Empirical Exploitation of Live Virtual Machine Migration

Jon Oberheide, Evan Cooke, Farnam Jahanian
Electrical Engineering and Computer Science Department
University of Michigan, Ann Arbor, MI 48109
{*jonojono, emcooke, farnam*}*@umich.edu*

ABSTRACT

As virtualization continues to become increasingly popular in enterprise and organizational networks, operators and administrators are turning to live migration of virtual machines for the purpose of workload balancing and management. However, the security of live virtual machine migration has yet to be analyzed. This paper looks at this poorly explored area and attempts to empirically demonstrate the importance of securing the migration process. We begin by defining and investigating three classes of threats to virtual machine migration: control plane, data plane, and migration module threats. We then show how a malicious party using these attack strategies can exploit the latest versions of the popular Xen and VMware virtual machine monitors and present a tool to automate the manipulation of a guest operating system's memory during a live virtual machine migration. Using this experience, we discuss strategies to address the deficiencies in virtualization software and secure the live migration process.

1 INTRODUCTION

Recent advances in virtualization have made virtual machines an increasingly important research and operational area. Successful commercial ventures including VMware, XenSource, and Parallels have accelerated the adoption of virtualization software in many organizations. According to a recent IDC report [10], the number of virtualized servers will rise at a compound annual growth rate of over 40% from 2005-2010.

Live migration of virtual machines (VMs), the process of transitioning a VM from one virtual machine monitor (VMM) to another without halting the guest operating system, often between distinct physical machines, has opened new opportunities in computing [5]. Implemented by several existing virtualization products, live migration can aid in aspects such as high-availability services, transparent mobility, consolidated management, and workload balancing [7, 13].

While virtualization and live migration enable important new functionality, the combination introduces novel security challenges. A virtual machine monitor that incorporates a vulnerable implementation of live migration functionality may expose both the guest and host operating system to attack and result in a compromise of integrity.

Given the large and increasing market for virtualization technology, a comprehensive understanding of virtual machine migration security is essential. However, the security of virtual machine migration has yet to be analyzed. This paper presents a detailed investigation of the problem and explores three classes of threats to the migration process.

1. **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.

2. **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.

3. **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.

This paper explores attacks against live virtual machine migration in the context of these three threats. We present several practical attacks against the migration functionality of the latest versions[1] of the Xen [1] and VMware [12] virtualization products and develop a tool to automate the manipulation of a guest virtual machine's memory during live migration. Using this experience, we discuss strategies to address the deficiencies in virtualization software and secure the live migration process.

[1] At the time of writing, the latest version of Xen is 3.1.0 and the latest version of VMware is Virtual Infrastructure 3.

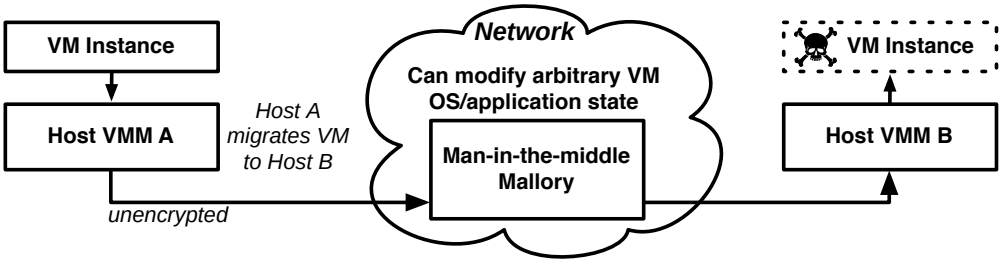

Figure 1: An example of a man-in-the-middle attack against a live VM migration.

2 BACKGROUND

Virtual machines and virtualization technology provide numerous technical and cost advantages [4]. However, the use of virtualization also introduces a novel set of security challenges [8]. In particular, there are novel concerns associated with virtual environments such as securing large numbers of virtual machines, securing a diverse range of operating systems and applications across virtual images, and securing mobile virtual machines that may move between different physical hosts and networks.

There are many ways in which a virtual machine can be moved from one VMM to another. Since virtual systems are typically stored as regular files on disk, the files associated with a halted system can be copied to another VMM using a network or using portable storage devices such as USB drives. In addition to the migration of halted virtual systems, many popular VMMs support *live migration*, the process of transitioning a VM from one virtual machine monitor to another without halting the guest operating system.

While various virtual machine monitors have different wire protocols for live migration, the underlying algorithms are similar. Live migration techniques [5, 9, 17] usually begin by copying memory pages of the VM across the network from the source VMM to the destination while the VM continues to run within the source VMM. This process continues as pages are dirtied by the VM. When the source VMM reaches a threshold and deems that no additional significant progress is being made in the transferring of dirty pages, it will halt the VM, send the remaining memory pages, and signal the destination VMM to resume the execution of the VM. The point at which the VMM decides to halt the source VM and copy the remaining pages is usually an implementation-specific heuristic that attempts to balance and minimize both the duration of migration and the downtime of the migrating VM. Other variations include the destination VMM resuming the VM early and requesting pages from the source VMM on-demand [14].

While one might assume that networks across which VM images are migrated are secure, this is not an entirely safe assumption anymore. As live VM migration becomes more common in many organizations, it is likely that the migration transit path may span multiple commodity networks and significant geographic distances. Indeed, virtual machines have been successfully migrated across continents with application downtimes as low as 1 to 2 seconds [15]. In addition, a compromised system inside a network employing live migrations can facilitate untrusted access to migrating VM images. The ability to view or modify data associated with live migrations or influence the migration services on source and destination VMMs raises several important security questions [11]. In the next section we elaborate on the some of these threats.

3 MIGRATION ATTACK CLASSES

In this section, we introduce three classes of threats to live virtual machine migration and describe several attacks applicable to each.

3.1 Control Plane

The communication mechanisms employed by the VMM to initiate and manage live virtual machine migrations must be authenticated and resistant to tampering. In addition, the protocols used in the control plane must be protected against spoofing and replay attacks. A lack of proper access control may allow an attacker to arbitrarily initiate VM migrations.

- **Incoming Migration Control**: By initiating unauthorized incoming migrations, an attacker may cause guest VMs to be live migrated to the attacker's machine and gain full control over guest VMs.
- **Outgoing Migration Control**: Similarly, 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 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.

As most existing VM products rely on manual intervention to initiate a migration, their access control mechanisms for the control plane are simplistic. For example, Xen employs a whitelist of host addresses allowed to perform migrations. However, as automatic migrations for load-balancing between many machines become more common, potentially across multiple administrative domains and between unpredictable host addresses, mechanisms for policing the control plane must be introduced and maintained.

3.2 Data Plane

The data plane across which VM migrations occur must also be secured and protected against snooping and tampering in order to protect the VM's state. An attacker may be able to logically position himself in the migration transit path using a number of techniques such as ARP spoofing, DNS poisoning, and route hijacking. With such a position, an attacker can conduct a man-in-the-middle attack as illustrated in Figure 1.

- **Passive Snooping**: Passive attacks against the data plane may result in leakage of sensitive information. By monitoring the migration transit path and associated network stream, an attacker can extract information from the memory of the migrating VM such as passwords, keys, application data, and other protected resources.
- **Active Manipulation**: One of the most severe attacks, an inline attacker may manipulate the memory of a VM as it is migrated across the network. Such a man-in-the-middle attack may result in a complete and covert compromise of the guest OS.

Even if proper encryption and identity management is used, it still may be possible for an attacker to gain valuable information from snooping on a migration stream. For example, an attacker may be able to uniquely identify guest VMs based on characteristics of the migration flow, such as size and duration, and identify the endpoint VMMs involved in the migration. This information may aid an attacker in targeting a later attack against a specific VM or critical infrastructure supporting that VM.

As we will demonstrate in the next section, popular VMMs deployed in production networks, such as Xen and VMware, fail to implement even simple data plane protection to ensure guest OS integrity during live migration and are vulnerable to attack.

3.3 Migration Module

The VMM component that implements live migration functionality must also be resilient to attacks. 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.

While such attacks are common across all types of software, special attention should be focused on the security of a VMM's migration module. As the VMM controls all the guest operating systems running within it, the severity of a VMM vulnerability is much greater than most normal software. If an attacker is able to compromise a VMM through its migration module, the integrity of any guest VMs running within the VMM, and any VMs that are migrated to that VMM in the future, may also become compromised.

As we will discuss in the next section, a brief audit of Xen's migration module resulted in multiple vulnerabilities that may compromise the VMM.

4 Implementation and Evaluation

We developed a tool, *Xensploit*, to perform man-in-the-middle attacks on the live migration of virtual machines. The tool operates by manipulating the memory of a VM as it traverses the network during a live migration. Xensploit is based on the `fragroute` [6] framework.

While its name is influenced by the first VMM (Xen) we applied it to, Xensploit is able to manipulate VMware migrations as well. In the following evaluations, we demonstrate attacks against the data plane class of both the Xen and VMware VMMs. In addition, we explore attacks against the migration module of Xen, resulting from multiple vulnerabilities discovered through an audit of Xen's migration code.

4.1 Attack Evaluation

4.1.1 Simple Memory Manipulation

To evaluate Xensploit, we performed a simple proof-of-concept manipulation during the live migration of a Xen VM. In Xen terminology, a host VMM is known as a `dom0` domain while guest VMs are known as `domU` domains. Our testbed consisted of three machines: the source `dom0`, the destination `dom0`, and a malicious node running Xensploit. We started a new guest `domU`, the domain to be migrated, within the source `dom0`. Inside `domU`, we executed a test process that simply prints a "Hello World" string to the terminal each second.

```
1180795919.260261: Hello World!
1180795920.270992: Hello World!
```

```
1180795921.281870: Hello World!
```

The live migration was then triggered to move `domU` from the source `dom0` to the destination `dom0`. As the memory pages of the running guest OS are transmitted over the network and pass through the malicious node running Xensploit, the "Hello World" string is replaced with our custom value.

```
1180795921.920290: Xensploited!
1180795922.932574: Xensploited!
1180795923.942636: Xensploited!
```

In a matter of seconds, the guest OS has been seamlessly migrated to the destination `dom0`. As expected, Xensploit's man-in-the-middle attack was successful and the memory of our test process has been manipulated, resulting in the new string being printed to the terminal of the guest OS within the destination `dom0`.

4.1.2 sshd Authentication Manipulation

As a more advanced and practical example of our tool, we instrumented Xensploit to manipulate the memory of the *Secure Shell daemon* (sshd) process of a guest VM during a live migration. Instead of performing the attack on Xen again, we switched our deployment to VMware Virtual Infrastructure [16] to demonstrate Xensploit's flexibility. Our testbed consisted of four machines: the source and destination VMMs both running VMware ESX Server 3.0.1, a management node running VMware Virtual Infrastructure Client/Server 2.0.1 to manage the VMMs and initiate the migration, and the malicious node running Xensploit.

Before initiating the migration, we attempted to ssh to the guest OS running within the source VMM. The sshd process was configured to only allow authentication of the type `PubkeyAuthentication`. As our public key was not in the root user's `.ssh/authorized_keys` file, access was denied.

```
jonojono@apollo ~ $ ssh root@testvm1
Permission denied (publickey,keyboard-interactive).
```

We then initiated the live migration via the Virtual Infrastructure Client and performed the man-in-the-middle attack using Xensploit. Specifically, the in-memory object code of the sshd process, originating from `user_key_allowed2` function in *auth2-pubkey.c*, is manipulated during migration to successfully authenticate any incoming ssh logins.

As seen below, after Xensploit's attack, our attempt to ssh to the VM succeeds due to the manipulated sshd process.

```
jonojono@apollo ~ $ ssh root@testvm1
Last login: Tue Jun 5 19:25:19 2007 from localhost
testvm1 ~ #
```

These examples of manipulating the memory of the guest OS are just a small subset of the possible attacks designed to evaluate our Xensploit tool. Much more insidious man-in-the-middle attacks can be performed such as transparently slipping a rootkit into kernel memory during the live migration.

4.1.3 Xen Migration Module

While exploring the Xen source code, we discovered multiple issues which fall into the migration module class of live migration threats. The vulnerabilities are present in Xen's VMM migration routines, specifically the code in *xen-3.1.0-src/tools/libxc/xc_domain_restore.c*, which is used to restore an incoming migration to operational state.

As we previously mentioned, exploitable vulnerabilities in a VMM are especially serious as the integrity of all the currently hosted VMs, and any VMs migrated to the exploited VMM in the future, may be compromised. One vulnerability exploits an integer signedness issue resulting in a stack overflow, and yet another involves a `malloc()` integer overflow resulting in a potential heap overflow. These two issues may allow a remote attacker to achieve privileged code execution and completely compromise the Xen VMM and host machine.

The vulnerabilities have been reported to the XenSource development team and will be resolved in an upcoming release. Further details regarding the specific routines affected can be found in Appendix A.

5 DISCUSSION

This paper has empirically demonstrated how two of the most popular and widely deployed VMMs, Xen and VMware are vulnerable to practical attacks targeting their live migration functionality. These threats are cause for concern and require that appropriate solutions be applied to each class of live migration threats.

In order to support the secure migration of virtual machines, mutual authentication of the source and destination VMMs, as well as any management agents, must be performed to protect the control plane communications. Flexible access control policies should allow administrators to manage migration privileges. The data plane over which the migration occurs must be secured against snooping and manipulation of the state of migrating VMs. Solutions include protecting the transit path using encryption or using a separate physical or virtual network to partition and isolate migration traffic from potential threats. While encrypting the migration may seem like a trivial solution to implement, effectively maintaining a public key infrastructure to ensure mutual authentication will add significant complexity to VM management software and may be infeasible for certain deployments. Finally, robust secure coding methods such as in-

put validation, privilege separation, type-safe languages, and frequent code audits can help reduce the chance of compromises of the migration module of a VMM.

Traditionally, a breach in network security results in a compromise of *data* integrity. However, when dealing with virtual machine migration of full operating systems, a breach in the network can result in not only a compromise of *data*, but also *host* integrity. This fundamental shift in the threat model of a network may require re-thinking existing access control and isolation mechanisms. Fine-grained network access control systems such as SANE [3] may provide sufficiently flexible policies to address such a threat model. Beyond VLANs, complete virtualization of network resources [2] throughout the stack may allow isolation for secure migrations and may provide an inherent complement to host virtualization.

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. *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2003.

[2] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: realistic and controlled network experimentation. *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–14, 2006.

[3] M. Casado, T. Garfinkel, A. Akella, M.J. Freedman, D. Boneh, N. McKeown, and S. Shenker. SANE: A Protection Architecture for Enterprise Networks. *Proceedings of the 15th USENIX Security Symposium*, 2006.

[4] P.M. Chen and B.D. Noble. When virtual is better than real. *Proceedings of the 2001 Workshop on Hot Topics in Operating Systems (HotOS)*, pages 133–138, 2001.

[5] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, and C. Limpach. Live Migration of Virtual Machines. *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation*, 2005.

[6] Dug Song. fragroute. `http://www.monkey.org/~dugsong/fragroute`.

[7] A. Ganguly, A. Agrawal, P.O. Boykin, and R. Figueiredo. WOW: Self-Organizing Wide Area Overlay Networks of Virtual Workstations. *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 30–41, 2006.

[8] T. Garfinkel and M. Rosenblum. When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments. *10th Workshop on Hot Topics in Operating Systems*, 2005.

[9] J.G. Hansen and E. Jul. Self-migration of operating systems. *Proceedings of the 11th workshop on ACM SIGOPS European workshop: beyond the PC*, 2004.

[10] IDC. Virtualization and multicore innovations disrupting the worldwide server market. `http://www.idc.com/getdoc.jsp?containerId=prUS20609907`, March 2007.

[11] M. Kozuch, M. Satyanarayanan, I. Res, and PA Pittsburgh. Internet suspend/resume. *Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on*, pages 40–46, 2002.

[12] M. Nelson, B.H. Lim, and G. Hutchins. Fast Transparent Migration for Virtual Machines. *Proceedings of the USENIX 2005 Annual Technical Conference*, 2005.

[13] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. *IEEE Int'l Conf. on Autonomic Computing (ICAC'06)*, 2006.

[14] C.P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M.S. Lam, and M. Rosenblum. Optimizing the Migration of Virtual Computers. *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[15] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath, and P. Yonghui Wang. Seamless live migration of virtual machines over the MAN/WAN. *Future Generation Computer Systems*, 22(8):901–907, 2006.

[16] VMware. Virtual infrastructure 3. `http://www.vmware.com/products/vi`.

[17] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*, 2007.

APPENDIX A

This appendix delves into the specific details of the vulnerabilities discovered in the migration module of Xen.

Vulnerability #1

The first vulnerability occurs early in the migration restoration code while reading in Xen's physical-to-machine (p2m) memory mapping from the wire during an incoming migration. We begin below at line 437 where the signed integer `j` is read in from the socket on which the incoming migration is occurring.

```
int j, nr_mfns = 0;
...
if (!read_exact(io_fd, &j, sizeof(int)))
{
    ERROR("Error when reading batch size");
    goto out;
}
```

After `j` is read in from the wire, several sanity checks are performed to ensure its validity. The following code checks whether `j` is -1, -2, 0, or greater than `MAX_BATCH_SIZE` and will handle the appropriate exceptions if `j` is one of these specific values.

```
if (j == -1)
    ...
if (j == -2)
    ...
if (j == 0)
    ...
if (j > MAX_BATCH_SIZE)
    ...
```

While `j` is checked for a maximum bound above `MAX_BATCH_SIZE` to ensure it does not overflow the size of `region_pfn_type`, `j` is not checked for a negative value other than -1 and -2. Any other negative value for `j` would evade the sanity checks and be passed as `j * sizeof(unsigned long)` to `read_exact()`.

```
if (!read_exact(io_fd, region_pfn_type, j*sizeof(unsigned long)))
{
    ERROR("Error when reading region pfn types");
    goto out;
}
```

Casting the `j * sizeof(unsigned long)` argument into a `size_t` in `read_exact()` will result in a large and controlled amount of data being read from the wire into `region_pfn_type` overflowing its bound of `MAX_BATCH_SIZE` and allowing an attacker to write arbitrary data onto the stack of `xc_domain_restore`.

Vulnerability #2

We will walk through the second vulnerability starting with the affected code of *xc_domain_restore.c* around line 906. As seen below, the code begins with a call to the `read_exact()` which reads 4 bytes into `count`, an unsigned integer, from `io_fd`, the socket on which the migration is occurring.

```
unsigned int count;
unsigned long *pfntab;
int nr_frees, rc;

if (!read_exact(io_fd, &count, sizeof(count)))
{
    ERROR("Error when reading pfn count");
    goto out;
}
```

The attacker-controlled `count` variable is used to determine how many unsigned long elements should be allocated for the `pfntab` structure. The value passed to `malloc()` is the result of `sizeof(unsigned long) * count`. Unfortunately, when large values of `count` are supplied, `sizeof(unsigned long) * count` will overflow the maximum bound of a `size_t` and wrap around, resulting in a very small amount of memory being allocated for `pfntab`.

```
if (!(pfntab = malloc(sizeof(unsigned long) * count)))
{
    ERROR("Out of memory");
    goto out;
}
```

So far, an attacker has provided a large `count` value and was able to influence a significantly smaller `pfntab` allocation than expected. In the next snippet, a similar integer overflow occurs when calling `read_exact()`, resulting in a small amount of data being read in from the `io_fd` socket, equal to the size of the allocated `pfntab`. This code snippet is only significant in that the attacker must remember to supply an adequate amount to be read into `pfntab`.

```
if (!read_exact(io_fd, pfntab, sizeof(unsigned long)*count))
{
    ERROR("Error when reading pfntab");
    goto out;
}
```

The last code snippet is where the effects of the attack are observed. Since the attacker has full control over `count` and `pfntab`, it follows that he also has control of `i` and `pfn`. Since the `pfntab` allocation is much smaller than expected, the `count` iterations will push `pfntab` passed its allocated bounds.

```
nr_frees = 0;
for (i = 0; i < count; i++)
{
    unsigned long pfn = pfntab[i];

    if (p2m[pfn] != INVALID_P2M_ENTRY)
    {
        pfntab[nr_frees++] = p2m[pfn];
        p2m[pfn]  = INVALID_P2M_ENTRY;
    }
}
```

With sufficient control of the values of `p2m`, a heap overflow past the allocated bounds of `pfntab` may be possible, resulting in a compromise of the Xen VMM.