A Framework for Security Context Migration in a Firewall Secured Virtual Machine Environment

Zahra Tavakoli*, Sebastian Meier, and Alexander Vensmer

Institute of Communication Networks and Computer Engineering (IKR)
University of Stuttgart
Pfaffenwaldring 47, 70569 Stuttgart, Germany
{tavakoza,sebastian.meier,alexander.vensmer}@ikr.uni-stuttgart.de

Abstract. Current virtualization technologies enable hosting of a large number of Virtual Machines (VMs) on a common physical host. The hypervisor interconnects these VMs via Virtual Networks (VNs). These VNs underlie the same security requirements as physical networks. Network elements such as stateful firewalls contribute in enforcing this security. With the advent of stateful firewalls on the hypervisor level, a new challenge arises when it comes to VM migration. Not only the VM itself, but also the associated Security Context (SC) has to migrate. Current open-source hypervisors do not address this issue. In this paper we present the architecture and implementation of our framework for migrating SC along with VMs.

Keywords: Virtualization, Firewall, Security, Migration

1 Introduction

Virtualization of end systems is becoming more and more popular. Optimizations such as Kernel SamePage Merging [10] enable hosting of a very large number of VMs on a common host. Due to the increasing number of VMs on a single physical host, VN topologies inside a hypervisor become more and more sophisticated.

Consequently, current hypervisor implementations support VN devices such as virtual switches and virtual routers. In an enterprise scenario those VNs are expected to satisfy the same requirements regarding monitoring and management as their physical counterparts. Vendors such as CISCO address this need by providing VN equipment [1] that provides the same configuration and management interfaces as physical network equipment. By doing so, the barrier for including more intelligent and elaborated network elements into a VN is lowered. Those network elements can perform different functions. We will focus on stateful firewalling in a VN in the following. Stateful firewalls provide means to protect a network from malicious or unwanted traffic.

In contrast to traditional networks, where machines and their attachments to the network are rather fixed, virtualization offers the opportunity to migrate

* At the time of writing, Zahra Tavakoli was a student at the IKR.

VMs between different physical hosts. In this context, managing stateful firewalls becomes more complex as security policies that are associated with a VM have to migrate together with the VM itself. This is most challenging for stateful firewalls, as in addition to static security policies dynamic state information has to migrate along with the VM. In this paper we present the architecture and the implementation of a framework that extends the KVM hypervisor to migrate stateful SC along with a VM.

The remainder of this paper is structured as follows. Section 2 gives an introduction to stateful firewalling and virtualization of end systems. Section 3 presents current research activities regarding VN security. In Section 4 we introduce our architecture for migrating SC within a virtual environment. In Section 5 we present our implementation of the architecture. Finally, Section 6 concludes this paper.

2 Background

In the following, we introduce stateful firewalls and virtualization technologies. In particular, we present implementations for the Linux operating system. Furthermore, we give a short overview on VM migration.

2.1 Stateful Firewall

Stateless firewalls perform policing by filtering packets according to static filter rules. These rules typically specify filter criteria that are applied to headers of packets that traverse the firewall. Stateful firewalls enhance this mechanism by relying on connection tracking. Connection tracking maintains state information about active connections and sessions. Tracking is typically done by observing network traffic. For instance, observing a TCP three way handshake will create state information within the connection tracking internal connection table. Stateful firewalls can access this information.

In comparison to stateless firewalls, this allows stateful firewalls to specify additional filter criteria. For instance, a stateful firewall may only forward TCP packets that are part of a TCP three way handshake or belong to an already established TCP connection. In the case of a Linux machine, filtering and connection tracking are both realized by the Netfilter framework [6] in the kernel. User space tools such as conntrack-tools and iptables allow manipulating connection tracking entries and firewall filter rules.

2.2 Virtualization in Linux

Virtualization is an approach for sharing common resources of a physical host between multiple VMs. One of the core tasks in virtualization is to isolate VMs against each other. This is done by the hypervisor. The hypervisor schedules the execution of VMs and mediates the access to the resources of the host system. Resources typically include memory, disk space, CPU and network devices.

There are several well-known implementations of hypervisors: Xen [9], VMware ESX [8] and KVM [3]. In the following we will focus on KVM. It is an open source hypervisor for Linux. KVM enables the Linux kernel to perform hypervisor functionality such as VM scheduling and memory management [14]. KVM relies on QEMU [7] for emulating a virtual environment, including CPU architecture and peripheral devices. The libvirt API [5] provides means to manage, control and monitor VMs that are hosted by KVM and QEMU. We use libvirt in our solution to manipulate the migration process of VMs and added extensions for transferring the SC.

2.3 Virtual Machine Migration

VM migration provides means to move a VM from one host to another. There exist several incentives to migrate VMs. For instance, a VMs might be moved from one physical host to another to shut down a physical machine and perform hardware maintenance or save energy.

Migration approaches can be divided into two categories: offline migration and live migration. For offline migration, a VM is suspended on the source host before being migrated to the destination. After a comparatively long shutdown time it will be resumed on the destination host. For live migration, the VM will be migrated with almost no interruption of the processes within it. Compared to offline migration, live migration provides many advantages and is therefore the commonly used VM migration approach. Thus, we focus on live migration scenarios in the following.

Regarding the network, one can additionally introduce two categories for VM migration: migration within the same subnetwork and migration across subnetworks. Migration across a subnetwork may trigger a change of a VM's IP address. Furthermore, it may require rerouting of IP packets that are exchanged with a communication partner. Currently, we consider such a scenario out of scope and focus on migration scenarios where the IP and MAC addresses of a migrating VM don't change.

For our migration scenario, we additionally assume that VMs may access a common storage network. This storage network contains the file systems of the VMs. Therefore, there is no need to migrate the file system of a migrating VM. Consequently, only state information such as memory pages, CPU register contents, et cetera are subject to migration. KVM already supports this kind of live migration [4]. However, KVM doesn't take migration of SC such as firewall rules and connection tracking state into account.

3 Related Work

Security vulnerabilities and possible countermeasures in VNs have been introduced by Dawoud et al. [11]. They performed a study regarding the security of Infrastructure as a service (IaaS) components. The result of their research is a

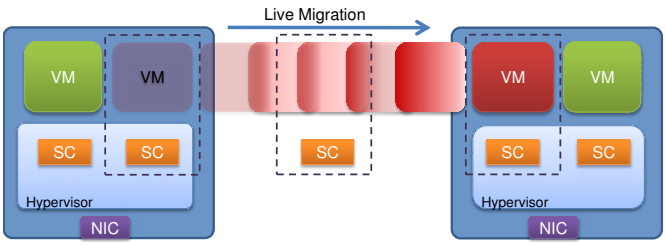

Fig. 1. SCM Principle

proposal for a Security Model for IaaS. Qi-Guang Miao et al. [13] presented another solution for improving security of VNs. They created an emulated network device called VSFilter. It consists of a virtual switch and a network packet filter. Hanquia et al. [15] suggested a novel VN model to provide isolation between VM traffic running on the same physical host. For isolation they relied on a hypervisor based firewall.

Although the above mentioned approaches suggested or relied on a hypervisor based firewall, they didn't address the challenges of SC migration. Xianqin, Chen et al. [16] acknowledged this challenge in context of stateful firewalls. They outlined why it is important to transfer the state of the connections that are related to a migrating VM. Furthermore, they modified the XEN live migration tool to migrate connection tracking information between hypervisors. However, they neither migrated the stateful firewall policies nor other SC information.

4 Security Context Migration Architecture

In contrast to previous work, we consider SC in a more general sense. We consider SC to be varying and its actual composition to be scenario dependent. For instance, SC may include firewall filtering rules in addition to connection tracking information. In more sophisticated scenarios additional elements such as IPSec [12] state information may be added to the SC.

Currently, we assume that SC information doesn't require modification during SC migration. In particular, we assume that the IP and MAC address of a migrating VM won't change, even if migrating to a different IP subnetwork. This assumption usually holds true, as VMs are typically attached to the network via Layer 2 over Layer 3 encapsulation or other tunnel solutions.

Regarding the SC itself, we differentiate between static and dynamic SC. We define SC as static, if it doesn't change during the life-cycle of a VM. Dynamic SC on the other hand may change constantly if the VM isn't suspended or stopped.

This kind of flexible SC requires an elaborate framework that is able to cope with the dynamic nature of the SC. In the following, we present the architecture of our flexible framework to handle and migrate SC that is associated with VMs. Figure 1 outlines the principle Security Context Migration (SCM) idea.

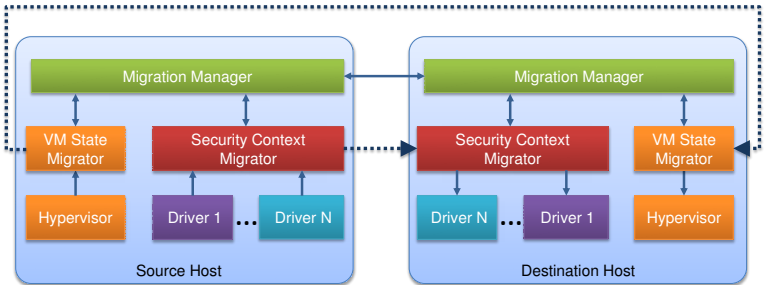

Fig. 2. SCM Framework Architecture

4.1 Structural View

Figure 2 shows the architecture of the SCM framework. We will introduce its components in the following.

Migration Manager: The Migration Manager coordinates the migration of VM state and SC state information between the importing and exporting host. In particular, the migration manager ensures a consistent snapshot of VM state and SC state. For this, the Migration Manager interacts with the VM State Migrator and SC Migrator, as detailed in subsection 4.2.

VM State Migrator: The VM State Migrator component interacts with the hypervisor. It is responsible for coordinating the transfer of VM state information, such as memory pages for instance. For this, it relies on the functionality provided by the underlying hypervisor to migrate the VM from the source to the destination host.

Security Context Migrator: Whenever a VM migrates, the SC Migrator extracts VM related SC information on the source host. The SC Migrator on the destination host is responsible for importing the extracted SC information. For exchanging SC information the involved SC Migrators establish a communication channel.

Drivers: Drivers are responsible for handling a subset of the SC. They abstract from implementation or operating system specific interfaces and data structures. For this, drivers provide a generic interface towards the SC Migrator for importing and exporting SC information. Furthermore, a driver may encapsulate the handled state information into a common data structure. Regarding SC handling, this abstraction is a key enabler for a flexible and extensible SC migration framework.

4.2 Behavioral View

The SCM framework extends the current KVM migration workflow to migrate the SC that is associated with a VM. The grey boxes in figure 3 indicate extensions to the current KVM workflow. We divide SC migration into two tasks.

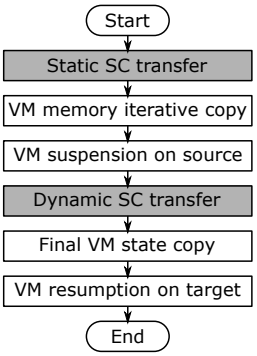

Fig. 3. VM Live Migration Workflow

The task *Static SC transfer* is responsible for migrating static SC. The task *Dynamic SC transfer* is responsible for migrating dynamic SC. We define two tasks to migrate as much SC information as possible while the VM is still running on the source host. This ensures that only dynamic SC state has to be migrated while the machine is suspended. This approach keeps additional VM downtime caused by SC migration at the bare minimum.

Figure 4 depicts the coordinated migration of a VM together with the associated SC. For simplicity we only show the SC Migrator and VM Migrator component. The migration is initiated by an external trigger. The trigger denotes which VM is subject to migration. Upon receiving this trigger, the SC Migrator of the source host starts extracting and transferring the VM related static SC information. The SC Migrator on the destination host imports the information and sends a positive reply. As soon as this reply is received on the source host, the SC Migrator triggers the VM Migrator to begin with the VM migration. Then, the VM Migrator begins to transfer the VM state incrementally to the destination. This tasks ends as soon as the VM state migration is complete. At this point, the source host suspends the VM and notifies the SC Migrator. Now the SC Migrator transfers the remaining dynamic SC information to the destination host. Once the reception of the dynamic SC has been acknowledged by the destination host, the VM Migrator on the source host will complete the VM migration. This is done by sending a trigger to the destination host to resume the VM.

5 Implementation

In this section, we will present the prototypic implementation of our SCM framework. For portability, we implemented the framework in Java. We rely on the external Libvirt library, which provides an abstraction layer between our framework and the underlying hypervisor. This kind of abstraction allows us to interact with a great variety of hypervisors using a common codebase. For testing the prototype, we focused on the KVM hypervisor of the Linux operating system.

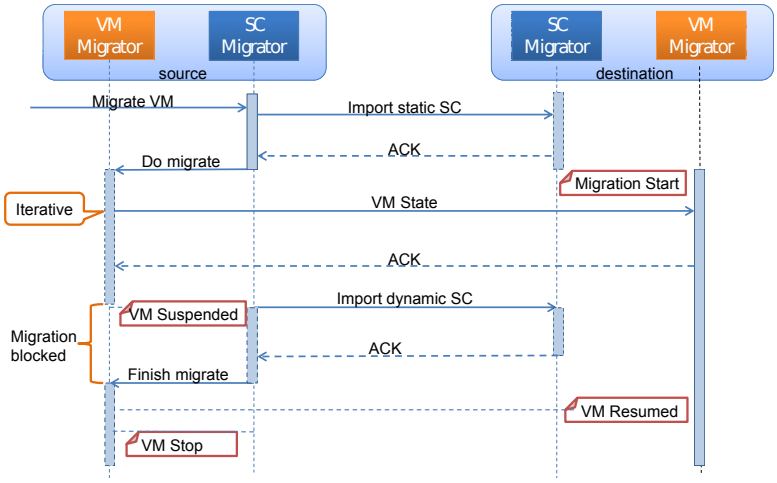

Fig. 4. Sequence Diagram for VM and SC Migration

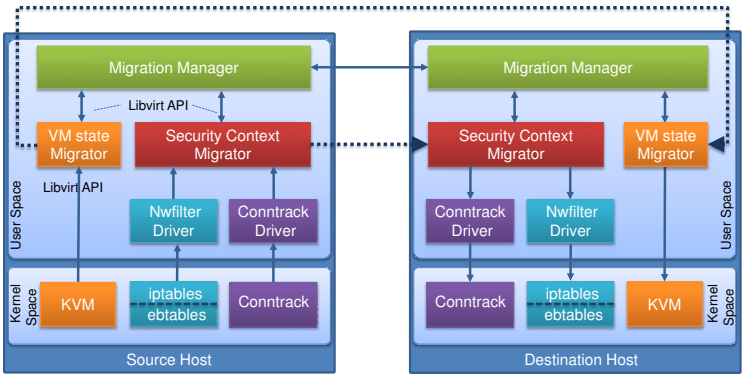

Fig. 5. SCM Framework Implementation

5.1 SCM Prototype

Figure 5 shows the structure of our implementation and its interaction with external components. In the following, we will provide some details on the components of the implementation and the integration of the prototype.

Migration Manager: The Migration Manager is responsible for keeping track of ongoing VM migration processes. For this, it interacts with the VM Migrator, which notifies the Migration Manager about the progress of a VM's state during migration. Based on this information, the Migration Manager coordinates VM and SC migration. For instance, when the Migration Manager is notified that a migrating VM is suspended on the source host, it triggers the dynamic SC transfer via the SC Migrator.

VM Migrator: KVM interacts with external components such as Libvirt or the QEMU monitor for VM migration. For VM migration we rely on an appliction called *virsh* which is part of the Libvirt toolset. To synchronize the VM and SC migration we have to interact with the Libvirt API. We added two additional hooks to the API so that we can realize the workflow that is depicted in figure 3.

The first hook is required to migrate the static SC information before the migrating VM is suspended on the source host. For this, we extended the virsh code to notify the SC Migrator before triggering the VM migration via the Libvirt API.

The second hook ensures that all dynamic SC information is imported on the destination host, before the machine is resumed there. We added this hook to the Libvirt API so that the SC Migrator is informed when the machine is suspended on the source host. Furthermore, the Libvirt API on the destination host won't resume the machine until notified by the SC Migrator. We implemented the inter-process communication via UNIX domain sockets.

SC Migrator: We subdivided the SC Migrator into two components. The first component is responsible for interacting with the Drivers. It coordinates the import and export of static and dynamic SC information.

The second component maintains communication sessions with SC Migrator components on remote hosts. For this, it establishes a TCP connection. The destination address is automatically derived from the VM migration target. For coordination and SC transport we defined a simple type-length-value protocol.

Divers: For the prototypic implementation of our framework, we focused on a stateful firewall scenario. Linux relies on the netfilter [6] and connection tracking [2] framework for this task. We implemented a driver for each of those components.

The *NWFilter Driver* is responsible for importing and exporting firewall rules. As Libvirt provides its own means to structure and organize VM related firewall rules, we do not interact with the netfilter framework directly. Instead we rely on an abstraction layer provided by Libvirt to import and export firewall rules that belong to a particular VM.

The *Conntrack Driver* imports and exports connection tracking entries. As Libvirt doesn't provide an interface for this task, we rely on the conntrack-tools [2] to export and import conneciton tracking information. To extract the correct subset of conneciton tracking entries, we match all connection tracking entries against the IP addresses that are associated with the migrating VM.

5.2 Functional Evaluation

To evaluate the functionality of our prototype and its impact on the VM migration process we have setup a small test scenario. We will present the testbed setup, the test cases and the results in the following.

Testbed Setup As figure 6 depicts, our test setup consists of three physical hosts. All hosts run the Linux operating system. Two hosts are virtualized via

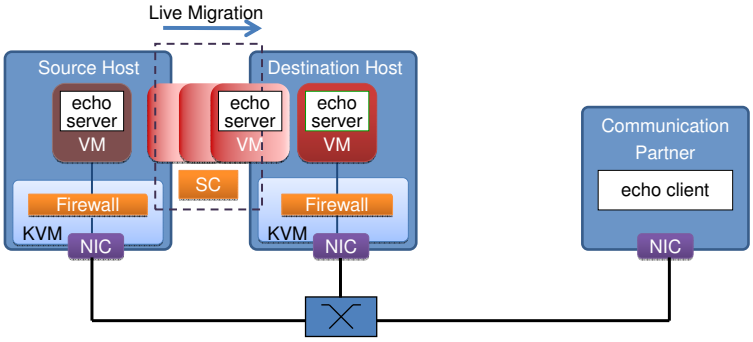

Fig. 6. Test Scenario

KVM, Libvirt and virsh. Initially, a VM runs on the source host. This VM executes a simple TCP echo server. The third host executes a TCP echo client application. Whenever the echo client sends data to the echo server, the echo server returns a copy of the received data to the echo client.

Test Cases and Results In the following, we will introduce three test cases and discuss their results. In the first test case we will perform a VM migration without any SC information. In the second test case we will migrate a VM with static SC information. In the third test case we will migrate a VM with static and dynamic SC information.

Test Case 1 In this test case we only migrate the VM and don't migrate any SC. The purpose of this test case is to evaluate whether our modifications to the VM migration process of Libvirt have any negative influence. For the test we disabled all firewalling functionality as well as connection tracking on the source and destination host. The test consists of migrating the TCP echo server and its VM from the source host to the destination host. We measure the time the migration takes and compare it to the migration time without our extensions. Furthermore, we check whether the echo client still can communicate with the echo server, after the VM of the echo server is migrated.

We measured that the migration takes about six seconds in total. The VM of the echo server is suspended for approximately one second. The echo client and echo server can continue to communicate after the migration is complete. Compared to a migration without our framework, the extension only adds a few milliseconds to the duration of the migration. Thereby the impact of our framework is insignificant for this test case.

Test Case 2 In this test case we verify whether our framework is able to migrate static SC. In particular, we want to verify whether our framework is able to migrate static firewall rules. For this, we setup a firewall default policy that

discards all VM related traffic. The policy is enabled on the hypervisor-level firewall on the source and destination host. Furthermore, we define a set of iptables rules on the source host that allows the echo client to communicate with the echo server. Only if these rules are migrated to the destination host, communication between echo client and server will be possible after the migration.

For the test we trigger the migration of the echo server VM. Again, the migration takes about six seconds. As we only migrate a few hundred bytes of static SC over a high speed local area network, transmission times and delay are insignificant. The VM is suspended for one second. As we migrate the static SC before the machine is suspended on the source host, the VM suspension time isn't affected at all.

Test Case 3 In the last test case we want to verify static and dynamic SC migration. For this test, we enabled connection tracking on the source host and destination host. Again the policy on both source host and destination host is to drop all VM related traffic. Furthermore, we defined a set of stateful firewall rules on the source host. The rules only allow echo client and server related TCP packets to traverse the firewall, if a TCP connection has been established. For this match criterion, the firewall interacts with the connection tracking framework of the Linux kernel.

For the test we trigger the migration of the echo server VM. As in the two tests before, the migration takes about six seconds. In contrast to the second test case, the suspension time of the VM should be slightly higher, as we migrate connection tracking information while the machine is suspended. However, as we still only migrate a few hundred bytes of SC information over a high speed local area network, the additional delay is insignificant. After the migration was complete, we verified that the firewall rules and connection tracking entries have been transferred successfully by exchanging messages between the echo client and server application.

All our tests were performed successfully. Thereby we conclude that our prototypic implementation is able to migrate dynamic as well as static SC information. The additional delay introduced by our prototype was insignificant for the considered scenario.

6 Conclusion

In this paper, we presented a flexible and extensible framework for Security Context migration. The framework is a key enabler for migrating Virtual Machines that are secured by a hypervisor-based stateful firewall. We presented the principle architecture of our framework as well as a prototypic implementation for KVM and the Linux operating system. We have shown some first measurement results with focus on verification of functionality. In future work, we plan a more comprehensive performance evaluation. In particular, we want to evaluate how the framework scales dependent on the amount of SC information to migrate. Furthermore, we intend to test our framework with other hypervisors that are supported by the Libvirt API.

Acknowledgements This work was funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (Förderkennzeichen 01BY1151, DynFire). The authors alone are responsible for the content of the paper.

References

1. Cisco nexus 1000v series switches (Mar 2012), http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns892/ns894/at_a_glance_c45-492852.pdf
2. Conntrack tools (Mar 2012), http://conntrack-tools.netfilter.org/index.html
3. Kvm (Mar 2012), http://www.linux-kvm.org
4. Kvm live migration (Mar 2012), http://www.linux-kvm.org/page/Migration
5. Libvirt (Mar 2012), http://www.libvirt.org
6. Netfilter (Mar 2012), http://www.Netfilter.org
7. Qemu (Mar 2012), http://www.qemu.org
8. Vmware (Mar 2012), http://www.VMware.com
9. Xen (Mar 2012), http://www.xen.org
10. Arcangeli, A., Eidus, I., Wright, C.: Increasing memory density by using kvm. In: Proceedings of the Linux Symposium. pp. 19 – 28 (2009)
11. Dawoud, W., Takouna, I., Meinel, C.: Infrastructure as a service security: Challenges and solutions. Security (2010), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5461732
12. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. RFC 4301, IETF (Dec 2005)
13. Miao, Q.G., Hui-Liu, Zhang, X.G., Liu, Z.L., Yang, Y.Z., Yun-Wang, Yin-Cao: Developing a virtual network environment for analyzing malicious network behavior. In: Educational and Network Technology (ICENT), 2010 International Conference on. pp. 271 –275 (june 2010)
14. Shah, A.: Kernel-based virtualization with kvm. Linux Magazine 86, 37–39 (2008), http://www.linux-magazine.com/w3/issue/86/Kernel_Based_Virtualization_With_KVM.pdf
15. Wu, H., Ding, Y., Winer, C., Yao, L.: Network security for virtual machine in cloud computing. In: Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on. pp. 18 –21 (30 2010-dec 2 2010)
16. Xianqin, C., Han, W., Sumei, W., Xiang, L.: Seamless virtual machine live migration on network security enhanced hypervisor. 2009 2nd IEEE International Conference on Broadband Network Multimedia Technology pp. 847–853 (2009), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5347800