Security technology of system virtualization platform

Dr. Qingni Shen
Peking University

Intel UPO Supported
Main Points

◆ Security analysis of system virtualization platform
◆ Security architecture of VMM
◆ Secure VM migration mechanism
◆ Technology of VM safety monitoring
◆ Analysis of covert channel between VMs
Security analysis of system virtualization platform
I/O security isolation can be achieved through reusing the same physical device by multiple virtual devices which are under VM virtual I/O interface’s control.

Through the sandbox security provided by VMM CPU virtualization, memory virtualization, and I/O virtualization to upper VM applications, in other words applications can only run in limited circumstances (VMs), so that the isolation is achieved between applications.

The isolation of VM machine memory space can guarantee the security of VM memory.

CPU virtualization security can be effectively controlled by CPU run level features.
Through inter-domain (between the VMs) communication mechanism, such as Xen event channel, grant tables etc, information flow between VMs is limited, which further enhances the isolation between the systems.

Event channel is an asynchronous event notification mechanism used by Xen between Dom and Xen, Dom and Dom, while both sides need to bind ports to allow sending event notification.

Grant Tables provide Xen a common memory sharing method between Doms. The security of shared memory page between VMs is guaranteed by giving a specific privilege to every memory page.
Virtualization technology brings safety advantages to computing platform, while at the same time, brings a series of security problems which are different from traditional computing mode.

**Problem of VM migration**

![Diagram showing VM migration](image1)

**Problem of VM identification**

![Diagram showing VM identification](image2)

**Problem of VM transient existence**

Create

Destroy

![Diagram showing VM transient existence](image3)

**Problem of VM diversity**

![Diagram showing VM diversity](image4)
Virtualization technology brings safety advantages to computing platform, while at the same time, brings a series of security problems which are different from traditional computing mode. It is difficult to track the source of the problem, and the virus infection to virtualization platform will be as simple as the infection to executable program, while reasonable resource mobility of user might be utilized by malicious attackers.

It is not feasible to create a unique identifier for MAC address in VM, while it leads to difficult establishment of responsibility mechanism.

The situation of sudden appearing or disappearing of large number of machines (VMs) in the network, which once used, will cause a more terrible spread of viruses.

VM can be instantiated in the network, and requests the same configuration and management as physical machine, while its state is able to be modified during execution or like the file system, which leads to arbitrary forgery and tempering.
Attackers will use covert channel to monitor CPU workload, cache response time, and status of multi-core processors to detect other VMs’ secret information on the same platform.

Attackers control the whole platform by attacking VM management domain utilizing the defects of device management model and management programs.

Attackers control guest OS’s kernel and application code completely to attack VMM software and other VM domains on the same platform through VMM interface.

Attackers could intercept, modify, discard and generate network traffic with no certain protection measures.
Security architecture of VMM

------XSM
What is XSM?

◆ General security framework for Xen
  ➢ Allow customization of modular security features
  ➢ Provide a common security interface
  ➢ Remove certain security code from the Xen kernel
The aim of framework interface

◆ Support mainstream security model
◆ Reduce the impact on Xen
◆ Can set whether to open the security framework
The rationality of XSM

◆ Different security goals according to the new model of Xen.
◆ Security goals should be able to dynamically configured.
◆ New security functions should not affect the mechanism of Xen.
New model

(1) For the decomposition of Dom0

- Remove all the dom0s which have excessive privileges.

For example:

- Endow each domain with minimal privilege.
  - Such as the privilege of separation of platform configuration and domain creation.

- One can create security module to define these privileges.
(2) For the division of resources

- How to divide and control resources?

For example:

- Allow specific domains to control the allocation of resources.
- Define the security module to control the allocation of resources.
(3) For the protection of the platform against third-party software attack.

- How to divide and control resources?
  - For example:
    - Isolation technology of device driver
    - Sandbox technology

- Define the security module to arbitrate the access across domains.
(4) For the protection of core security services of platform

- How to safely create a service of completely open platform?

  For example:
  
  - Media encryption
  - IP filtering or routing
  - Measurement and validation

- Define the security module to isolate and arbitrate the access to services to ensure the security of the service invocation.
Benefits of XSM security

◆ Encapsulation of security functions
  ➢ Only well designed security architecture can have good TCB.

◆ Extension to security functions
  ➢ Such as trusted IVC, while both sides can authenticate each without changing the existing communication mechanism.
Implementation of XSM

- Derive from the Linux Security Modules (LSM) (Linux 2.6.13.4)

- Security features
  - Security features from ACM
  - New customized security features

- Security module
  - Implementation of secure hooks
  - One can specify a certain security model.
Specific implementation of XSM

- **Initialization**
  - Before creating free Domain

- **Security structure of domain creation/destroy**
  - Domain_create
  - Domain_destroy

- **Event channel without exception**
  - evtchn_init
  - evtchn_bind_interdomain

- **Support no stack**
  - Behavior of stack-based security module, which may cause target failure, is difficult to predict.
  - Security module should implement a stack.
Module management of XSM

◆ Be registered and connected when start up.

◆ Modules can register a safe hypercall.

◆ Modules can register a policy magic number, which is used to identify and load a security policy when start up.
Implemented XSM modules

- Dummy (default)
- ACM/sHype (IBM)
- Flask (NSA)
XSM hooks

What do hooks do?

- Be inserted into appropriate position of the code.
- Allocation/Configuration of security structure.
- Initialization of platform security.
Methods of placing hooks

◆ Placed in key position

➢ Find the location by analysis.
  ● The availability of security related objects.
  ● Safety of location of hooks in code
  ● Usually good for security
  ● Comprehensive consideration of the position

➢ Placed in appropriate location of code
  ● Achieve a balance between location and maintenance.
- Reduce the impact on Xen code
  - Try to make use of existing error return path
- Keep reference to objects relying on function call
  - Xen will be safer if the security module does not refer to Xen’s objects directly
Existing hook positions

- dom0_ops.c
- domain.c
- grant_table.c
- event_channel.c
- setup.c
- mm.c
Performance analysis

More overhead with more checkpoints?

- A small overhead for each hook
- Assume a small overhead of “basic” call/return
- Hooks extra overhead is related with the definition of security modules.

No sign of significant performance overhead currently.

- Need further verification.
ACM Module

◆ Relationship between XSM and sHype/ACM

- sHype/ACM inserts in the XSM hook
- Apparent management, use and modification of sHype
- sHype/ACM can support a simple policy, such as the Chinese Wall or STE(simple TE)
Hooks Function

Types of hooks function

- PRE-Hooks
- POST-Hooks
- FAIL-Hooks
Operation procedures of hooks function

1. Start
2. Request of system operation
3. Pre-Hooks
4. Fail
   - No: Post-Hooks
   - Yes: Fail-Hooks
5. Fail-Hooks
   - Yes: Execute system operation
   - No: Error
6. Error
7. Rejection of system operation
8. End
ACM STE policy

- STE is a simplified TE policy, which controls the communication between VMs and VM’s access to resources.

  - Two attributes are defined in STE: type and label. Users define a series of types, and encapsulate these types into different labels, which marks the attributes of subjects and objects.

  | Label-d0={t1,t2,t3} | Label-d1={t2,t4} |

STE access rules are as follows:

1. If and only if VM X’s label-da has at least one same type as resource Y’s label-ry, VM X can access resource Y.
2. If and only if VM A’s label-da has at least one same type as VM B’s label-db, VM A and VM B can communicate with each other.

Label-s1={t1,t4}  Label-s2={t2}  Label-sn={t1}
**ACM Chinese Wall policy**

- Modified Chinese Wall policy determines whether two VMs run on the same platform simultaneously.

  - ACM CW model’s configuration files define the required security labels, called Chinese Wall Type. Each VM will be labeled one or more of the Chinese Wall Type security labels when the policy is enforced. Notably, the Chinese Wall Type label can only be used to label VMs other than mark system resources.

- ACM CW model makes the following provisions:

  Whether a new VM can start or not depends on the concentration of its conflict set in which whether there are other VMs. If the new VM and any VM of the same running state are in the same conflict set, it is not allowed to start.

  Assume that the system defines the following conflict set: \{Bank-A, Bank-B\}

  If VM 1 is active, the request of starting VM2 will be rejected.
Flask

◆ Outline

- ACM provides some security features, but it only implements 7 out of the 73 hooks defined in XSM. The implementation is neither perfect nor independent, and confuses the logic of strategy decision and strategy execution. Therefore, the Flask architecture, which is clearer and more independent, is implemented in Xen. Starting from Xen 3.0, the Flask architecture has been transplanted from SELinux into XSM, which meets various security policy requirements of Xen more flexibly.

◆ Model

- RBAC/TE
- MLS/MCS
How to use XSM in Flask

◆ Event channel
  ➢ For example: fine-grained control of the distribution of physical interrupt.

◆ Grant table
  ➢ For example: fine-grained control of the memory sharing between domains

◆ Dom0 operation
  ➢ For example: fine-grained resource allocation of I/O

◆ MMU
  ➢ For example: fine-grained control of external memory mapping
Secure VM migration mechanism
Benefits of VM migration

- Guarantee system platform’s load balancing
- Facilitate to update and maintain equipment and reduce the downtime
- Provide a choice for ensuring the isolation between VMs.

Dynamic migration technology means migrating a VM from one physical server to another physical server without interrupting services. Citrix, VMWare and Microsot’s products all support dynamic migration technology.

Methods of VM dynamic migration

Include:
1. Migration of file system (storage system)
2. Migration of memory
3. Redirection of input and output devices.

The first operation is via NFS. The second operation is through a phased approach to migrate the entire memory to the target node. The third operation makes use ARP protocol to redirect IP address to new physical machine within the cluster only.

Vmware Vmotion implements network connection migration by virtual network card of VM within the cluster only.
Secure VM migration mechanism mainly includes:
Secure migration mechanism based on trust third party MA (Migration Authority), and secure migration mechanism combined with trust computing

Instance of secure migration protocol based on trust third party
The process of migration should support the safe migration of vTPM instances between physical platform and protect data’s confidentiality and integrity in vTPM instances, and rebuild the trust chain in new platform.
Technology of VM safety monitoring
The necessity of VM security monitoring

Consider the situation that multiple VMs are running on the same physical machine, one VM’s security problem may affect more VMs, which expands the problem.

Main object of VM monitor

- VM’s performance
- VM’s memory
- VM network communication
- Events of virtual hardware
- Communication between VMs
- VM’s modification to files
- Behavior of the process inside VM
- ...

VM monitoring is usually connected with system’s audit module
VMI technology

Virtual Machine Introspection

◆ VMI refers to monitoring and analyzing VM’s state and information in the level of hypervisor.
  - When a VM’s access to file is transmitted to Dom0 or Xen kernel, it is no longer an identifiable file name, operation, or memory structure etc., it transforms into disk block, memory address completely, and needs VMI (Virtual Machine Introspection) technology to analyze.

◆ Current VMI technology of industry
  - Vmware’s VMSafe security architecture
  - XenAccess security API for Xen virtual architecture
VMWare VMSafe Security Architecture

VMSafe Architecture: Block malicious programs and network anomaly by using virtual layer

The third-party softwares, which support VMSafe component, are installed on VMWare ESX/ESXi platform, and are connected with Hypervisor’s VmSafe component, forming the VMSafe layer together.

Any traffic that through VM must pass the filter of VMSafe so as to be transferred to an external network or other VMs.

VMSafe is a virtual layer that integrates VMWare Hypervisor and third-party softwares. Through VMSafe technology, all the traffic that passes through VM will be checked and filtered, so that you do not need to install a security software on every VM, which reaches the same protection effect and saves the usage of hardware resource effectively.
XenAccess Security API

XenAccess, as a secure and flexible VM monitor architecture, provides a set of security API for third-party security monitoring softwares, and allows users to monitor the manipulation of VM memory and disk.

XenAccess runs in Domain 0 so as to communicate with the BlkTap architecture and XenCtrl library functions.
Analysis of covert channel between VMs
Outline

◆ Current situation

- Covert channel problem has been put forward for nearly 40 years, and the scope of research object includes program restriction, single OS, database system and network OS, as well as current cloud computing platform. There are some practical and instructive analysis approaches in every aspect of covert channel analysis from theory research and engineering.

- Multiple VMs can be isolated on the same virtualization platform, but there may be potential illegal information flows, the covert channel, between VMs even if mandatory access is enforced in VMM. This information flow will bring potential information leakage and DOS attack and so on.

◆ Multiple covert channel instances are found in Xen

- XenCC—Covert storage channel based on reading and writing the mapping table
- Covert timing channel based on CPU workload
- Covert channel based on shared memory and Cache
Types of covert channel on virtualization platform
1. **XenCC — Covert storage channel based on reading and writing the mfn2pfn mapping table**

**Existing reasons:**

The mapping table *mfn2pfn* is unique between machine address and pseudo physical address, so all guests can read and address owner can write.

**Reference information:**

- The mapping table is about the address rather than data, but user can write needed data because of the lack of check mechanism.
- Although pseudo physical address is readable, but we still need a mechanism to find writable area from all the address values. Therefore, the sender do write operation on a tag to urge the receiver to find the position, and then write the data that the sender want to transfer in the right position.

**Scenario construction:**

- The sender stores data of certain data structure (including tag, sync ack, data size, real data) into table *mfn2pfn*, and then waits for the other to read. It keep continuing to send after receiving ack until the transmission is completed.
- The receiver searches tag by traversing specified scope of table *mfn2pfn* and then read the corresponding data and set sync ack to notify the sender to continue to send until the receiving is complete.
Covert channel instances

2. Covert channel based on CPU workload

Existing reasons:

Scheduling mechanism based on quota distributes time slice to every VCPU regularly, while the actual CPU time is the time slice minus the consumption. The mechanism will dispatch again when all the VCPU time slices are consumed. Xen sets two parameters for each domain: weight and cap. While weight means the relative importance of domain scheduling, for example, a domain with a weight of 512 will get two times of CPU resource compared with the one with a weight of 256. Cap means the absolute upper limit of CPU resource that a domain can use, cap of 100 means maximum use of a CPU and cap of 50 is 0.5 CPU, while 0 means no limit.

Reference information: From the analysis we can see, the time of running a piece of code is 4T if only the receiving domain A is running, and the time of running a same code in domain A is 7T if the sending domain B is involved in scheduling. So domain A can transfer data by running several times of code in a fixed period of time.

Scenario construction:

The implementation refers to the times of a same code (that is the sender and the receiver process run the same program, namely the unitcomp, and its time is used as unit time that CPU consumes,) running circularly in the process in the same time period under the condition of individual operation and scheduling operation, stating that transmit 0 if the ratio is greater than 90%, otherwise transmit 0.
Covert channel instances

3. Covert timing channel based on shared memory

Existing reasons:

Covert timing channel based on shared memory means transferring bits when communication domains read or write shared memory.

Scenario construction:

P<sub>i</sub> and P<sub>j</sub> represent both sides of communication respectively, while T<sub>1</sub> and T<sub>0</sub> represent the length of time period, indicating the transmitted bit value 1 or 0.

1) P<sub>i</sub> encodes credential information that will be transmitted into binary string, using the symbol T<sub>0</sub> and T<sub>1</sub> respectively, and T<sub>0</sub>&lt;T<sub>1</sub>

2) P<sub>i</sub> establishes a ring sharing structure, and adds grant item in grant table, and then sends grant reference to P<sub>j</sub>

3) P<sub>j</sub> maps shared memory into its own address space.

4) P<sub>i</sub> represents the time of binary string, which is encoded of credential information, as T<sub>0</sub> and T<sub>1</sub>, and inserts them into sending time, and then sends raw data according to the modified send time interval sequence.

5) In each interrupt cycle, P<sub>j</sub> receives raw data and information of its arrival time, and calculates the time interval.

6) Continue to send raw data until the completion send when all credential information is sent.

7) P<sub>j</sub> gets all the raw data, and reverses time interval sequence into binary string, and then decodes into the credential information sent by P<sub>i</sub>. After the communication cycle, P<sub>j</sub> accesses shared memory mapping.

8) P<sub>i</sub> retrieves the grant reference, and the transmission cycle of covert channel is complete.
THE END, THANK YOU!