- Crosby, Simon, and David Brown. "The Virtualization Reality: Are hypervisors the new foundation for system software?."

In this article we provide an overview of system virtualization, taking a closer look at the Xen hypervisor and its paravirtualization architecture. We then review several challenges in deploying and exploiting computer systems and software applications, and we look at IT infrastructure management today and show how virtualization can help address some of the challenges.

Virtualization enables a single computer to host multiple different operating system stacks, and it decreases server count and reduces overall system complexity. All modern computers are sufficiently powerful to use virtualization to present the illusion of many smaller VMs (virtual machines), each running a separate operating system instance. An operating system virtualization environment provides each virtualized operating system (or guest) the illusion that it has exclusive access to the underlying hardware platform on which it runs. Of course, the virtual machine itself can offer the guest a different view of the hardware from what is really available, including CPU, memory, I/O, and restricted views of devices.

Operating system virtualization is achieved by inserting a layer of system software—often called the hypervisor or VMM (virtual machine monitor)—between the guest operating system and the underlying hardware. This layer is responsible for allowing multiple operating system images (and all their running applications) to share the resources of a single hardware server. Each operating system believes that it has the resources of the entire machine under its control, but beneath its feet the virtualization layer, or hypervisor, transparently ensures that resources are properly and securely partitioned between different operating system images and their applications. The hypervisor manages all hardware structures, such as the MMU (memory management unit), I/O devices, and DMA (direct memory access) controllers, and presents a virtualized abstraction of those resources to each guest operating system.

History and early defects (paravirtualization vs full virtualization)

The most direct method of achieving virtualization is to provide a complete emulation of the underlying hardware platform's architecture in software, particularly involving the processor's instruction set architecture. For the x86 processor, the privileged instructions—used exclusively by the operating system (for interrupt handling, reading and writing to devices, and virtual memory)—form the dominant class of instructions requiring emulation. By definition, a user program cannot execute these instructions. One technique to force emulation of these instructions is to execute all of the code within a virtual machine, including the operating system being virtualized, as user code. The resident VMM then handles the exception produced by the attempt to execute a privileged instruction and performs the desired action on behalf of the operating system. In early systems with "virtualization holes", certain instructions execute in both user mode and supervisor mode but produce different results, depending on the execution mode. A common approach to overcome these problems is to scan the operating system code and modify the offending instruction sequences, either to produce the intended behavior or to force a trap into the VMM. Unfortunately, this patching and trapping approach can cause significant performance penalties.

An alternative way of achieving virtualization is to pre-sent a VM abstraction that is similar but not identical to the underlying hardware. This approach has been called paravirtualization.

Paravirtualization and the Xen Hypervisor

Xen was initially developed by Ian Pratt and a team at the University of Cambridge in 2001-02, and has subsequently evolved into an open source project with broad involvement. Any hypervisor (whether it implements full hardware emulation or paravirtualization) must provide virtualization for the following system facilities:

1. **CPUs (including multiple cores per device)**
2. **Memory system (memory management and physical memory)**

3. **I/O devices**
4. **Asynchronous events, such as interrupts**

Let's now briefly examine Xen's approach to each of these facilities.

In **Xen's paravirtualization, virtualization of CPU a**nd memory and low-level hardware interrupts are provided by a low-level efficient hypervisor layer that is implemented in about 50,000 lines of code. When the operating system updates hardware data structures, such as the page table, or initiates a DMA operation, it collaborates with the hypervisor by making calls into an API that is offered by the hypervisor. Xen makes a guest operating system (running on top of the VMM) virtualization-aware and presents it with a slightly modified x86 architecture, provided through the so-called hypercall API. The operating system must be modified to deal with this change, but in a well-structured operating system, these changes are limited to its architecture-dependent modules, most typically a fairly small subset of the complete operating system implementation. Most importantly, the bulk of the operating system and the entirety of application programs remain unmodified.

For Linux, the Xen hypercall API takes the form of a jump table populated at kernel load time. When the kernel is running in a native implementation (i.e., not atop a paravirtualizing hypervisor), the jump table is populated with default native operations; when the kernel is running on Xen, the jump table is populated with the Xen hypercalls. This enables the same kernel to run in both native and virtualized forms, with the performance benefits of paravirtualization but without the need to recertify applications against the kernel.

Isolation between virtual machines (hence, the respective guest operating systems running within each) is a particularly **important property that Xen provides**. The physical resources of the hardware platform (such as CPU, memory, etc.) are rigidly divided between VMs to ensure that they each receive a guaranteed portion of the platform's overall capacity for processing, memory, I/O, and so on.

I/O virtualization in a paravirtualizing VMM such as Xen is achieved via a single set of drivers. The Xen hypervisor exposes a set of clean and simple device abstractions, and a set of drivers for all hardware on the physical platform is implemented in a special domain (VM) outside the core hypervisor. These drivers are offered via the hypervisor's abstracted I/O interface for use within other VMs, and thus are used by all guest operating systems.

In **full-virtualization (emulation)** implementations, the platform's physical hardware devices are emulated, and the unmodified binary for each guest operating system is run, including the native drivers it contains. In those circumstances it is difficult to restrict the respective operating system's use of the platform's physical hardware, and one virtual machine's runtime behaviors can significantly impact the performance of the others. Since all physical access to hardware is managed centrally in Xen's approach to I/O virtualization, resource access by each guest can be marshaled. This provides the consequential benefit of performance isolation for each of the guest operating systems.

Virtualization allows many large application loads to be placed on one hardware platform, or a smaller number of platforms. This can cut both per-server capital cost and the overall lifetime operational costs significantly.**Server consolidation**.+ dynamic provisioning, high availability, fault tolerance, and a "utility computing" paradigm in which compute resources are dynamically assigned to virtualized application workloads + The administrative tasks of installing and configuring an operating system and the necessary applications prior to instantiating and launching a software service on a platform are no longer needed.

packaged VM offers additional benefits - Easily and instantly provisioned onto testing equipment,quickly tested in a cost-efficient environment before being made available as a packaged VM, ready for deployment into production

Live relocation—the ability to move a running VM dynamically from one server to another, without stopping it. When coupled with load-balancing and server resource optimization software, this provides a powerful tool for enabling a “utility computing” paradigm.

On the client, virtualization offers various opportunities for **enhanced security, manageability**, greater worker mobility, and increased robustness of client devices.

Increased security of data, applications and their context of use, and reduced overall cost of administration for client systems are important aspects of this technology. Enhanced reliability and security can be achieved, for example, by embedding function-specific, hidden VMs on a user’s PC, where the VM has been designed to monitor traffic, implement “embedded IT” policies, or the like. VMware’s free Player application is a thin, client-side virtualization “player” that has the ability to execute a packaged VM. Examples include prepackaged secure Web browsers that can be discarded after per-session use (to obtain greater security) and secured, user-specific or enterprise-specific applications.

- Morabito, Roberto, Jimmy Kjällman, and Miika Komu. : Hypervisors vs. lightweight virtualization: a performance comparison

In this article we provide an overview of performance comparison of traditional hypervisor based virtualization and new lightweight solutions (normally in cloud - Container-based Virtualization). performance analysis using different benchmark tools with hypervisor-based solutions, container and alternative solutions. The idea is to quantify the level of overhead introduced by these platforms and the existing gap compared to a non-virtualized environment. we focus on containers in Linux, which are implemented primarily via control groups (cgroups) and namespaces in recent Linux kernels.

benchmarking applications using KVM(hypervisor), LXC(container based), Docker(container based), and OSv(hybrid hypervisor based). We use native (non-virtualized) performance as a base case in order to measure the overhead of the virtualization. The benchmark tools measure (using generic workloads) CPU, Memory, Disk I/O, and Network I/O performance.

Hypervisors abstract hardware, which results in overhead in terms of virtualizing hardware and virtual device drivers. A full operating system (e.g., Linux) is typically run on top of this virtualized hardware in each virtual machine instance (Large size of image). In contrast, containers implement isolation of processes at the operating system level, thus avoiding such overhead. These containers run on top of the same shared operating system kernel of the underlying host machine, and one or more processes can be run within each container.

Due to the shared kernel (as well as operating system libraries), an **advantage** of container-based solutions is that they can achieve a higher density of virtualized instances, and disk images are smaller compared to hypervisor-based solutions. The shared kernel approach has also a few **disadvantages**. One limitation of containers is that, e.g., Windows containers cannot be run on top of a Linux host. As another tradeoff, containers do not isolate resources as well as hypervisors [35] because the host kernel is exposed to the containers, which can be an issue for multi-tenant security

- Type-1: native or bare-metal hypervisors (operate on top of the host’s hardware)
- Type-2: hosted hypervisors (operate on top of the host’s operating system).

ZeroVM [hybrid virtualization solutions] that provides a single-application environment running inside NaCL sandbox from Google. Since ZeroVM operates entirely as a lightweight and restricted userspace

process, it is easy to launch ZeroVM applications to process.A limitation of ZeroVM is that it is very challenging to port existing Linux software to it because it implements a subset of Linux functionality.

Cloud OS solution called OSv - OSv is intended to be run on top of a hypervisor (KVM, Xen, VirtualBox, and VMware). In other words, it achieves the isolation benefits of hypervisor-based systems, but avoids the overhead (and configuration) of a complete guest OS. OSv can only run executables in Linux relocatable shared object format because OSv mostly implements the ABI of Linux [34]. As OSv supports only a single process, a limited subset of POSIX, Linux syscalls, this means that Linux libc can be invoked. Other system calls, such as fork(), exec(), clone(), are not supported at the moment. Thus, porting software to OSv usually requires some effort.

CPU Performance (Y-cruncher): Both container-based solutions perform better than KVM. Considering only the computation time, containers display performance almost similar to the native environment.

CPU, FPU (Floating Point Unit) and memory system performance measurements (NBENCH): noticeable difference only in terms of memory index between the analysed platforms: KVM introduces roughly 30 percent performance degradation.

CPU clock speed ("noploop") verifying that all systems perform on the same level without major differences in this simple case. Counterintuitively, noploop performs better in OSv than in the native environment, despite the difference is minimal.

MegaFLOPS (Linpack tests the performance of a system using a simple linear algebra problem):relative differences between the different platforms are not substantial.Linpack results with varying N, where OSv presents some performance degradation in the rising zone, but the differences are neglectable with larger values of N.

Disk I/O Performance: sequential write (Block Output) and sequential read (Block Input) speed (Bonnie++): The two container-based platforms offer very similar performance in both cases, which are quite close to the native one. KVM write throughput is roughly a third and read throughput almost a fifth of the native one

Disk I/O Performance: Random Write test (Bonnie++): LXC is performing now much better than Docker (approximately 30% better), KVM still worse and native still best

With Native, KVM, and LXC we obtained roughly always the same result without any significant deviation. Differing behavior was observed with Docker, which was performing for a few runs even better than Native (135 MB/s).

Memory Performanced(STREAM): The performance of OSv is approximately half of the others according to these benchmarks.

Network I/O Performance(Netperf):

Using TCP, LXC and Docker achieve almost equal performance compared to the native one, and KVM is 28.41% slower. OSv performs better than KVM and it introduces a gap equal to 26.46% compared to the Native system

Using UDP, All platforms offer lower throughput with UDP. LXC and Docker offer comparable performance between them (but respectively 42.14% and 42.97% lower than native); KVM overhead is the largest (54.35%). OSv ranks in the middle (46.88% worst than native).

TCP_RR test, LXC and Docker introduce a moderate level of overhead (17.35% and 19.36%) when compared to native execution. KVM offers the lowest result (47.35% slower than native). OSv performs better than KVM (roughly 4% faster). UDP_RR test, the relative differences between the different

platforms are similar to the TCP_RR test, with. LXC and Docker that slightly reduce the performance gap (difference is now 10.82% for LXC, and 12.13% for Docker). OSv introduces roughly the same gap as for the TCP test (43.14%), while KVM is 45.76% slower than non-virtualized environment.

Conclusion:

KVM: especially Disk I/O efficiency can still represent a bottleneck for some types of applications, even though a further evaluation for Disk I/O is still needed due to some inconsistency between the different tools

Container Based: The level of overhead introduced by containers can be considered almost negligible. Taking all of the differences between LXC and Docker into account, we confirm that containers perform well, albeit the versatility and ease of management is paid in terms of security

OSv:OSv represents an interesting work-in-progress alternative, although it introduces some limitations in terms of software portability. Indeed, porting existing software requires some effort. However, some of the performance measurements look promising, and also the size of VM images is relatively small (in the range of hundreds of MBs).

- Jiang, Congfeng, et al. "Energy efficiency comparison of hypervisors."

Owing to hardware abstraction and the semantic gap between the virtual machine (or container) and the underlying hardware, the virtual machine operating system cannot invoke actual power-aware management as in a non-virtualized environment. Moreover, the use of hypervisors may affect different hardware platforms and guest virtual machines at different levels in terms of energy efficiency. This can provide crucial information for design of data centers.

In this article we compare the energy efficiencies of hypervisors on the same server. In particular, we attempt to answer the following questions:

(Q1) Do different hypervisors have approximately the same energy efficiency for a virtual machine running the same application on the same hardware platform? If not, how much is the difference? Answering this question is essentially equivalent to investigating how we can choose hypervisors to host virtual machines.

(Q2) Does one hypervisor have different energy efficiencies on different hardware platforms running the same virtual machine and same application? Answering this question is equivalent to investigating whether more than one type of hypervisor needs to be deployed to harvest the energy efficiency variability of existing servers or whether a single hypervisor can be deployed on all the servers to simplify the management cost for a given workload.

(Q3) Does one hypervisor have different energy efficiencies on the same hardware platform running different applications within its virtual machines? Answering this question is equivalent to investigating whether hypervisor affinity exists across different applications or workloads.

(Q4) Should we always use newer hardware to achieve better energy efficiency in a virtualized environment? Do these platforms significantly differ in terms of energy efficiency in a virtualized environment?

We conducted experiments in three orthogonal dimensions: hardware, hypervisor, and workload. Workload intensity into four workload levels, namely very light, light, fair, and very heavy, in order to emulate a realistic multi-tenant virtualized cloud environment. Four mainstream hypervisors and a

container engine, namely VMware ESXi, Microsoft Hyper-V, KVM, XenServer, and Docker, on six different platforms (three mainstream 2U rack servers, one emerging ARM64 server, one desktop server, and one laptop).

Computation-intensive workload(PrimeSearch), Memory-intensive workload(STREAM),Mixed workload(LAMP).

Results:

Computation-intensive workloads: Rack servers, the maximum power variation occurs when the workload is the heaviest, whereas on the desktop server, laptop, and ARM64 server, the maximum power variation occurs when the workload is the lightest.

Although the hypervisors have different energy efficiencies aligned with different workload types and workload levels, no single hypervisor outperforms the other hypervisors in terms of power, energy consumption, or completion time for all workload levels on all platforms. One hypervisor may have the highest power consumption for some or all workload levels on one platform, whereas it may have the lowest power consumption for some or all workload levels on another platform. In other words, there is no hypervisor affinity across different hardware and workload levels

Container virtualization is not more power-efficient than conventional virtualization technology for computation-intensive workloads.

Mixed workloads: Except for the much higher power and energy variations among all the hypervisors on the same platform running mixed workloads, the power variations during the execution of the mixed workloads are greater than those of the computation-intensive workloads.

Memory-intensive workloads:Docker has the highest power consumption for all workload levels except the 1/4 workload level. Because Docker does not use memory virtualization, it is the first to complete all memory-intensive operations followed by KVM. Moreover, Docker has the highest power deviation because it has the shortest completion time for all workload levels. Although Docker has the highest power consumption, it outperforms the other hypervisors in terms of the best rate and completion time.

Insights on energy efficiency of hypervisors:

energy per database insertion for all the hypervisors. On all the platforms and at all workload levels, ESXi consumes the least energy among all the hypervisors.

Insight #1: The hypervisor, hardware, and workload type are coupled with each other, and such complication requires system designers to be mindful of virtualized infrastructure and cloud data centers to carefully select hypervisors.

Insight #2: The power and energy efficiencies change with the workload level.

Insight #3: ESXi should be deployed in a non-power-sensitive environment to achieve high computing performance, while KVM should be deployed in a power-sensitive environment in order to deploy as many virtual machines as possible and achieve reasonable computing performance.

Insight #4: Typical 2U servers have a higher idle power percentage and should always be running with heavy virtual machine workload because their idle power percentage decreases when the workload increases.

Insight #5: The newly manufactured highly energy-efficient servers consume less (but not always the least) energy for the same VM workload, compared with older servers. Thus, we need to carefully select the hypervisors for specific workloads to achieve further energy reduction even on newer hardware platforms.

Insight #6: Although the ARM64 server has lower max power than a laptop, it has a higher idle power percentage and a less dynamic range. In other words, the ARM64 server should be deployed in a steady power usage server room, while a laptop (customized mobile server) should be deployed for highly dynamic workloads to leverage and complement the power fluctuation due to workload variations.

Insight #7: ESXi uses power more actively to achieve high performance and high throughput, especially in highly contending conditions (for very stressful workload). ESXi and XenServer consume power more actively than Hyper-V and KVM on customized mobile servers.

Insight #8: ESXi-based virtual machines are migration candidates for power shifting or capping conditions because ESXi uses power more actively during the early stage of mixed workload experiment execution on a typical 2U server.

Conclusion: We used power and energy measurements to investigate the power and energy characteristics of different mainstream hypervisors on different types of servers. Our results showed that hypervisors exhibit different power and energy characteristics on the same hardware with the same workload. Moreover, different hypervisors exhibit different attributes and align with different workload types and workload levels. In addition, they may be deployed for different workload levels in different power situations. Our results also showed that container virtualization is a lightweight technique in terms of system implementation and maintenance, but essentially not more power-efficient than conventional virtualization technology. Finally, although ARM64 servers have low power consumption, they require long execution times to complete computing jobs and sometimes consume a large amount of energy as well. Thus, laptop processors and motherboards are strong competitors of the ARM64 server in terms of both power and energy consumption.

- Riddle, Andrew R., and Soon M. Chung. "A survey on the security of hypervisors in cloud computing."

Attacks that allow a malicious virtual machine (VM) to compromise the hypervisor, as well as techniques used by malicious VMs to steal more than their allocated share of physical resources, and ways to bypass the isolation between the VMs by using side-channels to steal data. Also discussed are the security requirements and architectures for hypervisors to successfully defend against such attacks.

Establish Co Residency through Side Channel: A side-channel is a method of communication using a medium not originally designed for data transfer. A side-channel typically makes use of shared resources, such as the CPU cache, memory, network, power consumption, etc. to extract information. By analyzing the behavior of the hardware, information about what is occurring in the software can be inferred. Malicious VMs can then extract information from other co-resident VMs by monitoring the hardware. This usually takes the form of analyzing the memory and cache latency to transfer information

SIDE-CHANNEL ATTACKS AND DEFENSES:steal information from a target VM

In this **attack(cache-based)**, the length of the overlapping execution times of the target and malicious VMs is important to determine the **bandwidth of the side-channe**l. The longer the VMs are running concurrently, the greater the bandwidth of the side channel.

[**Defense**] To successfully defend, the scheduler can try to limit the overlapping execution times of any two VMs on the system while maintaining an acceptable level of performance(limit the frequency of VM switching which reduces performance).

Another possible defense to disrupt the side-channel is for the hypervisor to **inject noise** into the side channel

XenPump, as an addition to the Xen hypervisor, which is designed to limit the effectiveness of timing channels. XenPump adds random latencies into the system to limit the bandwidth of timing channels.

Prime-trigger-probe Attack: **defense**:defend against this side-channel, they proposed disabling the overlapping aspect of the cache for the VM tenants on a machine. Another option is to flush the cache when switching between VM domains. This adds a 15% performance overhead

PERFORMANCE-BASED ATTACKS:slow down other co-resident VMs or use more than the allocated share of resources to steal from the cloud provider.

Attacker VM yields just before the next scheduler tick. This always ensures that another co-resident VM is running when the scheduler executes. In non-boost mode, this will cause the attacker to never have its CPU credit debited. In boost mode, the hypervisor is unable to tell the difference between a VM awaking after it deliberately yielded and the one waking for an event such as an interrupt. So after the attacker deliberately yields during the scheduler tick, it can preempt the currently running VM.On Amazon's EC2, this can get around their 40% cap for any VM and use up to 85% of the CPU. This can also be used by a malicious user for residency purposes. By having two co-resident VMs run this attack simultaneously, each VM receives 42% of the CPU. With this knowledge, the malicious user starts up two VMs and starts the attack program on each VM. Each VM then measures the CPU share it receives. If it is 85%, then they are not coresident; if it is 42%, then they can assume that they are coresident.

Defense: exact scheduler which is based on a high precision clock to measure CPU usage time whenever a VM yields and goes idle. The other is to use a randomized scheduler

I/O performance based attacks to slow down a co-resident target VM. They deploy specially crafted I/O workloads and manipulate the shared I/O queues to reduce the performance of a target VM. The first task to carry out this attack is to extract the scheduling characteristics of the hypervisor, and then use that information to reduce I/O performance by overloading the I/O resources. **Defense:** fail if there are multiple hard disks installed on the physical machine.

HYPERVISOR ATTACKS AND DEFENSES:

A malicious VM can also attempt to compromise the hypervisor. This is known as virtual machine escape.

Hypervisor Control Flow Integrity:HyperSafe is designed to provide control flow integrity to the hypervisor. The Trusted Platform Module (TPM) is a hardware based module that provides secure storage and secure attestation in addition to cryptographic hashes and signatures. TPM can provide load-time integrity for the hypervisor, but the challenge is to provide run-time integrity. Two techniques are proposed in HyperSafe to solve the runtime integrity checking problem:

1. HyperSafe is to implement a non **bypassable memory lockdown**. This locking down of memory pages prevents any unauthorized writes to the pages. The unlocking process is designed in such a way that it prevents any modifications to hypervisor code and data.
2. **restricted pointer indexing** to essentially add a layer of indirection to all pointers.

Hypervisor Integrity Checking:. HyperSentry uses a software component that is isolated from the hypervisor to provide stealth and in-context integrity checking of the hypervisor. existing hardware and firmware for the software integrity component and isolates it from the hypervisor by using TPM. The key for HyperSentry to work is stealth, so that it will not be vulnerable to the scrubbing attack which removes all evidence of an attack when a measurement by a higher software layer is detected. This is done by using an out of band channel to trigger HyperSentry. The out of band channel used by HyperSentry includes the Intelligent Platform Management Interface (IMPI), System Management Mode (SMM), and Baseboard Management Controller (BMC).IMPI is a platform management interface implemented in

hardware and firmware. All its functions operate completely independently of the CPU and all software on the system. The BMC is installed on the motherboard and is the interface between the remote verifier and the hardware management component. IMPI triggers the SMM which provides a secure environment that the software running on the machine cannot manipulate.

Return Oriented Programming Attack on Hypervisors: using return oriented programming (ROP) can be used to mount a successful attack against the Xen hypervisor. The goal of the ROP attack is to modify the data in the hypervisor that controls the VM privilege level. This way, an attacker can escalate their VMs to a privileged state. **Defense**:continuously analyze the stack, looking for possible ROP attacks and then quarantine for further investigation. They usea key feature of ROP that requires many addresses in the range of a program and its libraries to search for such attacks

Modifying Non-control Data:For success helpful to know the version number of the hypervisor.

1. privilege level data which could escalate a VM to a more privileged state
2. resource utilization data which could let an attacker gain more than their fair share of physical resources
3. security policy data which could let a sensitive VM run on a machine with other VMs that would then attempt to steal its data via side-channels

Defense: use hardware features to only allow certain functions to write the memory locations where the non-control data is located

VM Rollback Attack:assumes that the hypervisor has already been compromised. The compromised hypervisor will then execute a VM from an older snapshot. This happens without the VM owner's awareness. This attack results in a part of the target VM's execution history being lost, which could let the attacker bypass security systems or undo security patches and updates applied to the target VM.
Defense:

1. securely log all the rollback actions, and then the users can audit the log as they see fit. TPM would be used to protect the integrity of the log. This would require four hypercalls for VM boot, VM shutdown, VM suspend, and VM resume, that would all securely log the corresponding action.
2. protected memory to isolate and encrypt the VM's memory pages from the hypervisor, thereby preventing the hypervisor from modifying or reading the pages

VM ISOLATION TECHNIQUES:

1. Secure Turtles: makes use of nested virtualization to protect a guest VM. The Level 0 hypervisor is the most privileged and protects the VM running in Level 2 from the attacks by Level 1, even in the case Level 1 has been compromised.
2. eliminating the hypervisor almost entirely: temporary hypervisor which only runs at initialization that pre-allocates hardware resources such as processor cores and memory, sets up virtualized I/O devices, and avoids indirection to bring the VM in more direct contact with the hardware.
3. A mandatory access control (MAC) policy is described to allow coalitions of VMs to share resources such as network, disk, memory, events, and domain operations. This coalition could even be distributed among multiple hypervisors.It uses bind-time authorization for high performance and includes a Chinese-wall. In the Chinese-wall, each VM is assigned a type, and the types that conflict with each other are not allowed to run concurrently.
4. multi-level set of security requirements: Two types of hypervisors are presented: the pure isolation and sharing hypervisors. The pure isolation hypervisor divides a machine into partitions and does not allow any sharing of resources other than CPU and memory. The sharing hypervisor allows the sharing of files. A high security partition has read-only access to a lower

level security data. The low level partition gets read/write to the same low lever security data. This can be implemented as a one way network.

5.
6.
7.
8.
9.