowdown for total migration time and 7.92 times slowdown for downtime comparing to Xen live migration. We also compared two configurations of the PALM, one of

fully functional and the other with all the protection mechanisms disabled. The slowdown is at most 1.18 times for total migration time and 3.67 times for downtime.

5.1 Experiment Environment

We evaluate the prototype PALM on four machines: one as NFS server, two as working nodes, and one as client node. The four machines are connected by 100M Ethernet LAN. The two working nodes are each equipped with a 2.33GHz Intel Core 2 Duo CPU, 2GB RAM, a single 320GB SATA disk, and an RTL8169 Gigabit Ethernet NIC. The NFS server is DELL sc1420 server with two 3.20GHz Xeon processors, 2GB RAM, one 73.4GB SCSI disk, and NetXtreme Gigabit Ethernet NICs. The client node is equipped with one 2.8GHz Pentium 4 CPU, 1GB RAM, a single 160GB SATA disk, and an RTL8139 Ethernet NIC.

For Xen live migration, we use Xen-3.0.2 kernel and Xenlinux-2.6.16 for both domain 0 and domain U. We also use the genuine Xend tools to manage the migration. In PALM setup, we use PALM modified Xen kernel and modified Xenlinux-2.6.16 kernel for both domain 0 and domain U. The migration manager is also modified for PALM. The PALM modified Xen kernel is configured into two versions: with and without data integrity and confidentiality protection. The version without data protection is for identifying the actual overhead the protection mechanism incurs during migration.

We evaluate the system performance under the workload of two prevalent server applications to show how much overhead the protection in PALM incurs during the VM migration. The two application benchmarks are the very secure FTP daemon (*vsftpd 2.0.4*), and Apache http daemon (*httpd 2.2.2*). These two server applications are well protected in the CHAOS system. Throughout all the experiments, the two server applications are statically compiled and linked (this is a requirement in the CHAOS system to guarantee memory isolation).

5.2 Performance Evaluation

In the experiment, we get the migration time and VM downtime under 5 different system workloads.

1. *idle*: the system is migrated with no workload.
2. *vsftpd-idle*: the vsftpd server daemon is running in the background.
3. *httpd-idle*: the httpd server daemon is running in the background.
4. *vsftpd-bench*: the vsftpd server daemon is running while the client node is running benchmarks to connect and get big files from the server.

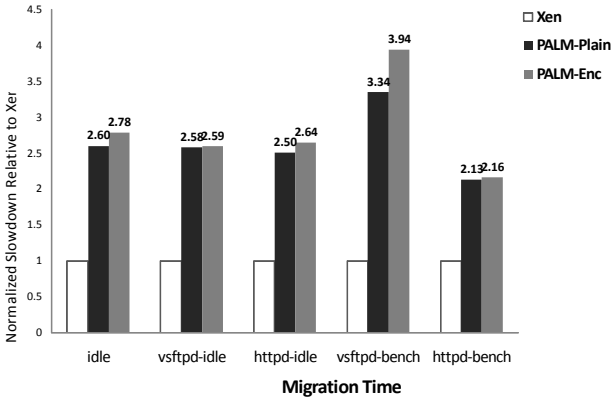

Figure 5. VM total migration time. The slowdown of the full functioned PALM is shown in the third bar. The second bar shows the slowdown of PALM with protection mechanisms disabled.

5. *httpd-bench*: the httpd server daemon is running while the client node is running Apache benchmark (ab) to connect and get files from the server.

The benchmark results for VM total migration time are shown in Figure 5 and downtime in Figure 6. We also provide a set of data for PALM system with protection disabled to identify the actual performance overhead incurred by PALM. The result shows that PALM incurs at most 3.94 times slowdown for total migration time and 7.92 times slowdown for downtime comparing to Xen live migration. And the slowdown comparing to PALM-plain is at most 1.18 times for total migration time and 3.67 times for downtime. Comparing to PALM-plain, PALM incurs negligible performance slowdown as the overhead is amortized among a long transmission phase (about 10s of seconds in our benchmarking).

As shown in Figure 6, the slowdown of PALM for VM downtime is at most 7.92 times comparing with Xen. The slowdown mainly comes from encryption and decryption operations in the stop-and-copy phase and resumption phase. As the total VM downtime is very short (about 10s to 100s of milliseconds in our benchmarking), this overhead is much more noticeable. However, the absolute overhead is still very slight (about 100s of milliseconds).

6 Related Work

VM migration has been explored in [18, 22] as one of the most desirable virtualization features. Migration of the entire operating system facilitates fault tolerance, load balance, and hardware maintenance for clusters and data cen-

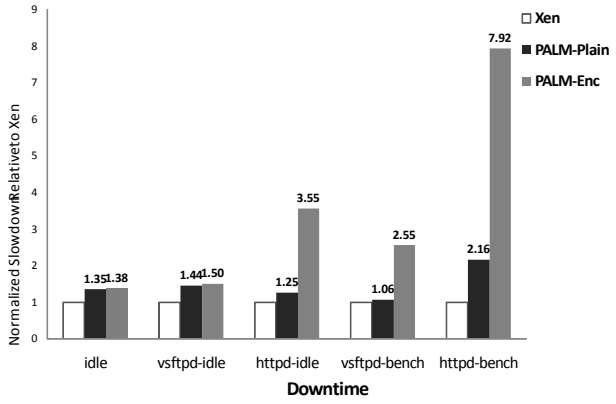

Figure 6. VM downtime. The slowdown of the fully functional PALM is shown in the third bar. The second bar shows the slowdown of PALM with protection mechanisms disabled. The slowdown of *httpd* is relatively high due to its large writable working sets.

ters. VM Live Migration [8, 10] aims to minimize the migration time and VM downtime for the VM migration and thus maximize the service availability. A pre-copy phase is used to transmit most of the runtime environments without stopping the VM and thus greatly reduced the downtime. Similarly, we target to a VM migration with minimal downtime. In addition, we also target to preserve the privacy and integrity protection to the protected execution environment during the migration. Instead of improving migration performance, our efforts are focused on retaining protection strength during VM migration while minimizing the performance degradation that security mechanisms bring.

VMMs have been advocated as the trusted computing base (TCB) for its small code size, constrained interface and privileged position. Small and manageable code size makes the VMM a more suitable TCB than the commodity operating system which is huge and error prone. As the most privileged software layer on the machine, the VMM can manage all the system resources and has full control of VMs.

Based on the Trusted Virtual Machine Monitor (TVMM), Terra [9] partitioned the platform into isolated VMs. Applications can run in the low-assurance open platform VM for compatibility, or high-assurance specially tailored VM for security. NGSCB [16] has a similar architecture that partition the the platform into half, one trusted and the other untrusted. The trusted operating system runs small, high-assurance programs that are responsible for security-sensitive operations and information. These programs cooperate with code running in the untrusted general purpose operating systems.

OpenTC [12] aims to provide the VMM-based isolated high-assurance VMs with open source software like Xen [1], a popular open source virtualization system and L4 [20]. OpenTC advocates separating privileges from the solo control VM (the ”domain 0” in Xen parlance) and put them in dedicated functional VMs. Murray et al. [14] improves Xen virtual machine system by aggregating privileged operations to multiple control VMs. IBM sHype [17] focuses on controlling information flow to providing the full-isolation and controlled resource sharing between VMs.

To further improve the practicality of the systems above, it is desired that the high-assurance VM can provide compatible execution environment with open platform operating systems. The dilemma of functionality and compatibility has been solved by a novel approach that manages to provide processes running on untrusted operating systems with high-assurance execution environments [4, 5, 7, 23]. The main idea is to use the trusted VMM to shield the contents of the protected processes from the guest operating system.

These systems are not designed with VM migration in mind. They impose both implementation and security challenges for VM live migration. Our efforts are focused on exploring the feasibility, security preservation, and implementation effort of VM live migration for such systems. While there are design and implementation differences in these systems, we expect similar issues are imposed in these systems and similar solutions could be applied.

Hardware support for Trusted Computing have been advocated. Such as Intel LT [11] and AMD Pacifica [19]. Their goal is to facilitate the construction of high-assurance execution environments and to protect sensitive information from software-based attacks by enhancing hardware components like processors, chipset, keyboard and graphics. These technologies can be used to assist the design and simplify our implementation for protected VM live migration.

7 Conclusions

In this paper we presented a security preserving VM live migration framework that provides live migration capability to VMs in VMM-enforced process protection systems. Our system guarantees the security strength is not lowered during and after the migration.

In the framework, we designed three modules for the privacy and integrity protection of the sensitive data, the metadata, and the live migration process itself. We identified several security vulnerabilities that may incur serious attacks during the process of migration and provided corresponding solutions.

To evaluate the design framework, we implemented a working prototype named PALM. The coding effort of the prototype system is moderate. And the performance eval-

uation shows that the performance degradation in the live migration is slight. We expect the framework can be similarly applied to other VMM-enforced process protection systems.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of SOSP'03*, pages 164–177. ACM, 2003.

[2] M. Bishop and M. Dilger. Checking for Race Conditions in File Accesses. *Computing Systems*, 2(2):131–152, 1996.

[3] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall. Cloud Computing. *White paper, IBM Corporation*, 2007.

[4] H. Chen, J. Chen, W. Mao, and F. Yan. Daonity–grid security from two levels of virtualization. *Information Security Technical Report*, 12(3):123–138, 2007.

[5] H. Chen, F. Zhang, C. Chen, R. Chen, B. Zang, P. Yew, and W. Mao. Tamper-Resistant Execution in an Untrusted Operating System Using A Virtual Machine Monitor. Technical Report 2007-08001, Parallel Processing Institute, Fudan University, Aug. 2007.

[6] P. Chen and B. Noble. When virtual is better than real. In *Proc. of the 8th IEEE Workshop on Hot Topics in Operating Systems*, May 2001.

[7] X. Chen, T. Garfinkel, E. Lewis, P. Subrahmanyam, C. Waldspurger, D. Boneh, J. Dwoskin, and D. Ports. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 2–13, 2008.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of NSDI*, pages 273–286, 2005.

[9] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proc. of SOSP'03*, pages 193–206, 2003.

[10] J. Hansen and A. Henriksen. Nomadic operating systems. *Master's thesis, Dept. of Computer Science, University of Copenhagen, Denmark*, 2002.

[11] Intel. LaGrande Technology Architectural Overview. Technical Report 252491-001, Intel Corporation, Sep. 2003.

[12] D. Kuhlmann, R. Landfermann, H. Ramasamy, M. Schunter, G. Ramunno, and D. Vernizzi. An open trusted computing architecture - secure virtual machines enabling user-defined policy enforcement. Technical Report RZ3655, IBM Research, 2006.

[13] W. McPhee. Operating system integrity in OS/VS2. *IBM Journal of Research and Development*, 13(3):230, 1974.

[14] D. Murray, G. Milos, and S. Hand. Improving Xen security through disaggregation. *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 151–160, 2008.

[15] M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Proc. USENIX ATC*, 2005.

[16] M. Peinado, Y. Chen, P. England, and J. Manferdelli. NGSCB: A Trusted Open System. In *Proc. ACISP*, pages 86–97, 2004.

[17] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. van Doorn, J. Griffin, and S. Berger. sHype: Secure Hypervisor Approach to Trusted Virtualized Systems. Technical Report RC23511, IBM Research, Feb. 2005.

[18] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation(OSDI)*, pages 377–390, Boston, MA, USA., December 2002.

[19] G. Strongin. Trusted computing using AMD Pacifica and Presidio secure virtual machine technology. *Information Security Technical Report*, 10(2):120–132, 2005.

[20] System Architecture Group. L4Ka::Pistachio Whitepaper. *White paper, University of Karlsruhe, Germany*, 2003.

[21] R. Ta-Min, L. Litty, and D. Lie. Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable. In *Proc. of OSDI'06*, pages 279–292, 2006.

[22] A. Whitaker, R. Cox, M. Shaw, and S. Gribble. Constructing services with interposable virtual hardware. *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI04)*, 2004.

[23] J. Yang and K. Shin. Using hypervisor to provide data secrecy for user applications on a per-page basis. *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 71–80, 2008.