# Background study on Dom0 disaggregation and usage of Introspection technique in applications for virtualization security and monitoring

Mohammad-Reza Memarian

20 Oct 2014

## 1 Introduction

In this material, I have discussed some of the works done around VMI area and also some works that have discussed drawbacks of VMI and countermeasures to them. At last, some works on splitting Dom0 in Xen is discussed. Although We discussed a bit about Xen in the previous report, but it is worthy to give a deeper overview of it here.

### 1.1 Xen

Xen is a type 1 (baremetal) Virtual machine monitor (VMM) that traditionally used paravirtualized approach to accomplish virtualization. Paravirtualization means that the guest OS is aware of being run in virtual environment. This approach requires guest's OS modification. With the advancement of processors, full virtulization is possible through Xen too. Xen uses split domain architecture by keeping regular OSes in unprivileged domains (DomU). Virtual machines are referred to as Domains in Xen. A single administrative domain called domain 0 (Dom0) exists. It has complete access rights to all VMs and also works as a device driver proxy for DomU's Virtual devices. It is the first domain which boots after Xen. Apart from device drivers, Dom0 owns the primary control software. Xen is a hypervisour that provides fundumental isolation between virtual machines. Apart from roles that VMM plays like proper isolation between virtual machines and performing minimal resource management, it is responsible for partitioning system memory between VMs. Xen achieves this using three levels of memory: machine, physical and virtual address.

## 2 Related work

In this section, We go through related and mostly recent works that has been done is the area of Virtual Machine Introspection (VMI).

## 2.1 Secure and flexible monitoring of virtual machines

In [1], authors proposed a set of requirements that can be used to guide the development of virtual machine monitoring solutions and related techniques like introspection. Then They introduced XenAccess monitoring library. XenAccess has been designed based on Six requirements:

1. No superfluous modification to VMM: Since VMM is part of TCB, it should remain simple and small.

2. No modification to the VM or the target OS: Since modification of target OS may require access to the source code, it may be problematic.

3. Small performance impact: A huge overhead may turn monitoring worthless.

4. Rapid development of new monitors: Monitor code should be kept small and simple as they may be needed any time for new attacks. it means that APIs should be simple and standard for monitor developers.

5. Ability to monitor any data on target OS: Monitors should have a complete view into target OS. Monitoring architecture should not be limited to a small part of the OS. So the more info a monitor can view, the harder job is for attacker to remain hidden.

6. Target OS can not tamper with monitors: Monitors should be isolated from target OS for preventing to be tampered by them. So if All monitor code is placed in an isolated VM, it is an easy job but if they are not all placed within Trusted Computing Domain (TCB), then it is a bit hard to keep it safe. The paper focuses on the matter that disk monitoring is an important matter too that attracted attentions in the context of virtual machine.

In the section of related work of this paper, several other works and their violation to the six requirements mentioned earlier have been discussed . The authors claim that if the monitoring systems do not meet all the six requirements, monitors can be compromised or disabled easier by the attackers. The main contribution of the work is introduction of XenAccess, a VM monitoring library for OSes running on Xen. XenAccess in addition to enabling monitor application to have access to monitored VM's memory, It enables access to disk activities of the monitored system. Security applications running in virtual environment, benefit from segregation in virtual environments between VMs resulting in smaller TCB as Implementing introspection in a secure manner itself is an important job.

As mentioned earlier, XenAccess provides monitoring capabilities for both memory and disk I/O. However the architecture can be easily extended to mon-

itor additional information such as network traffic, CPU context and static disk content. The two primary monitoring functionalists in XenAccess are virtual memory introspection and virtual disk monitoring. XenAccess uses a function of libxc library to create access to VMs's memory so no changes is made neither to VMM nor OS. In Xen, the function which maps the memory from one VM into another already exists. So when the memory is mapped it can be treated as local memory. In order to convert a XenAccess call into a call to existing function, XenAccess requires several information about the target OS and several memory address translation should take place. XenAccess allows introspection into low-level disk traffic, and it is able to map raw memory pages. Using introspection, XenAceess can view and modify (read/write) data in memory of a running OS. Applications which use XenAccess, benefit from functions like listing running processes, viewing memory pages of a particular process on the target OS, monitoring changes in the system call table, intrusion detection and integrity checking and disk-based intrusion detection system to detect suspicious file/directory creation/deletion commonly done by root-kits. Several monitoring applications have been developed using XenAccess and introduced by the author, such as an application that monitors for the changes in the syscall table, an application that monitors the integrity of an installed Linux kernel module (LKM) and list the LKMs installed in a DomU machine. Another example is a disk monitoring application that outputs all file/directory creation/removals happening in DomU's/root directory as soon as they are committed to disk. Another example is a disk base intrusion detection system which detects suspicious file/directory creation/deletion commonly done by root-kits.

## 2.2 A framework for detecting malware in Cloud by identifying symptoms

Based on [2], The big problem facing cloud computing is to cope with identifying diverse set of malware. In this paper, a method of detecting malware by identifying symptoms of malicious behavior as opposed to looking for malware itself has been presented. for instance, an important registry key may be changed by not only a single malware but by a family of malwares. Of course the registry keys can be changed for legitimate reasons but this symptoms are just indications for further investigation. Detecting the change is enough to detect a bunch of malware instead of one.

So in this paper creation of forensic virtual machines (FVM) which are mini-virtual machines has been proposed to monitor other VMs to discover symptoms via VMI in real time. Each FVM is specialized for looking for one type of symptom. The FVMs exchange messages through a secure channel. FVMs report to command and control center so that center can correlate the information for suitable remedial actions. command and control can gather, compile and analyze information received by FVMs. The Command and control can freeze the memory or deny CPU cycle from VM. Malwares today are component based and use components of other malwares So better malwares with reduced rate of error in them is produced. Sometimes a malware uses and combines sev-

eral components. Because of the large number of combinations it is not simply possible to create VM inspectors that can identify all such possibilities. On the other hand, creation of large number of VMs to monitor a host VM is not practical and the matter of resource allocation and management gets really important. Any new solution must not introduce new attack vector. Relying on VMI, external view can observe some matters like changing state of VM's memory, processes that take inordinately long time to initialize, pieces of code that has been obfuscated, system codes that has been replaced and pieces of code containing known crypto algorithms. So based on the mentioned matters, they can be counted as symptoms which may not even be malicious. Some of the examples of symptoms which can be considered suspicious are in the following:

1. missing processes

2. modification of in-memory code

3. tampering with registry keys

4. checking for symptoms at start-up

5. modifying the time attribute of a file

6. identifying suspicious pieces of code

Because act of FVM monitoring is costly and the symptoms may show up occasionally, the designed FVMs change their target VMs periodically. So an algorithm called mobility algorithm is designed for this purpose. This mobility algorithm is a solution to the problem that mentioned earlier, efficient resource allocation. Each mobility algorithm is designed in a way so all the VMs will be visited and each FVM holds a copy of mobility algorithm. Not only integrity of an FVM is important but it is important to be convinced that FVMs do not do undesirable activities. That is why four step guideline for designing FVMs are proposed. This steps are:

1. FVMs only read: this keeps integrity of FVM and also help to remain hidden from the hackers as FVMs do not infer to VMs.

2. FVM should be small so each one only looks for one symptom. in this way their code can be inspected manually. FVMs inspect one VM at a time. For preventing information leakage, they monitor one VM then they flush their memory (VM's memory) before leaving to other VM.

3. Secure communication: FVMs communicate with each other and the management system through sending messages via secure multi-cast channel.

In the implementation of this paper, it is assumed that the VMs are running on the top of Xen and they use XenAccess library.

### 2.3 FVM: Dynamic defense in the Cloud via Introspection

The work done in [3], is basically written based on [2] with some improvements and more details. Here the major differences and improvements are discussed. In [3], FVMs are implemented using Mini-OS. Mini-Os is a lightweight Operating systems that is computationally cheap to run and only consist of necessary required functions. The previous paper mostly focused on the mobility algorithm but in this work they have focused on the development and deployment of FVMs. so the FVMs have been used to detect symptoms related to several families of malwares like ZeuS, Spyeye and Guass. In this work four requirements for design of FVMs are introduced which are Smaller FVMs to reduce attack surface, modular design to allow reuse, efficient introspection and lightweight design aiming to remain undetected. A Forensic Virtual Machine comprises four modules. These modules are in the following:

1. Main component, that is management of FVM. It provides the API for rest of the components. It checks whether TTL is expired or not to move to the next VM.

2. Black-board inter-domain communication, that communication between FVMs is done through a shared communication message-board. The main black-board in kept within Dom0 and FVMs can read and write to it through a library called XenStoreSocket.

3. Search component, that consists of a function and an API to search within memory pages. Searching a string in the memory page of a given process can be a possible functionality.

4. Mobility Algorithm: three mobility algorithms have been implemented by the authors. Two algorithm in [3] and one in [2]. These algorithms can be passed as parameters to the Mini-Os by the users. In their previous work [2], they have used XenAccess as introspection library, but in this work [3] they have used LibVMI introspection library. The difference between XenAccess and LibVMI is that XenAccess is primarily designed for Xen but LibVMI which is a sub-product of XenAccess is suitable for both KVM and Xen.

## 3 Security problems of VMI technique

Apart from the benefits that Introspection has for system monitoring in the context of virtualization, there are some security problems and drawbacks too. Various works are done on the drawbacks of VMI. Some of them are discussed

in the following sections.

### 3.1 Defense-in-depth Against Malicious Insiders in the Cloud

Based on [4], One of the critical shared resources in multi-tenant environment is rea-Only-Memory (RAM). RAM is vulnerable to attacks in Xen because of the high privileged permission that is granted to Dom0. In [4], an attack has been done against new inter-virtual machine communication mechanism, libvchan which is under development for xen hypervisour. Just to give a short description, Inter-VM communication can be used to obtain high-throughput communication between VMs running in the same physical host. libcvhan implements high throughput between running processes in different VMs running on the same host. As a mandatory access control (MAC) mechanism is vital for Xen, an architecture has been proposed for tackling this problem. The main focus of this paper is to defend against malicious insider attack and it discusses how they (malicious insiders) can compromise data on memory area allocated to a customer VM. The current memory access model used for Xen's Dom0 does not comply with the security design principle of least privilege So granting full access instead of partial access to memory for Dom0 is too permissive. This work [4], demonstrates a sophisticated attack against VM in Xen-powered cloud infrastructure, Proposes and implements a lightweight MAC for Xen and reduces the TCB for a Xen-powered cloud infrastructure. At the first sight cryptography is the first solution that comes to mind for protecting confidentiality of the stored and in transit data. On the other hand processing the user data which is encrypted is not a complete solution yet as computational power is needed. Various works have been done in this area such as reducing TCB size of VMM, inserting a security dedicated VMM below the current VMM. So as mentioned earlier, in [4], a sophisticated attack is done to demonstrate how a malicious insider can use the functionality offered by VMI to compromise data from a running process in a DomU's memory space. Also they have came up with a design to completely pull Dom0 out of VMM TCB.

### 3.2 CryptVMI: A Flexible and Encrypted Virtual Machine Introspection System in the Cloud

According to [5], VMI tools normally are written in a way that they do not work with other security systems as they do not provide a standard interface. On the other hand VMI technique somehow breaks the border between multiple tenants specially in public cloud environments. So this paper focused on building a flexible and encrypted VMI system which is called cryptVMI to address mentioned concerns. In this solution a client-side application is designed working on the user-end which sends queries to the cloud and parses the results and returns them in a standard way. Also a handler is designed in the cloud side

which works with the introspection application and handles queries.

Companies running their services on public cloud services would not want to expose their application states to the cloud service provider, even this technique can create opportunity for the administrators to get advantage of the situation which should be minimized. So VMI can be seen as a useful technique for customers to get state of their application but possibility of exposing information from VM makes it difficult to apply this technique in the public cloud. so CryptVMI presents system status information in a confidential manner to users. As a result a desired way is to have a whole process of VMI encrypted. Indeed the aim of the research is to prevent potential attackers including cloud administrators from knowing the entire state of a user's VM. In order for VMI system to be easily extensible to other systems, CryptVMI gets benefits of RESTful API. REST is a data style designed for modern web and distributed application while ignoring the detail of component implementation and protocol syntax in order to provide better performance and scalability of component interaction. For the implementation, authors of the work have benefited JavaScript Object Notation(JSON) for queries and results. As the query itself may reveal information, both query and result are encrypted using symmetric key.

So the design has three components. First one is the application at user side which accepts queries from the user and directs them to the handler. The received data is decrypted and decoded if needed by the same application. Clients can modify the code to get the result in different formats to make it compatible with other applications. Second component in located in Cloud Service provider (CSP) side and handles queries sent from the application. Third component is strategies which to acquire desired results with encryption.

CryptVMI is implemented in an environment which consists of Openstack as cloud platform, Xen as VMM and LibVMI as introspection library. Three components mentioned earlier are written in Ruby.

### 3.3 CloudVMI: Virtual Machine Introspection as a Cloud Service

According to [6], despite benefits of VMI, it can not be presented to users in public clouds. The reason is that VMI requires privileged access to the VMM which should be strongly protected. As a solution to this problem CloudVMI virtualizes the VMI interface and makes introspection, as-a-service in a cloud environment. The implementation of this work is based on LibVMI library. It provides the functionality of mapping raw memory pages of VMs inside the privileged VM and relies on monitoring software to interpret the contents of these mapped pages. Due to semantic gap between the information presented in the memory pages (raw-bits) and the higher-level information used by the application and the operating system, Monitoring software builds the higher-level abstraction by using knowledge of the internals of application or operating

system running inside the VM. For this reason VMI based monitoring software gets closely tied with the specific Operating system and application type. From the view of a cloud service provider, offering VMI tools capability to customers is a dangerous work with administrative overhead. Also this matter should be considered that there are variety of operating system versions and distributions running in the cloud which supporting all of them are neither feasible nor salable for CSPs. So cloudVMI addresses this concern and allows VMI to be offered as a cloud service by CSP to users. The focus of this paper is to look into the first part, how CSPs will allow users to perform privileged memory mapping functionality. The design of CloudVMI is to virtualize the libVMI interface and allowing cloud customers to invoke it in a safe manner. Second, requiring customers to use their own policies and implemented monitoring software that relieves cloud provider from supporting many application and OS versions.

Assuming a user with multiple VMs running in a cloud, she should apply for having a single monitoring VM in the cloud environment. From that single monitoring VM he can monitor all of his VMs using VMI capabilities provided by CSP. This is a centralized approach. So there are two sets of VMs, monitoring VMs and monitored VMs. In implementation section of the work, authors have used Xen as VMM and LibVMI as introspection library. There are two modules, *vLibVMI* client library and the vLibVMI service module. The vLibVMI service module is a server program that receives VMI requests from monitoring virtual machines, perform VMI using the underlying libVMI interface and return the result. Implementation of vLibVMI is done in C++. CloudVMI implementation exposes the vLibVMI interface as a RESTful API which enables users to perform VMI using HTTP request regardless of the languages and platform they use. One of the benefits of this solution is that it allows Monitoring machines to monitor VMs that are not even located on the same physical machine and they are spreaded in the cloud using same monitoring VM.

# 4 Dom0 Disaggregation

Dom0 consists of important functions which makes each VM's Trusted computing Base large. Several works have been done toward splitting Dom0 into smaller parts to make VM TCB smaller. This is called Dom0 disaggregation. Currently There is an ongoing project to split Dom0 into multiple privileged domains known as driver domains and stub domains. The current architecture just has three levels of separation: dom0, the OpenStack domU, and the completely unprivileged customer VMs.

Based on [7], Dom0 was a monolithic privileged VM. However the term Dom0 disaggregation was intended to break Dom0 into several privilaged service domains. The dissagregation can lead to better security, reliability, isolation and audit-ability. The prevalent use of Dom0 disaggregation is in the area of security compared to other areas. For this reason Windsor(is explained later) has

been defined as Dom0 Disaggregation for XenServerXcp. Some projects like Qubes OS, Citrix XenClient and Xoar have helped to ease Dom0 disaggregation. Windsor is 3rd generation of XenServer architecture which uses domain0 disaggregation for scalibility and performance.

## 4.1 Improving Xen Security through Disaggregation

In [8], a nice work has been done on disaggregation. This paper makes VMs TCB smaller. In the previous conventional Xen design, DomU VMs had to trust the whole Dom0 structure. But with this design, DomU VMs should only trust Dom0 kernel-space as User-space of Dom0 has been disaggregated and pulled out of TCB. Management of Xen-based system requires a privileged operating system to be included in the Trusted computing Base (TCB). In this paper they have introduced their work to disaggregate the management Virtual machine in a Xen-Based system. So they have chosen one of the Dom0 Component as case study which is the component for VM creation called domain builder. They have moved domain builder, the most important privileged component into a minimal trusted component. In short Trusted computing base or TCB are set of component in which S trusts not to violate the security of S. In order to evaluate the trustworthiness of a software system, it is necessary to identify its TCB. So before disaggregation the TCB of Xen-based system apart from VMM consists of fully privileged operating system (Dom0) and a set of user-space tools. These tools require privileges like creation of a new VM. Due to inclusion of Dom0's user-space in TCB, the TCB itself becomes Unbounded. So system administrators can run any arbitrary code which makes this fact that system admin should be trusted. So they have transferred VM-building functionality into a small trusted VM that runs alongside Dom0. As delegation of domain builder task to Dom0 required privileged accesses, special privileges can be exposed to other user-space processes in dom0. So in order to make an appropriate partition of the code base, it is necessary to determine which section of code must be trusted and which may be not trusted.

So for this matter operations should be categorized. According to [8], there are three categories of operations. privileged operations, sanitizing operations and invokable operations. Privileged operations must be implemented in the different address space from the untrusted code. It is always said that lines of code is a base for trustworthiness of the piece of of code. But in this work, two new criteria has been determined for trustworthiness of code. First, The size of the interface and second, size of the TCB state space. For example sanitizing process in very important as a lot of vulnerabilities can be exploited using them. So reducing size of sanitizing code is more important.

So as mentioned earlier, the focus of this paper is on domain builder as it is the fundamental operation that has led to a large TCB for xen-based systems. So to make the TCB smaller it must be made sure that privileged operations are not invokable from untrusted code. so in VMM base system, it is possible by moving

privileged operations to another VM and make a sensitization between all calls from untrusted code. So in this work, authors put a sanitizing operation between Dom0 user-space and the privileged operations in the domain builder. As the implementation detail, domain builder is ported to a minimal paravirtualized operating system which is Mini-OS.

### 4.2 Breaking up is Hard to do: Security and Functionality in a commodity Hypervisour

According to [9], VMM platform have a large aggregate TCB that includes a monolithic control VM with numerous interfaces exposed to unprivileged VMs. So the paper has introduced Xoar, a modified version of Xen that retrofits the modularity used in Mikro-kernel onto a mature virtulization platform. Xoar breaks the control VM into single purpose components called service VMs. For example in the case of Xen, control VM houses a lot of functionalities such as device emulation and multiplexing, system boot, administrative tool stack and etc. The primary contribution of [9] is to perform a component-based disaggregation of a mature, broadly deployed virtualization platform in a manner that is practical to maintain.

Authors decomposed Xen's control VM into a set of 9 classes of service VMs. As mentioned earlier, the resulting system is called Xoar. VMM plus the control VM, are part of the TCB. A compromise of any component in TCB gives the attacker two benefits. First they gain privilege of that component such as access to arbitrary sections of memory, second they can access its interfaces to other elements of the TCB which allows them to inject malicious request or response maliciously over those interfaces. In the paper's threat model, the attacker is a guest VM that is aiming to attack another guest VM that they are sharing the same underlying platform. It is very important that in the case of control VM, a successful attack on any one of its interfaces can lead to innumerable exploits against guest VMs. Also this paper works on isolation of components so compromise of one component does not affect other components.

## References


[1] W. L. Bryan D. Payne, Martim D.P. de A. Carbone, "Secure and fleible monitoring of virtual machines," in *23rd Annual Computer Security Applications Conference.*

[2] S. T. A. C. I. A. N. Keith Harrison, Behzad Bordbar, "A framework for detecting malware in cloud by identifying symptoms," in *2012 IEEE 16th International enterprise distributed object computing conference.*

[3] J. S. K. H. C. I. D. Adrian L. Shaw, Behzad Bordbar, "Forensic virtual machine: Dynamic defence in the cloud via introspection," in *2014 IEEE International Conference on Cloud Engineering.*

[4] A. V. M. Francisco Rocha, Thoma Gross, "Defense-in-depth against malicious insiders in the cloud," in *2013 IEEE International Conference on Cloud Engineering*.

[5] R. H. C. Fangzhou Yao, Read Sprabey, "Cryptvmi: A flexible and encrypted virtual machine introspection system in the cloud," in *ACM Security in Cloud Computing(SCC) 2014*.

[6] J. V. d. M. Hyun-wook Baek, Abhinav Srivastaa, "Cloudvmi: Virtual machine introspection as a cloud service," in *2014 IEEE International Conference Cloud Engineering*.

[7] . () Dom0 disaggregation. [Online]. Available: http://wiki.xen.org/wiki/Dom0$_D isaggregation$

[8] S. h. Derek G. Murray, Grzegorz Milos, "Improving xen security through disaggregation," in *Virtual Execution Environments (VEE) 2008*.

[9] J. Z. W. A. G. C. T. D. P. L. A. W. Patrick Colp, Mihir Nanavati, "Breaking up is hard to do: Security and functionality in a commodity hypervisour," in *2011: 23rd ACM Symposium on Operating Systems Principles (SOSP)*.