A Brief Tutorial on Live Virtual Machine Migration From a Security Perspective

Diego Perez-Botero
Princeton University, Princeton, NJ, USA
diegop@princeton.edu

ABSTRACT

Virtualization has gained traction in a wide variety of contexts. The rise of Cloud Computing and the wide adoption of the OpenFlow API in computer networks are just a few examples of how virtualization has changed the foundations of computing. In general, the term "virtualization" refers to the process of turning a hardware-bound entity into a software-based component. The end result of such procedure encapsulates an entity's logic and is given the name of Virtual Machine (VM). The main advantage of this technique is that multiple VMs can run on top of a single physical host, which can make resource utilization much more efficient. Of particular interest are those VMs with high availability requirements, such as the ones deployed by cloud providers, given that they generate the need to minimize the downtime associated with routine operations.

VMs can deal with availability constraints much more gracefully than their physical equivalents. While physical hosts have to be powered down for maintenance, the VMs that they serve can migrate to execute on other physical nodes. It is also common to migrate VMs when load balancing is needed in the physical plane. The process of migrating VMs without any perceptible downtime is known as Live Virtual Machine Migration and is the topic of this paper. This non-trivial problem has been studied extensively and popular hypervisors (e.g. Xen, VMware, OpenVZ) have now put reasonable solutions to practice. After covering the *pre-copy* and *post-copy* approaches to live VM migration, a variety of design decisions will be discussed along with their pros and cons. Having set the necessary theoretical background, a security-focused survey will be carried out, documenting the state-of-the-art in Live VM Migration exploits and countermeasures.

1. SYSTEM VS PROCESS MIGRATION

The problem of *process migration* was thoroughly studied during the 90's. Unfortunately, the fact that applications are strongly connected with the OS by way of open sockets, file descriptors and other resource pointers makes process migration very difficult. In some cases (e.g. shared memory between processes), such migration is not even possible unless processes are partitioned *a priori* [1].

Migrating an entire OS with its applications is a much more manageable procedure, especially in the presence of a Virtual Machine Monitor (VMM). VMMs expose a narrow interface to the OS, so the entity to be migrated encapsulates most of the complexity. That is, the details of what is occurring inside the VM can be ignored during migration.

2. LIVE VM MIGRATION STRATEGIES

In this section, we will consider the most common setting for Live VM Migration: a clustered server environment. The three main physical resources that are used under such conditions are memory, network and disk [2]. While memory can be copied directly from one host to another, local disk and network interface migration are not trivial.

To be able to preserve open network connections and to avoid network redirection mechanisms, a VM should retain its original IP address after migration. If the migration is within the same LAN, which is the norm in a clustered server environment, this can be done by generating an unsolicited ARP reply advertising the new location for the migrated VM's IP [2].

Local disk migration should not be needed inside a server farm. Data centers use network-attached storage (NAS) devices, which can be accessed from anywhere inside the cluster. Thus, secondary storage doesn't have to be migrated with the VM. Consequently, **in a clustered server environment, the Live VM Migration problem is reduced to finding a way of consistently transferring VM memory state from one host to another**.

2.1 Memory Migration

Memory migration can be divided into three phases [3]:

- *Push phase*: The source VM continues running while certain pages are pushed across the network to the new destination. To ensure consistency, pages modified during this process must be resent.
- *Stop-and-copy phase*: The source VM is stopped, pages are copied across to the destination VM, then the new VM is started.
- *Pull phase*: The new VM starts its execution and, if it accesses a page that has not yet been copied, this page is faulted in across the network from the source VM.

Most migration strategies select either one or two of the above phases. While the *pre-copy* approach combines push with stop-and-copy, the *post-copy* approach combines pull with stop-and-copy.

2.2 Pre-Copy

As pointed out by [4], Xen[1] uses *pre-copy* as its live migration strategy. The *pre-copy* algorithm proposed by [2] uses an iterative push phase, followed by a minimal stop-and-copy. The iterative nature of the algorithm is the result of what is known as *dirty pages*: memory pages that have been modified in the source host since the last page transfer must be sent again to the destination host. At first, iteration i will be dealing with less dirty pages than iteration $i-1$. Unfortunately, the available bandwidth and workload characteristics will make it so that some pages will be updated at a faster rate than the rate at which they can be transferred to the destination host. At that point, the stop-and-copy procedure must be executed. A 5-step view of the *pre-copy* technique is shown in Figure 1:

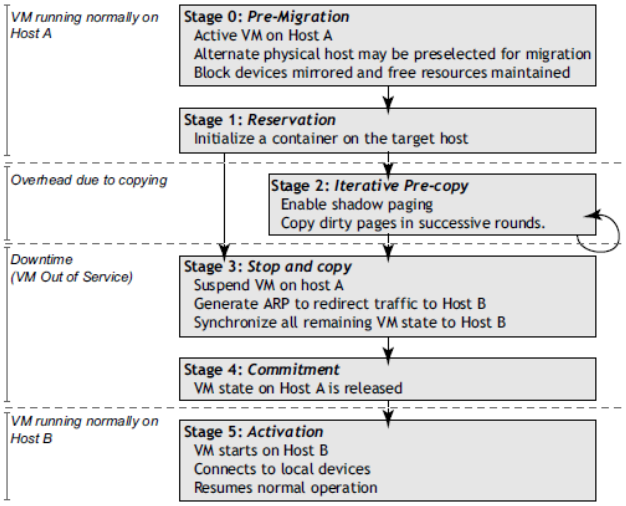

Figure 1: *pre-copy* algorithm. Taken from [2]

The stop-and-copy phase is when the CPU state and any remaining inconsistent pages are sent to the new host, leading to a fully consistent state. Determining the time to stop the pre-copy phase is non-trivial, since there exists a trade-off between *total migration time* and *downtime*. If it is stopped too soon, more data must be sent over the network while both the source and the destination are down, leading to a larger downtime. Nonetheless, if stopped too late, some time will be wasted on pages that are written too often and defeat any pre-copy efforts. As explained by [3], most server workloads exhibit a small, but frequently updated set of pages known as *writable working set (WWS)* or *hot pages*, that can only be transferred during the stop-and-copy stage. Depending on the workload characteristics, [2] registered downtimes with the *pre-copy* technique of only 60 ms and 210 ms with normal applications, and a worst-case 3.5 second downtime with an intentionally "diabolical" workload.

2.2.1 Availability Concerns

The *pre-copy* algorithm actively scans memory pages and sends them through the network. As such, CPU resource and bandwidth consumption should be monitored to minimize service degradation. A reasonable heuristic is to start off with low bandwidth usage, transferring the relatively static memory pages without any perceivable impact on quality of service. Afterwards, more bandwidth and CPU resources can be allocated to the migration process incrementally to be able to transfer frequently-updated pages. This would culminate with the maximum throughput (high performance impact) for a short period of time to reduce the *hot pages* to a minimum before the stop-and-copy phase begins.

[1]Xen is a very popular Open Source Type-I Hypervisor. http://www.xen.org

2.3 Post-Copy

Post-copy migration defers the memory transfer phase until *after* the VM's CPU state has already been transferred to the target and resumed there. As opposed to *pre-copy*, where the source host handles client requests during the migration process, *post-copy* delegates service execution to the destination host. In the most basic form, *post-copy* first suspends the migrating VM at the source node, copies minimal processor state to the target node, resumes the virtual machine at the target node, and begins fetching memory pages from the source over the network. Variants of post-copy arise in terms of the way pages are fetched. The main benefit of this approach is that each memory page is transferred *at most once*, thus avoiding the duplicate transmission overhead of pre-copy [5]. Figure 2 contrasts the pre-copy and the post-copy procedure timelines:

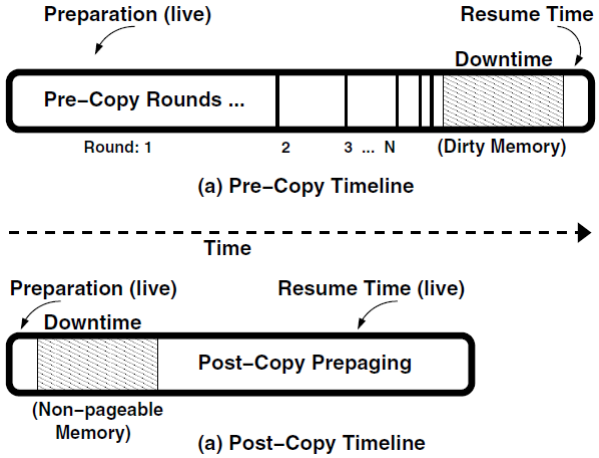

Figure 2: The timeline of (a)*pre-copy* vs (b)*post-copy* migration. Taken from [5]

There are three main ways of handling page fetching in *post-copy* schemes [5]:

- *Post-Copy via Demand Paging*: after the VM resumes at the target, page faults are serviced by requesting the referenced page over the network from the source node. Usually, this results in unacceptable *total migration times* and *application degradation*.
- *Post-Copy via Active Pushing*: a better way to tackle residual dependencies if to proactively push the pages from the source to the target, even as the VM continues executing at the target machine. Page faults can

be serviced with higher priority than the other pages being pushed.

- *Post-Copy via Pre-Paging*: this approach extends the active pushing technique by estimating the spatial locality of the VM's memory access pattern in order to anticipate the occurrence of major page faults. This way, the transmission window of the pages being pushed changes in real time with each new page fault to minimize *application degradation*.

2.3.1 Performance Concerns

As shown by [5], hybrid *post-copy* schemes provide lower total migration times than *pre-copy* because the *writable working set* issue is eliminated. Unfortunately, downtime is increased due to the way page faults are handled. Therefore, this migration strategy might be optimal for environments in which low network overhead is critical.

3. DIVISION OF RESPONSIBILITIES

When performing Live VM Migration, the operation can be either VMM-intensive or OS-intensive. Depending on the decision made, either the OS kernel or the VMM will need to be modified. Under some circumstances, the developer is forced to go for one option or the other. For example, OS-intensive procedures might not be possible if a closed-source OS is being used, such as Windows 7.

3.1 Managed Migration (VMM-based)

In case of managed migration, the migration is performed by the daemons running in the management VMs of the source and the destination. These daemons are responsible for creating a new VM on the destination machine, and coordinating transfer of live system state over the network. Let us consider the case in which live migration is being run in *pre-copy* mode. In the initial round, all the pages are transferred and subsequently only those pages that were dirtied in the previous rounds (as indicated by a dirty bitmap) are migrated. Xen uses shadow page tables to log dirty pages. The shadow tables are populated using guest page tables and reset after each phase of pre-copying. When pre-copy phase is no longer beneficial, a control message is sent to the OS to suspend itself in a state suitable for migration on host, and prepare for resumption on the destination. The dirty bitmap is scanned for remaining inconsistent memory pages, and these are transferred to the destination along with the VM's check-pointed CPU register state [3].

3.2 Self Migration (OS-Based)

In this design, no modifications are required to the VMM. The implementation aspects are present within the OS itself. The destination machine must run a migration stub to listen for incoming migration requests, create an appropriate empty VM, and receive migrated system state. No modifications are necessary for a source machine. Let us consider the case in which live migration is being run in *pre-copy* mode. Difficulty arises in transferring a consistent OS checkpoint, as the OS must continue to run in order to transfer its final state. The solution is to logically checkpoint the OS on entry into its final two stages of the stop-and-copy phase. In the pre-final stage, only migration is allowed and a shadow buffer is updated with the current dirty pages. In the last stage, contents of the shadow buffer are transferred to complete the migration [3].

4. VIRTUALIZATION OPTIONS

Virtualization offers many benefits such as live migration, resource consolidation, isolation, intrusion prevention/detection, and checkpointing. However, the overhead of virtualization cannot always be justified [6]. Regardless, Live Migration is a very valuable tool. For this reason, considerable research has been made to reap the benefits of native performance while still being able to conduct Live Migration.

4.1 Virtualized Environment

Virtualized environments are the common denominator when it comes to cloud computing, grid computing, and data centers in general. Native performance is sacrificed in order to accomplish higher resource utilization by executing various VMs on each host. Applications in this setting are usually I/O and network intensive, so virtualized devices suffice as no specialized operations are required. Most importantly, scalability is easily achievable in virtualized environments, since instances of the same service can be dynamically spawned or eliminated.

Live VM Migration is a very important operation in this context. It serves as the means through which the virtual-to-physical host mapping can be altered to achieve load balancing, energy efficiency, and easy hardware maintenance, among other administrative tasks.

4.2 No Virtualization

Virtualization technology adds an extra layer of abstraction with at least three unwanted effects [1]:

- *Capability lag*: VMMs typically expose "lowest common denominator" virtual devices to enhance portability. This leads to the inability to use the highest performance features of specialized physical devices, such as GPU-accelerated video decoding.
- *Additional software management*: VMMs can introduce additional complexity into software management (Xen has 200K lines of code in the hypervisor itself [7]). Virtualization typically does not reduce the total number of software components running on a system. Hence, there are more lines of code to manage, more patches to apply, etc.
- *Performance hit*: In many applications, virtualized performance is within an acceptable margin of native performance and, therefore, the additional layers of software introduced through virtualization are tolerated, but there are also cases where it is not. Kozuch et al. [1] show how a parallel robotics simulator suffers a 40% slowdown in a standard VM configuration.

Non-virtualized live migration is very challenging. The basic idea is the same as described in section 3.2 (Self Migration), but even migration to another machine with the same hardware presents new difficulties. Even non-deterministic aspects of the boot process (e.g. BIOS and/or OS might enumerate the devices in a different order or use different IRQs)

stand in the way of a successful migration [1]. Kozuch et al. [1] propose modifications to device drivers and the OS kernel that would make it possible to conduct live self-migration of non-virtualized OS instances, but the complexity of the operation makes it seem much more error-prone than its virtualized counterpart.

In terms of security, the presence of a hypervisor (virtualization) increases the possible attack vectors between co-hosted VMs. The NoHype architecture for cloud computing eliminates the hypervisor attack surface by enabling the guest VMs to run natively on the underlying hardware while maintaining the ability to run multiple VMs concurrently [7]. Such a feat is made possible by the very unique characteristics of cloud computing (e.g. pre-allocation of processor cores and memory resources, use of virtualized I/O devices, etc.). A live migration mechanism is yet to be proposed for the NoHype context, so non-virtualized live migration is a very relevant topic in cloud computing.

4.3 On-Demand Virtualization

Kooburat and Swift [6] show that virtualized and native execution are not necessarily mutually exclusive. Their *on-demand virtualization* proposal attempts to enable switches between both execution modes on-the-fly. They leverage the existing hibernation mechanism found in modern Operating Systems to capture system state and modify the *resume kernel* to boot up the machine in the other execution mode. Throughout the conversion, active connections remain open. Nonetheless, the downtime of their initial prototype is around 90 seconds, which is unacceptable. If a live native-to-virtualized conversion technique is developed, the non-virtualized live migration challenges detailed by [1] could be avoided.

5. THE SECURITY PROBLEM

Live VM migration includes a lot of state transfer through the network. During the procedure, protecting the contents of the VM state files is an important consideration as the volatile state being transferred may contain highly sensitive information like passwords and encryption keys. A secure channel is at times not enough for protection. Mutual validation among the hosts involved in the migration might even be a more important issue to be considered [3].

Live VM Migration, like any other network-bound process, is susceptible to network attacks such as ARP spoofing, DNS poisoning, and route hijacking. If an attacker somehow manages to place himself between the source and the destination host, he can then conduct passive (sniffing) or active (man-in-the-middle) attacks. The fact that the live migration procedure is usually carried out inside a LAN makes it even more likely for a network attack to be successful, especially in situations where different third-parties run their VMs inside the same network subnet, which is the case in cloud computing.

6. THREAT MODEL

Our *Trusted Computing Base (TCB)*, shown in Figure 3, is comprised by the hardware and the hypervisor, excluding its Live Migration Module. Given that the traditional hypervisor modules are trusted, co-hosted VMs do not pose a threat for the migration source and destination VMs, taking into account that the non-migration-related operations provided by the VMM cannot be used as an attack vector. Such assumption could be held true by employing a hardened hypervisor, such as HyperSafe [8].

The migration source and destination VMs are untrusted to each other, so mutual authentication and attestation mechanisms must be in place. Since the main usage scenario for Live VM Migration is in the cloud computing context, we also assume that other VMs inside the same network segment are untrusted third parties (potential attackers), but that the cloud provider is trusted. Last but not least, the communication medium (LAN) is taken to be untrusted and prone to interception by malicious parties. As a result, we focus our attention on remote LAN-bound threats. The migration module can be directly targeted by a remote party, but the migration process itself is also exposed to attacks, given that it takes place over the untrusted network infrastructure.

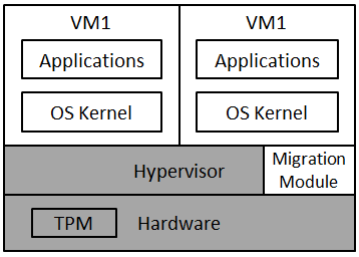

Figure 3: TCB for Live VM Migration. Gray components are trusted. White components are outside the TCB.

7. DETECTING MIGRATION OF VIRTUAL MACHINES

The first thing to consider in order to evaluate the viability of possible attacks to Live VM Migration is whether migration processes can be detected by an attacker located outside the source and destination hosts. That is, we must find a way to detect migration processes inside a network without relying on methods that are restricted to co-hosted VMs (e.g. the cache-based side-channel from [9]).

Konig and Steinmetz [10] show that the round-trip time (RTT) of ICMP packets is a promising metric for remotely detecting VM migration processes. By targeting a VM with ICMP packets, they can determine when that specific VM is migrating to another physical machine. As shown in Figure 4, a generalized increase in round-trip time is observed throughout the entire migration process. Additionally, a peak round-trip time at the beginning of the migration process is detected, as well as packet loss at the end. The packet loss at the end of the migration phase is caused by the virtual machine's CPU being stopped while its registers are transferred to the target machine [10]. When the VM being migrated is under high CPU load, the RTT peaks behave differently. This might be due to the fact that memory pages are being updated more frequently, leading to a greater amount of *hot pages*, which demands another network traffic pattern (a larger transfer at the end, before the

stop-and-copy phase).

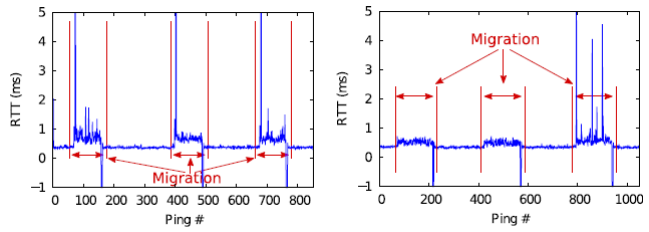

Figure 4: Remote detection of live migration process with low CPU load (left) and high CPU load (right). Taken from [10]

Konig and Steinmetz's results confirm the feasibility of conducting network attacks on Live VM Migration operations. If remote detection of migration initiation was not possible, ARP flooding and other aggressive network attacks would need to be activated for long periods of time, resulting in noticeable performance degradation and easy detection by Intrusion Detection Systems (IDS).

8. THREATS AND ATTACKS

Live VM Migration threats can be classified into 3 different classes [11]:

- *Control Plane*: The communication mechanisms employed by the VMM to initiate and manage live VM migrations must be authenticated and resistant to tampering. An attacker may be able to manipulate the control plane of a VMM to influence live VM migrations and gain control of a guest OS.
- *Data Plane*: The data plane across which VM migrations occur must be secured and protected against snooping and tampering of guest OS state. Passive attacks against the data plane may result in leakage of sensitive information from the guest OS, while active attacks may result in a complete compromise of the guest OS.
- *Migration Module*: The VMM component that implements migration functionality must be resilient against attacks. If an attacker is able to subvert the VMM using vulnerabilities in the migration module, the attacker may gain complete control over both the VMM and any guest OSes.

8.1 Control Plane

As a part of the control level threat, an attacker can manipulate the control realm of a VMM to arbitrarily initiate VM migration and thereby gain control of a guest OS. The possible loopholes at the control plane include [3]:

- *Incoming Migration Control*: By initiating unauthorized incoming migrations, an attacker may cause guest VMs to be live migrated to the attacker's machine and hence gain full control over guest VMs.
- *Outgoing Migration Control*: By initiating outgoing migrations, an attacker may migrate a large number of guest VMs to a legitimate victim VMM, overloading it and causing disruptions or a denial of service.
- *False Resource Advertising*: In an environment where live migrations are initiated automatically to distribute load across a large number of servers, an attacker may be able to falsely advertise available resources via the control plane. By pretending to have a large number of spare CPU cycles, the attacker may be able to influence the control plane to migrate a VM to a compromised VMM.

8.2 Data Plane

Melvin Ver [12] shows how packet sniffing of VMware VMotion's live migration process with widely-available tools like Wireshark[2] can reveal sensitive information in plain text, even when encryption is enabled. For example, the content of the files that the VM's legitimate user is currently viewing can be captured that way.

Figure 5 shows a logical view of a Man-in-the-Middle (MiTM) condition. As previously mentioned, there are a variety of network attacks that can generate the necessary conditions for an attacker to become part of the data path between the migration source and destination.

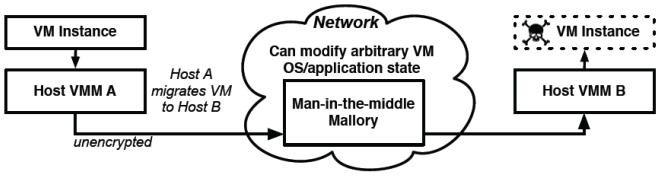

Figure 5: Man-in-the-Middle attack against a Live VM Migration. Taken from [11]

These attacks are not theoretical. Tools like Xensploit[3] work against Xen and VMware migrations. An example of this functionality was shown at the Black Hat DC Briefings 2008[4] by Oberheide et al.:

1. The attacker tries to gain root access to the target virtual machine via an SSH session before the VM migration happens:

```
jonojono@jonojono ~ $ ssh root@vm-test
Password:
Password:
Password:
Permission denied (keyboard-interactive).
jonojono@jonojono ~ $
```

Figure 6: Failed attempt at gaining root access through an SSH session

[2]http://www.wireshark.org
[3]http://blogs.iss.net/archive/XenSploit.html
[4]http://www.blackhat.com/html/bh-dc-08/bh-dc-08-main.html

2. The attacker intercepts the Live VM Migration and uses a hex editor to modify the SSH module's user authentication code before sending the memory pages to the destination host:

```
805daa0:    81 c4 9c 20 00 00    add    $0x209c,%esp
805daa6:    31 c0                xor    %eax,%eax
805daa8:    5b                   pop    %ebx
805daa9:    5e                   pop    %esi
805daaa:    5f                   pop    %edi
805daab:    5d                   pop    %ebp
805daac:    c3                   ret
805daad:    8d 76 00             lea    0x0(%esi),%esi
```

Figure 7: Hex editor view of authentication code

3. Attacker tries to gain root access again after VM migration is completed:

```
jonojono@jonojono ~ $ ssh root@vm-test
Last login: Thu Oct 18 15:18:37 2007 from jonojono.eecs.umich.edu
fjbox1 ~ # id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(a
py),20(dialout),25(at),26(tape),27(video),1006(vmware)
fjbox1 ~ #
```

Figure 8: Successful attempt at gaining root access through an SSH session

As seen in the example above, even if the VM and the VMM are secure against a particular threat under normal conditions, the migration procedure can render traditional security measures useless and compromise an entire system.

8.3 Migration Module

As the migration module provides a network service over which a VM is transferred, common software vulnerabilities such as stack, heap, and integer overflows can be exploited by a remote attacker to subvert the VMM. Given that VM migration may not commonly be viewed as a publicly exposed service, the code of the migration module may not be scrutinized as thoroughly as other code [11].

9. SECURITY MECHANISMS

Research in the area of VM migration mainly focused on optimizing migration performance through live migration. While the semantics and performance of live VM migration are well explored, the security aspects have received very little attention [13]. The threats as described in the previous section require that appropriate solutions be applied at every level. Mutual authentication of source and destination hosts is necessary for a secure migration. Also, migration *capabilities* and *access policies* should be introduced to allow administrators to manage migration policies [3].

9.1 vTPM

Trusted computing is an approach to building systems such that their integrity can be verified. It is based on the concept of transitive trust where initial trust in a hardware module is delegated to other system components [13]. The industry-standard trusted hardware module is the Trusted Platform Module (TPM). The full TPM specification by the TCG consortium can be found online[5].

[5]http://www.trustedcomputinggroup.org

Berger et al. [14] identify the requirements for a virtual TPM (vTPM) and propose a vTPM design that supports running vTPMs in memory or on a cryptoprocessor. This architecture has been implemented on the Xen hypervisor. Central to this architecture is a privileged VM (Dom0 in the case of Xen) dedicated to running vTPMs. This VM has access to the hardware TPM and coordinates all requests to it. This VM also runs a vTPM manager that manages all the communication between a VM and its vTPM. VMs can optionally be configured to use vTPMs. On starting a VM that is configured to have a vTPM, a corresponding vTPM instance is started as a user-space process in the privileged VM [13].

Figure 9 shows the vTPM architecture proposed by [14]. Each vTPM instance is assigned a unique 4-byte identifier that never leaves the privileged VM. This unique number is mapped to a unique interrupt (number) that is assigned to the VM. The VM uses this interrupt to communicate with its vTPM. The vTPM-to-interrupt_id mapping is stored in the XenStore[6] in the case of the Xen hypervisor along with the VM-to-vTPM instance mapping. On receiving a vTPM request, the backend driver prepends the instance number to the request using the mapping table. Communication is then managed using a split device driver model. The front end driver resides inside the VM and the back end driver in the privileged VM. To aid with the split device driver model, a special feature in Xen called the *xen-bus* is employed. The xen-bus enables a VM to map a portion of its memory as shared and allow the privileged VM to access it. Since communication happens using shared memory, unauthorized access to vTPMs by co-hosted VMs is not possible [13].

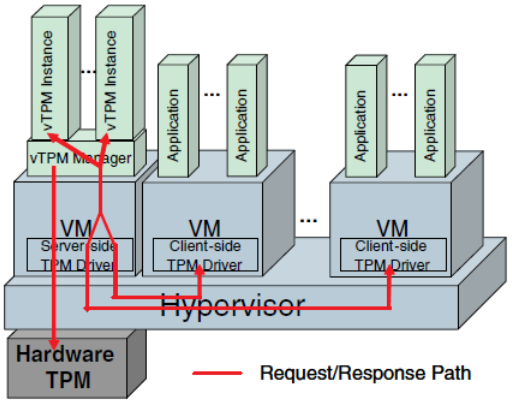

Figure 9: vTPM Architecture. Taken from [14]

9.2 Secure VM-vTPM Migration

The extension of trusted computing to virtualized systems using vTPMs allows applications in the VM to use the vTPM for secure storage and to report platform integrity. In order to ensure the correct operation of applications after migration, the vTPM must be migrated alongside the VM. Secure VM-vTPM migration is the name given to such operation.

[6]XenStore is an information storage space shared between domains.

Proposals for vTPM designs have been accompanied by proposals for vTPM migration [13].

9.2.1 *Berger et al.'s Proposed Protocol*

In [14], Berger et al. assume that the destination is trustworthy and propose the protocol shown in Figure 10 for migration between identical platforms. **They state that it can be used alongside live VM migration**. A migration-controlling process initiates the transfer by creating a new vTPM instance at the destination. Then, it creates a nonce and transfers it to the source in encrypted form. The key used for encryption is not clear. At the source, this nonce is used to lock the vTPM to prevent further changes to it. The vTPM is then encrypted using a newly generated symmetric key, which is in turn encrypted using the virtual Storage Root Key (vSRK) of the vTPM's parent instance. The encrypted state information includes keys, counters, any permanent flags, authorization and transport sessions, and data. A hash of each of the mentioned parts is added to an internal migration digest. The vTPM is deleted from the source and the encrypted state is transferred to the destination host along with the migration digest. The authors state that the vSRK of the parent vTPM instance is transferred to the destination using mechanisms applicable to migratable TPM storage keys[7]. At the destination, the received binary object is decrypted to extract the vTPM state. The digest is verified and, if no violations are detected, the vTPM is unlocked using the nonce and restarted. Since the vTPM keys are assumed to be independent from the hardware TPM keys, no key regeneration occurs.

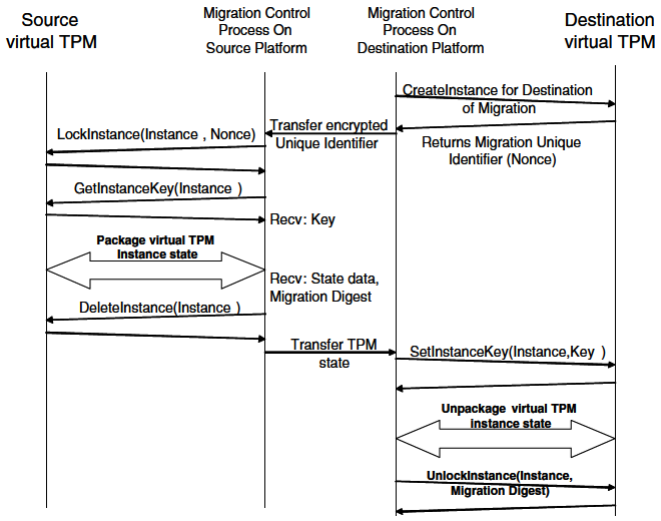

Figure 10: vTPM Migration as proposed by [14]

9.2.2 *Masti et al.'s Proposed Protocol*

Figure 11 shows a high-level view of Masti et al.'s secure VM-vTPM Migration protocol. The protocol proceeds in four phases. Initially, the source and destination mutually authenticate each other and agree upon confidentiality and integrity preserving cryptographic mechanisms for protecting the rest of the transfer process. Next, the source sends an attestation request to the destination to ensure that the VM is migrated to a secure platform. Having ensured the authenticity and integrity of the destination platform, the source then locks the VM and vTPM and transfers them securely using the previously agreed upon cryptographic primitives. Then, the destination checks the integrity of the received VM and its vTPM. If no violations are detected, the destination imports the VM-vTPM pair (which is implementation specific) and sends an acknowledgment to the source on success. Finally, in the last phase, the source deletes the migrated VM and vTPM to prevent duplication and informs the destination that the migration is complete. The destination then resumes the newly received VM and its vTPM. The various phases of the protocol can be linked to a single session explicitly (using a session identifier) or implicitly (by ensuring that each phase depends on any of the previous phases) [13].

The outlined protocol design assumes that the source and the migrating VM are trusted by the destination. This is reasonable in a setting where dynamic platform state measurement occurs ensuring that any malicious state changes to the source platform and the VM are detected and handled. Ideally, after the entire migration, the migrated VM should be able to report its new configuration to the destination on demand. This is meaningful only in a context where dynamic state measurements are enabled because otherwise, the VM could just replay its state before the migration [13].

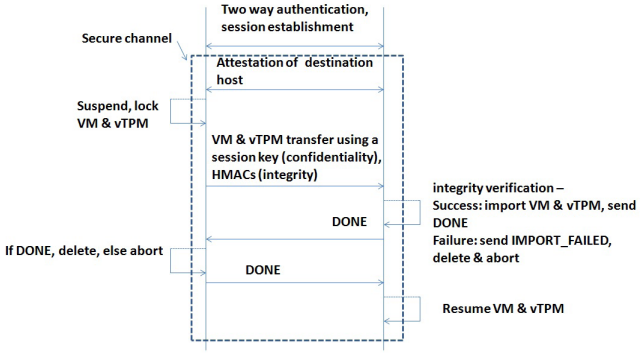

Figure 11: High-Level Outline of vTPM Migration as proposed by [13]

9.3 Secure Protocols and Live Migration

The semantics of VM migration are important because any changes to the VM should be synchronized with the vTPM. For the migration of a powered off VM or a suspended VM, only a secure transfer protocol is required. Live migration is more complex due to the need to synchronize VM changes with the vTPM, which makes the relative timing of vTPM and VM resumption at the destination important. Since some live migration techniques allow the VM to be started on the destination before it is stopped at the source, ensuring consistency between VM state and its vTPM is very difficult. Also, secure migration may not start the VM at the destination immediately after the transfer. This complicates the usage of live VM migration with vTPMs.

A secure Live VM Migration protocol with the security guarantees provided by the protocols covered in this section (au-

[7]Refer to the TPM specification (http://www.trustedcomputinggroup.org).

thentication, confidentiality, replay resistance, non-repudiation, atomicity, integrity, etc.) is yet to be seen in practice. While challenging, such end result does not seem impossible to attain, as pointed out by Berger et al [14].

10. CONCLUSIONS AND FUTURE WORK

Secure Live VM Migration has not received the amount of attention that it deserves. There is a clear trend towards delegating computation to the Cloud and, at the same time, Live VM Migration is becoming an everyday operation inside clustered server environments. Considering that security is defined by the "lowest common denominator", the existence of vulnerabilities in current migration mechanisms nullifies strong security guarantees provided by secure hardware and hypervisors (or lack thereof).

There is much ground for improvement when it comes to securing the two Live VM Migration network-related planes (control and data). A cloud computing context with the Trusted Computing Base (TCB) described in this paper could benefit from strategies used in other contexts. For example, VoIP and other next-generation multimedia services use a three-level security stack (signaling, key exchange and media) that could be ported to Live VM Migration to address network attacks. Schemes detailed in [15], such as S/MIME+MIKEY for secure signaling and key exchange, can be coupled with AES-encrypted live migration data to achieve confidentiality and integrity. In addition, the fact that the cloud provider is part of the TCB can be leveraged to deploy PKI-based mutual authentication protocols.

Secure Live VM-vTPM Migration and Non-Virtualized Live Migration are also interesting research topics. While the former would aid at hardening secure hypervisor architectures (e.g. HyperSafe [8]), the latter would help in eliminating the need for a hypervisor in the first place (e.g. NoHype [16]).

11. REFERENCES

[1] M. A. Kozuch, M. Kaminsky, and M. P. Ryan, "Migration without virtualization," in *Proceedings of the 12th conference on Hot topics in operating systems*, HotOS'09, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2009.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium pr & Implementation - Volume 2*, NSDI'05, (Berkeley, CA, USA), pp. 273–286, USENIX Association, 2005.

[3] S. Venkatesha, S. Sadhu, S. Kintali, and S. Barbara, "Survey of virtual machine migration techniques," *Memory*, 2009.

[4] P. S. Pisa, N. C. Fernandes, H. E. T. Carvalho, M. D. D. Moreira, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "Openflow and xen-based virtual network migration.," in *WCITD/NF* (A. Pont, G. Pujolle, and S. V. Raghavan, eds.), vol. 327 of *IFIP International Federation for Information Processing*, pp. 170–181, Springer, 2010.

[5] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," *Proceedings of the 2009 ACM SIGPLANSIGOPS international conference on Virtual execution environments VEE 09*, p. 51, 2009.

[6] T. Kooburat and M. Swift, "The best of both worlds with on-demand virtualization," in *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, HotOS'13, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 2011.

[7] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "Eliminating the Hypervisor Attack Surface for a More Secure Cloud," in *ACM Conference on Computer and Communications Security*, Oct. 2011.

[8] Z. Wang and X. Jiang, "Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity," in *In Proceedings of the 31st IEEE Symposium on Security and Privacy*, 2010.

[9] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, (New York, NY, USA), pp. 199–212, ACM, 2009.

[10] A. König and R. Steinmetz, "Detecting migration of virtual machines," in *Proceedings of the 10th Würzburg Workshop on IP: Joint ITG, ITC, and Euro-NF Workshop Visions of Future Generation Networks(EuroView2011)*, Julius-Maximilians-Universität Würzburg, Lehrstuhl für Informatik III, Aug 2011.

[11] J. Oberheide, E. Cooke, and F. Jahanian, "Empirical exploitation of live virtual machine migration," *Electrical Engineering*, no. Vmm, 2008.

[12] M. Ver, "Dynamic load balancing based on live migration of virtual machines: Security threats and effects," 2011.

[13] R. Jayaram Masti, "On the security of virtual machine migration and related topics," 2010.

[14] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. Doorn, "vtpm: Virtualizing the trusted platform module," in *In USENIX Security*, pp. 305–320, 2006.

[15] D. Perez-Botero and Y. Donoso, "Voip eavesdropping: A comprehensive evaluation of cryptographic countermeasures," in *Networking and Distributed Computing (ICNDC), 2011 Second International Conference on*, pp. 192 –196, sept. 2011.

[16] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "Nohype: virtualized cloud infrastructure without the virtualization," *SIGARCH Comput. Archit. News*, vol. 38, pp. 350–361, June 2010.