Towards Automated Security Policy Enforcement in Multi-Tenant Virtual Data Centers

Serdar Cabuk[*], Chris I. Dalton[†], Konrad Eriksson[‡], Dirk Kuhlmann[§], HariGovind V. Ramasamy[¶], Gianluca Ramunno[‖], Ahmad-Reza Sadeghi[**], Matthias Schunter[††] and Christian Stüble[‡‡]

Abstract

Virtual data centers allow the hosting of virtualized infrastructures (networks, storage, machines) that belong to several customers on the same physical infrastructure. Virtualization theoretically provides the capability for sharing the infrastructure among different customers. In reality, however, this is rarely (if ever) done because of security concerns. A major challenge in allaying such concerns is the enforcement of appropriate customer isolation as specified by high-level security policies. At the core of this challenge is the correct configuration of all shared resources on multiple machines to achieve this overall security objective.

To address this challenge, this paper presents a security architecture for virtual data centers based on virtualization and Trusted Computing technologies. Our architecture aims at automating the instantiation of a virtual infrastructure while automatically deploying the corresponding security mechanisms. This deployment is driven by a global isolation policy, thus guarantees overall customer isolation across all resources. We have implemented a prototype of the architecture based on the Xen hypervisor.

Keywords. Virtualization, Virtual Networks, Trusted Computing, Trusted Virtual Domain, Virtual Data Center.

[*]S. Cabuk has been with the Hewlett-Packard Labs, Bristol, UK. Email: serdar.cabuk@gmail.com.

[†]C. I. Dalton is with the Hewlett-Packard Labs, Bristol, UK. Email: cid@hp.com.

[‡]K. Eriksson is with the IBM Zurich Research Laboratory, Rüschlikon, Switzerland. Email: kon@zurich.ibm.com.

[§]D. Kuhlmann is with the Hewlett-Packard Labs, Bristol, UK. Email: dirk.kuhlmann@hp.com.

[¶]H. V. Ramasamy is with the IBM T. J. Watson Research Center, Hawthorne, NY, USA. Email: hvramasa@us.ibm.com.

[‖]G. Ramunno is with the Politecnico di Torino, Turin, Italy. Email: ramunno@polito.it.

[**]A-R. Sadeghi is with the Ruhr-University Bochum, Germany. Email: ahmad.sadeghi@trust.rub.de.

[††]M. Schunter is with the IBM Zurich Research Laboratory, Rüschlikon, Switzerland. Email: mts@zurich.ibm.com.

[‡‡]C. Stüble is with the Sirrix AG Security Technologies, Bochum, Germany. Email: stueble@sirrix.com.

1 Introduction

Hardware virtualization is enjoying a resurgence of interest fueled in part by its cost-saving potential in data centers. By allowing multiple Virtual Machines (VMs) to be hosted on a single physical server, virtualization helps improve server utilization, reduces management and power costs, and controls the problem of server sprawl.

A large number of the companies that outsource their operations are small and medium businesses (SMBs) that cannot afford the costs of a dedicated data center in which all the data center's resources are used to host a single company's IT infrastructure. Hence, the IT infrastructure belonging to multiple SMBs may be hosted inside the same data center facility. Today, even such "shared" data centers operate in the so-called *physical cages* model, wherein different customers' IT infrastructures typically run on distinct physical resources (e.g., different servers or racks), resulting in physical isolation of customers from each other inside the same data center. Resource sharing is typically limited to power and building-related resources.

Moving from the physical cages model to one that allows resource sharing among customers (called the *shared resources* or the *logical cages* model) would enable a more flexible and efficient management of the data center's resources by the infrastructure provider, and thereby result in significant cost benefits. Despite these advantages, the physical cages model remains the status quo primarily because of security concerns and the increased management complexity associated with the logical cages model. From a security point of view, customers today do not trust that the logical cages model can provide sufficient isolation for preventing information leakages across the boundary separating the virtual resources of two customers and for retaining security incidents within that boundary.

Our Contribution: We describe technologies for realizing the logical cages model. At the same time we address the customer isolation and management complexity issues associated with it. Our solution is based on the concept of Trusted Virtual Domains or TVDs [8]. It allows the grouping of VMs belonging to a specific customer dispersed across multiple physical resources into a virtual zone or TVD in which isolation requirements as specified by customer policies[1] are automatically enforced. Isolation policies have various aspects, e.g., storage, networking, and TVD membership. Even if the VMs in a TVD are migrated to different physical platforms (say, for load-balancing purposes), the logical topology of the TVD would still be unchanged. Ensuring the integrity of policy enforcement components and evaluating their trustworthiness are of critical importance. For this purpose we leverage Trusted Computing concepts [40].

Our main contributions are (1) the realization of TVDs using a combination of secure networking and storage virtualization technologies, (2) orchestration of TVDs through a management framework that automatically enforces isolation

[1]Addressing covert channels would exceed the scope of this paper.

among different customer zones, and (3) a reference implementation for virtual data centers based on the Xen hypervisor and Trusted Computing mechanisms. In particular, the management framework takes a significant step forward towards automating the verification, instantiation, and deployment of the appropriate security mechanisms and virtualization technologies. The automation is based on an input security model that specifies the required level of isolation and permitted information flows.

Outline The remainder of the paper is organized as follows. In Section 2 we introduce the key concepts such as Trusted Virtual Domains and survey related work. In Section 3, we explain the security policies that are enforced in a virtual data center. The key idea is to declare an inter-TVD policy that defines how customers are isolated. Each customer can then define an intra-TVD security policy that defines security objectives of TVD-internal data. Both policies are mandatory and can subsequently be refined. In Section 4, we then discuss how these policies are enforced. For enforcement, we focus on the two most important resources that are potentially shared, namely, networking and storage. The key idea here is to enforce the inter-TVD policy also at the resource level by assigning virtual networks and storage to a given domain.

In Section 5, we describe our prototype implementation of a virtual data center. In Section 6, we present the limitations and evaluation of our prototype and discuss some useful lessons learned as a result of our implementation exercise. In Section 7 we conclude the article and summarize our key findings.

2 Background and Related Work

In order to put our work in context we survey key concepts that underlie our approach. Section 2.1 presents the TVD concept, which can be thought of as a virtualization of today's security zones while making security requirements explicit. Section 2.2 describes *Trusted Computing* concepts. The core of this concept is a security hardware device called *Trusted Platform Module* that guarantees certain security functionalities in spite of attacks. We finally survey related work on trusted channels in Section 2.3 and on secure virtual networking in Section 2.4.

2.1 Overview of Trusted Virtual Domains

Bussani *et al.* [8] introduced the concept of TVDs. A Trusted Virtual Domain consists of a set of distributed Virtual Processing Elements (VPEs), storage for the VPEs, and a communication medium interconnecting the VPEs [8, 22, 17]. The TVD provides a policy and containment boundary around those VPEs. VPEs within each TVD can usually exchange information freely and securely with each other. At the same time, they are sufficiently isolated from outside VPEs, including those belonging to other TVDs. Here, isolation loosely refers to the requirement that a dishonest VPE in one TVD cannot exchange information (e.g., by sending messages or by sharing storage) with a dishonest VPE in another TVD, unless the inter-TVD policies explicitly allow such an exchange. There is a TVD *infrastructure* (for each TVD) that provides a unified level of security to member VPEs, while restricting the interaction with VPEs outside the TVD to pre-specified, well-defined means only. Unified security within a virtual domain is obtained by defining and enforcing *membership requirements* that the VPEs have to satisfy before being admitted to the TVD and for retaining membership. Each TVD defines rules regarding information exchange with the outside world, e.g., restrictions regarding in-bound and out-bound network traffic.

Figure 1 shows customer VMs as VPEs belonging to TVD_1 spanning two platforms (contained in the dashed boxes). The Master (TVD1 Master) and Proxy components (Proxy1 on each platform) are part of the TVD infrastructure, which we describe in detail in Section 4.1. The TVD Master is the orchestrator of the TVD deployment and configuration. There is one TVD Proxy for each platform hosting VMs belonging to that TVD. If the platform hosts VMs belonging to multiple TVDs, then there are multiple TVD proxies on that platform, one per TVD. The TVD Proxy on a platform is configured by the TVD Master and can be thought of as the local TVD policy enforcer. VMs belonging to the same TVD can usually exchange information freely with each other unless restricted by VM-level policies. For example, traffic originating from VM_{A1} or VM_{A2} on Host A is routed to VM_{Bi} ($i = 1, \cdots, 4$) on Host B without any restrictions. Information exchange among TVDs can be allowed; however, it is subject to the network and storage policies stated by each TVD Master and locally enforced by each TVD Proxy.

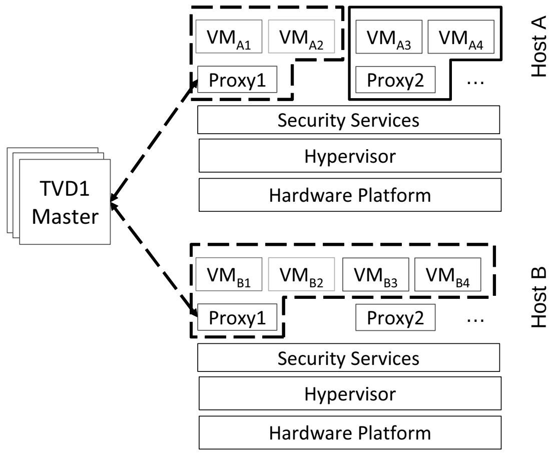

Figure 1: TVD Architecture: High-Level Overview.

In this paper, we provide a realization of TVDs that enforces high-level customer isolation policies by configuring and using low-level networking and storage constructs. We ensure the integrity of the policy enforcement points (e.g., the TVD infrastructure) by leveraging Trusted Computing techniques.

2.2 Trusted Computing – The TCG Approach

It is important to have reliable mechanisms for a system to reason and verify the trustworthiness (i.e., compliance with a certain security policy) of a peer endpoint (local or remote). A recent industrial initiative towards realizing such a mechanism was put forward by the *Trusted Computing Group* (TCG) [40], a consortium of a large number of IT enterprises that proposes a new generation of computing platforms that employs both supplemental hardware and software (see, e.g., [30, 38]). The TCG[2] has published several specifications on various concepts of trusted infrastructures [42].

The Trusted Platform Module The core component the TCG specifies is the *Trusted Platform Module* (TPM). Currently, the widespread implementation of the TPM is a small tamper-evident chip[3] that implements multiple *roots-of-trust* [43, 41], e.g., the root-of-trust for reporting and the root-of-trust for storage. Each root-of-trust enables parties, both local and remote, to place trust on a TPM-equipped platform that the latter will behave as expected for the intended purpose. By definition, the parties trust each root-of-trust, and therefore it is essential that the roots-of-trust always behave as expected. Given that requirement, a hardware root-of-trust – especially one that is completely protected from software attacks and tamper-evident against physical attacks, as required by the TPM specification – is assumed to provide a better protection than software-only solutions.

Attestation and Integrity Verification The Trusted Computing features we leverage in this paper are protection of keys, secure recording of integrity measurements, attestation, and sealing. Integrity verification mechanisms enable a remote party to verify whether system components conform to certain security policies. *Measurement* of a component involves computing the SHA-1 hash of the binary code of that component. In particular, each software component in the Trusted Computing Base (TCB) is first measured and then its measurement recorded before control is passed to it. The hash values are then appended to a hash chain, which is kept in special protected registers called *Platform Configuration Registers* (PCRs), thus acting as accumulators for measurements. *Recording* a measurement means appending it to the hash chain by

[2]TCG's claimed role is to develop, define, and promote open and vendor-neutral industry specifications for Trusted Computing, including hardware building blocks and software interface specifications across multiple platforms and operating environments.

[3]Many vendors already ship their platforms with TPMs (mainly laptop PCs and servers).

PCR extend operation[4]. The sequence of measured values are also stored in a *measurement log*[5], external to the TPM.

TCG proposed *attestation*, also called *binary attestation*, refers to the challenge-response-style cryptographic protocol for a remote party to query the recorded platform measurement values and for the platform to reliably report the requested values. The verifier first sends a challenge to the platform. The platform invokes the TPM_Quote command with the challenge as a parameter. The invocation also carries an indication of which PCRs are of interest. The TPM returns a signed *quote* containing the challenge and the values of the specified PCRs. The TPM signs using an Attestation Identity Key (AIK), whose public key is certified by a third party that the verifier trusts, called *Privacy CA* in TCG terminology. The platform then replies to the verifier with the quote signed by the TPM, along with the AIK public key certificate and the log information that is necessary to reconstruct the platform's configuration. Based on the reply, the verifier can decide whether the platform is in an acceptable state.

Sealing is a TPM operation that is used locally to ensure that a certain data item is accessible only under specific platform configurations reflected by PCR values. The *unsealing* operation will reveal the data item only if the PCR values at the time of the operation match the PCR specified values at the time of sealing.

A more general and flexible extension to the binary attestation is *property-based attestation* [36, 34, 24]: Attestation should only determine whether a platform configuration or an application has a desired property. Other approaches with similar goals have also been proposed: The *semantic remote attestation* [18] uses a language-based trusted VM to remotely attest high-level program properties. The general idea is to use a *trusted VM* (TrustedVM) that verifies the security policy of another virtual machine on a given host. In [27, 28, 29] a software architecture based on Linux is proposed that provides attestation and binding. It allows for binding short-lifetime data (e.g., application data) to long-lifetime data (e.g., the Linux kernel) and for accessing that data only if the system is compatible with a security policy certified by a security administrator.

However, our prototype is based on binary attestation.

2.3 Trusted Channels

The standard approach for establishing secure channels over the Internet is to use security protocols such as Transport Layer Security (TLS) [12] or Internet Protocol Security (IPSec) [23]), which aim at assuring confidentiality, integrity, and freshness of the transmitted data as well as authenticity of the endpoints involved. However, as mentioned before, secure channels do not provide any

[4]Extending PCR values is performed as follows: $PCR_{i+1} := \text{SHA1}(PCR_i|I)$, with the old register value PCR_i, the new register value PCR_{i+1}, and the input I (e.g. a SHA-1 hash value).

[5]Since each PCR holds only the digest of (part of) the chain of trust, keeping the list of all measured values is required if afterwards, during the attestation process, a remote party wants to identify each measured component.

guarantees about the integrity of the communication endpoints, which can be compromised by viruses or Trojans. Based on security architectures that deploy Trusted Computing functionality, one can extend these protocols with integrity reporting mechanisms (e.g., the TLS extension proposed in [16, 5]). Such extensions can be based on binary attestation or on property-based attestation.

2.4 Secure Network Virtualization

Previous work on virtualizing physical networks can be roughly grouped into two categories: those based on Ethernet virtualization and those based on TCP/IP-level virtualization. Although both categories include a substantial amount of work, few of these studies have an explicit focus on security.

A secure network virtualization framework was proposed by Cabuk *et al.* [10] for realizing the network flow aspects of TVDs. The focus of [10] is a security-enhanced network virtualization, which (1) allows groups of related VMs running on separate physical machines to be connected together as though they were on their own separate network fabric, and (2) enforces intra-TVD and inter-TVD security requirements such as confidentiality, integrity, and inter-TVD flow control. This has been achieved by an automatic provisioning of networking components such as VPNs, Ethernet encapsulation, VLAN tagging, and virtual firewalls.

A second concept for managing VLAN access has been proposed by Berger *et al.* in [7]. Both papers contain similar concepts for managing VLANs inside the data center with some differences. The work of Berger *et al.* has more of a focus on integrity assurance using Trusted Computing. The work of Cabuk *et al.* [10] allows provisioning of secure virtual networking even if no VLAN infrastructure is present.

3 Security Policies for Virtual Data Centers

Data centers provide computing and storage services to multiple customers. Customers are ideally given dedicated resources such as storage and physical machines. In the physical cages approach, only few resources such as the Internet connection may be shared between multiple customers. For cost efficiency, our logical cages approach promotes securely extending sharing to other resources such as storage and networks. This is enabled by preventing unauthorized information exchange through shared resources.

To model and implement the logical caging approach, we introduce a domain-based security model for enforcing unified security policies in virtualized data centers. We focus on isolation policies that mimic physical separation of data center customers. Our goal is to logically separate networks, storage, VMs, users, and other virtual devices of one customer from another customer. For our purposes, we define *domain isolation* as the ability to enforce security policies within a domain independently of other domains that may co-exist on the same infrastructure and interact with that domain. The core idea is to use this isolation property as a foundation for guaranteeing desired security properties within each virtual domain while managing shared services under mutually agreed policies.

We now explain the policies that describe this controlled information exchange in a virtualized data center. In Section 4 we describe the individual components that enable us to enforce these policies.

3.1 High-level Policy Model

Our security model is based on TVDs [8], which inherently isolate their resources from resources of other TVDs to successfully enforce their domain policies. A TVD comprises a TVD infrastructure and several members that are managed through this infrastructure. Active elements (subjects) are physical or virtual machines that can be member of one or more TVDs. Passive elements (objects) are resources such as physical or virtual disks that can be member of one or more TVDs. Note that computing platforms that host VMs are not directly member of a TVD.

The security model includes two high-level policies defining the security objectives that must be provided by the underlying infrastructure:

Inter-TVD Policy: By default, each TVD is isolated from the outside world. The high-level information-exchange policy defines whether and how information can be exchanged with other TVDs. If no information flow with other TVDs is permitted, no resources can be shared unless the data center operator can guarantee that the isolation is preserved. If information flow to/from other TVDs is allowed, sub-policies further qualify the exact information flow policy for the individual resources.

Intra-TVD Policy: Domain policies allow TVD owners (e.g., customers) to define the security objectives within their own TVDs. Examples of such policies include how the internal communication is to be protected and under what conditions resources (e.g., storage, machines) can join a particular TVD.

We further define more fine-grained policies by the use of *roles* that can be assigned to any member VM, say to a member machine. This allows us to define and enforce role-based policies within and across TVDs. For example, machines can now assume internal or gateway roles with corresponding permissions; while a workstation may not be allowed to connect to non-TVD networks, machines with the "firewall" role can be allowed to connect to selected other networks. Figure 2 depicts three VMs in a single TVD. Each VM is given different levels of access to resources with respect to its role for that TVD.

3.2 Security Objectives and Policy Enforcement Points

Policies are enforced for all shared resources in the TVD infrastructure (see Figure 3). The basis of all policies is isolation at the boundary of each TVD.

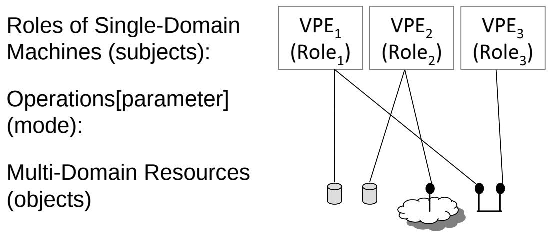
Roles of Single-Domain Machines (subjects):
Operations[parameter] (mode):
Multi-Domain Resources (objects)
VPE$_1$ (Role$_1$)
VPE$_2$ (Role$_2$)
VPE$_3$ (Role$_3$)

Figure 2: Policy Model: Single-TVD Machines operate on Shared Resources

By default, each resource is associated with a single domain. This achieves a basic level of isolation. If information flow between TVDs is allowed, resources can also be member of different TVDs. For example, a TVD can allow certain types of resources on certain hosts to provide services also to other domains. Each TVD defines rules regarding in-bound and out-bound information flow for restricting communication with the outside world. The underlying policy-enforcement infrastructure then has to ensure that only resources trusted by all TVDs are shared.

Architecturally, there are two ways of enforcing such rules, depending on the trust between the TVDs. The first method involves two shared resources connected by an intermediate domain. In this method, each TVD enforces its side of the flow control by means of its own shared resource. An example of this type of connection is the one that exists between $TVD\ A$ and $TVD\ B$ in Figure 3. This method is used when the trust level between $TVD\ A$ and $TVD\ B$ is low, and the two cannot agree on a shared resource that is mutually trusted. The shared resource in $TVD\ A$ will enforce $TVD\ A$'s policies regarding in-bound traffic from $TVD\ B$, even if the shared resource in $TVD\ B$ does not enforce $TVD\ B$'s policies regarding out-bound traffic. The shared resources can be thought of as being a part of a "neutral" TVD ($TVD\ AB$) with its own set of membership requirements. The second method that requires shared trust is to establish one or more shared resources that are accessed from both TVDs while allowing controlled information flow. This mechanism is used between $TVD\ B$ and $TVD\ C$ in Figure 3.

Security within a virtual domain is finally obtained by defining and enforcing *membership requirements* that resources have to satisfy prior to being admitted to the TVD and for retaining the membership. This may also include special requirements for different machine types: Because, for example, shared resources play a key role in restricting information flow between TVDs, the software on those machines may be subject to additional integrity verification as compared to the software on regular VMs.

3.2.1 Permitted Flows in Data Centers

At a high level flow control policies define the allowed traffic flow between two domains and how the domains should be protected. Allowed information flows

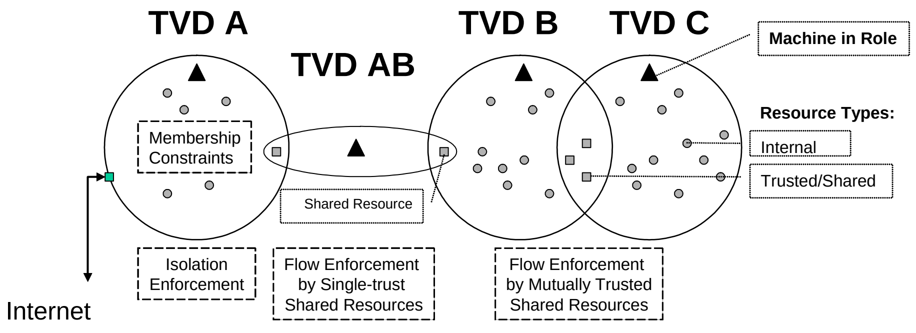

Figure 3: Usage Control for Shared Resources: Machines use resources belonging to TVDs.

From / to	D_I	D_D	D_i
D_I	**1**	1	0
D_D	0	**1**	1
D_i	0	1	**1**

Table 1: High-level Directed Flow Control Matrix for Internet D_I, DMZ D_D, and Intranet D_i.

can be represented by a simple flow control matrix as depicted in Table 1, where 1 allows information flow and 0 denies it. This example implements a basic enterprise policy that regulates incoming flow from untrusted outside entities (D_I) through a semi-trusted intermediary domain (D_D), and disallows any outgoing flow. Note that this matrix is directional, i.e., it might allow flows in one direction but not in the opposite direction. If flow policies between two TVDs are asymmetric, only shared resources that can enforce these policies are permitted.

Device-specific policies (network, storage) can then refine these basic rules. If an information flow is not permitted, then also shared resources are not permitted between these TVDs.

3.2.2 Membership Requirements

Membership requirements define under what conditions resources may join a domain. From a high-level policy perspective, several criteria can be applied to decide whether an entity is allowed to join a domain, for example:

- *Certificates*: An authority defined by the TVD policy can certify a resource to be member of a TVD. A common example is that an enterprise issues machine certificates to allow its machines to join the corporate network.
- *Integrity Proofs*: A resource may prove its right to join a TVD using

integrity proofs. It may, e.g., prove that the integrity of the base operating system is intact and that all required patches have been applied [39].

- *User-identity*: Only machines operated by a certain user can join. This can be validated by user-name/password or by a cryptographic token.

In general, a resource may need to show proper credentials to prove that it fulfills certain properties before allowing the resource to join the TVD (see, e.g., [36]). More formally, a machine m is permitted to join a TVD t if and only if there is at least one property of m that satisfies each security requirement of t. The validations of these properties are usually done on a per-type and role basis. For example, requirements for a shared resource are usually stronger than the requirements for a TVD-internal resource.

3.3 Example Policy Refinements for Protected Resources

Policies alone are not sufficient to enforce customer separation in a virtualized data center. Ultimately, one needs to transform these policies into data center configurations and security mechanisms specific to each resource (e.g., VLAN configuration). To do so, we introduce a policy management scheme that accepts high-level domain policies and transforms them into resource-specific low-level policies and configurations. In Section 5 we demonstrate a prototype based on this architecture that enforces high-level TVD policies by lower-level network and infrastructure configurations, which is then deployed onto each physical platform to assist customer separation.

3.3.1 Refinement Model

The high-level policy defines the basic flow control, protection, and admission requirements. We aim at enforcing these high-level objectives throughout all resources in the data center.

In the high-level model, flow control across customer domains is specified by a simple matrix such as the one in Table 1 that defines whether flows are permitted. This however is not sufficiently fine-grained for specific resources. TVDs, for example, want to restrict their flow across boundaries by means of firewall rules. As a consequence, we need to introduce a notion of policy refinement [44], because as translation moves towards lower levels of abstraction, it will require additional information (e.g., physical arrangement of the data center, "subjective" trust information) to be correctly and coherently executed.

Our notion of policy refinement mandates the enforcement of "no flow" objectives while allowing each resource to refine what it means so that flows are permitted and how exactly unauthorized flows shall be prevented. Similarly, we do not allow resources to deviate from the confidentiality/integrity objectives; however, certain resources can be declared trusted so that they may enforce these objectives without additional security mechanisms such as encryption or authentication.

Flow to →	D_I		D_D		D_i	
Enforced by ↓	gate	internal	gate	internal	gate	internal
D_I	**1**	**1**	P_{ID}	0	**0**	**0**
D_D	0	0	**1**	**1**	P_{Di}	0
D_i	**0**	**0**	P_{Di}	0	**1**	**1**

Table 2: Example Network Flow Control Policy Matrix for Three TVDs.

Similarly, the fact that admission is restricted is then refined by specific admission control policies that are enforced by the underlying infrastructure.

Note that conflict detection and resolution [44, 26] can later be used to extend this simple notion of refinement. However, we currently stay on the safe side: Connections are only possible if both TVDs allow them. Similarly, if one domain requires confidentiality, information flows are only allowed to TVDs that also require confidentiality. Other schemes for more elaborate flow control have been proposed in [14, 9, 13, 15].

3.3.2 Network Security Policies

We now survey the policy model of [10] and show how it is related to the corresponding high-level policy. Similar to our high-level policies, there are two types of policies governing security in the network. The first limits flow between networks, whereas the second defines membership requirements to each network.

Network Security Policies across TVDs A policy covers isolation and flow control between TVDs as well as integrity and confidentiality against outsiders. These basic security requirements are then mapped to appropriate policies for each resource. For example, from a networking perspective, isolation refers to the requirement that, unless the inter-TVD policies explicitly allow such an information flow, a dishonest VM in one TVD cannot (1) send messages to a dishonest VM in another TVD (information flow), (2) read messages sent on another TVD (confidentiality), (3) alter messages transmitted on another TVD (data integrity), and (4) become a member of another TVD network (access control).

TVDs often constitute independent organizational units that may not trust each other. If this is the case, a communication using another TVD can be established (see the communication between TVD A and B in Figure 3). The advantage of such a decentralized enforcement approach is that each TVD is shielded from security failures in other TVDs. For networks, the main inter-TVD security objectives are controlled information sharing among the TVDs as well as integrity and confidentiality protection of the channel.

While the high-level model specifies whether information exchange is allowed between domains or not, we now refine this policy as follows:

- We refine the active elements (subjects) of given domains by introducing roles that machines can play. This allows us to set different permissions to boundary machines as compared to internal machines.

- In case information flow is permitted in principle, we refine the network security policies by introducing flow control rules that can further restrict the actual information exchange. A network policy may disallow flow even though it has been allowed from a high-level policy perspective.

An information flow control matrix is a simple way of formalizing these network connectivity objectives. Table 2 shows a sample matrix for the three example TVDs introduced earlier. Each matrix element represents a policy specifying permitted connections between a pair of TVDs, as enforced by one of the TVDs. The depicted policies P_x that limit information exchange will be implemented by firewall rules that are used to program the boundary firewalls. The **1** values along the matrix diagonal convey the fact that there is free information exchange within each TVD. The **0** values in the matrix are used to specify that there should be no direct information flow between two TVDs, e.g., between the Internet D_I and the intranet D_i. Care must be taken to ensure that the pairwise TVD policies specified in the information flow control matrix do not accidentally contradict each other or allow undesired indirect flow.

Intra-TVD Network Security Policy Within a TVD, all VMs can freely communicate with each other while observing TVD-specific integrity and confidentiality requirements. For this purpose, the underlying infrastructure may ensure that intra-TVD communication only takes place over an authenticated and encrypted channel (e.g., IPSec), or alternatively, a trusted network[6].

3.3.3 Towards Storage Security Policies

Virtual disks attached to VMs must retain the advantages offered by storage virtualization while at the same time enforcing TVD security policies. Advantages of storage virtualization include improved storage utilization, simplified storage administration, and the flexibility to accommodate heterogeneous physical storage devices. Similar to network, we now show a refinement of the high-level TVD policies into access control policies for VMs in certain roles to disks belonging to a domain.

Inter-TVD Storage Security A virtual disk has a single label corresponding to the TVD it belongs to. Whenever a virtual machine operates on virtual storage, the global flow matrix described in Section 3 needs to be satisfied. For flexibility, each TVD can define a set of storage policies that govern usage and security of its storage. A single policy is then assigned to and enforced for each storage volume.

[6]A network is called *trusted* with respect to a TVD security objective if it is trusted to enforce the given objective transparently. For example, a server-internal Ethernet can often be assumed to provide confidentiality without any need for encryption.

Flow to →	D_I		D_D		D_i	
Disk ↓	gate	internal	gate	internal	gate	internal
D_I	r/w	r/w	w	0	**0**	**0**
D_D	r	0	r/w	r/w	r/w	0
D_i	**0**	**0**	r/w	0	r/w	r/w
Blank	$r/$	0	$r/$	0	$r/$	0
	$w \rightarrow D_I$	0	$w \rightarrow D_D$	0	$w \rightarrow D_i$	0

Table 3: Example of a Refined Disk Policy Matrix for Three TVDs.

As the starting point of our storage policy refinement, we define a *maximum permission policy* as follows:

1. Any machine in domain TVD_A playing any role can write to a disk of domain TVD_B iff flow from domain TVD_A to domain TVD_B is permitted.

2. Any machine in domain TVD_A playing any role can read from a disk of domain TVD_B iff flow from domain TVD_B to domain TVD_A is permitted.

3. Any single machine in any domain can read/write mount a blank disk. After data is written, the disk changes ownership and is now assigned to the domain of the machine which has written data.

Table 3 shows the resulting maximum disk access control policy. Actual policies are then valid with respect to a maximum-permission policy for a domain if they permit a subset of its permissions. Note that as flow within a domain is always allowed, this implies that disks of the same domain as the machine may always be mounted read/write.

Intra-TVD Storage Security By default, we consider the content of a disk to be confidential while the storage medium (possibly remote) is deemed to be untrusted. As a consequence, if a given domain does not declare a given storage medium as trusted, we deploy whole-disk encryption using a key that is maintained by the TVD infrastructure.[7] Another aspect reflected in the disk policies is the fact that we have a notion of blank disks. Once they are written by another domain, they change color, and are then associated with this other domain while being encrypted under the corresponding key. In the future, it would be desirable to have integrity-protected storage [11, 33] where the TVD can validate that its content has not been changed by untrusted entities.

For protecting the data in a particular TVD, virtual storage may in addition specify which conditions on the system must be satisfied before a disk may be *re-mounted* by a VM that has previously unmounted the disk, and whether shared mounting by multiple systems is allowed. Note that these membership

[7]Note that the VM only sees unencrypted storage, i.e., the TVD infrastructure automatically loops in encryption.

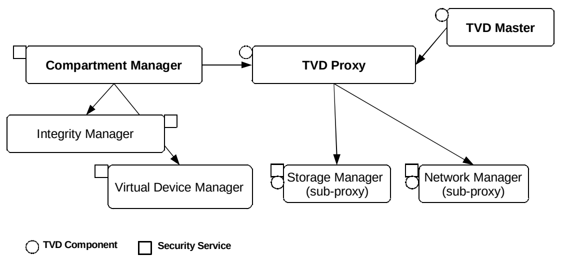

Figure 4: TVD Components and Security Services.

restrictions require bookkeeping of disks and management of access of VMs to disks.

4 Unified Policy Enforcement for Virtual Data Centers

In this section, we introduce a TVD-based policy enforcement framework that orchestrates the deployment and enforcement of the type of policies we presented in Section 3 across the data center. Existing storage and network virtualization technologies as well as existing Trusted Computing components (in software and hardware) are the building blocks of our solution. Our framework (1) combines these technologies to realize TVDs and (2) orchestrates them using the TVD infrastructure, which provisions the appropriate security mechanisms.

4.1 TVD Infrastructure

The TVD infrastructure consists of a management layer and an enforcement layer. The TVD management layer includes TVD masters, proxies, and factories, whereas the TVD enforcement layer consists of various security services. Each TVD is identified by a unique *TVD Master* that orchestrates TVD deployment and configuration. The TVD Master can be implemented as a centralized entity (as in our prototype described in Section 5) or have a distributed fault-tolerant implementation. The TVD Master contains a repository of high-level TVD policies and credentials (e.g., VPN keys). The Master also exposes a TVD management API through which the TVD owner can specify those policies and credentials. In the deployment phase, the TVD Master first verifies the suitability and capability of the physical host (which we refer to as pre-admission control). It then uses a generic *TVD Factory* service to spawn a *TVD Proxy*, which acts as the local delegate of the TVD Master dedicated to that particular host. The TVD Proxy is responsible for (1) translation of high-level TVD policies into low-level platform-specific configurations, (2) configuration of the host and security services with respect to the translated policies, and (3) interaction with the security services in TVD admission and flow control.

Security services implement the security enforcement layer of our TVD infrastructure. They run in a trusted execution environment on each physical host (e.g., Domain-0 in Xen) and (1) manage the security configuration of the hypervisor, (2) provide secure virtualization of resources (e.g., virtual devices) to the VMs, and (3) provide support to TVD proxies in enforcing flow and access control policies within and across TVD boundaries. Figure 4 shows a high-level list of security services and their interaction with the TVD components. Most importantly, the *compartment manager* service manages the life-cycle of VMs in both para-virtualized and fully virtualized modes. This service works in collaboration with the TVD Proxy to admit VMs into TVDs. The *integrity manager* service implements Trusted Computing extensions and assists the TVD Proxy in host pre-admission and VM admission control. The *virtual network manager* and *virtual storage manager* services are invoked by the TVD Proxy. They implement resource virtualization technologies and enforce parts of the high-level TVD policies that are relevant to their operation. Lastly, the *virtual device manager* service handles the secure resource allocation and setup of virtual devices assigned to each VM.

Our TVD infrastructure is geared towards automated deployment and enforcement of security policies specified by the TVD Master. Automated refinement and translation of high-level policies into low-level configurations are of particular interest. For example, for information flow between two hosts in a trusted data center environment, other mechanisms need to be in place than for a flow between two hosts at opposite ends of an untrusted WAN link. In the latter case, the hosts should be configured to allow communication between them only through a VPN tunnel.

Another important consideration is policy conflict detection and resolution [44, 26]. In fact, conflicting high-level policies (e.g., a connection being allowed in the inter-TVD policy but disallowed in the intra-TVD policy) can potentially result in an incorrect configuration of the underlying infrastructure. We cannot solely rely on the TVD owner to specify conflict-free policies. It is important to detect policy conflicts and provide feedback to the owner in case one is detected. In the present prototype, policy refinement is performed manually. The result is a set of configuration files that we use for configuring the security services at the policy enforcement layer (e.g., the virtual networking infrastructure). In future work, we will investigate the automation of this step using, for example, the IETF policy model [35] and various graph-based mechanisms from the literature. We will also investigate different techniques for resolving conflicting policies [14, 9, 13, 15].

4.2 Virtual Networking Infrastructure

Virtual networking (VNET) technologies enable the seamless interconnection of VMs that reside on different physical hosts as if they were running on the same machine. In our TVD framework, we employ multiple technologies, including virtual switches, Ethernet encapsulation, VLAN tagging, and VPNs, to virtualize the underlying network and securely group VMs that belong to the same

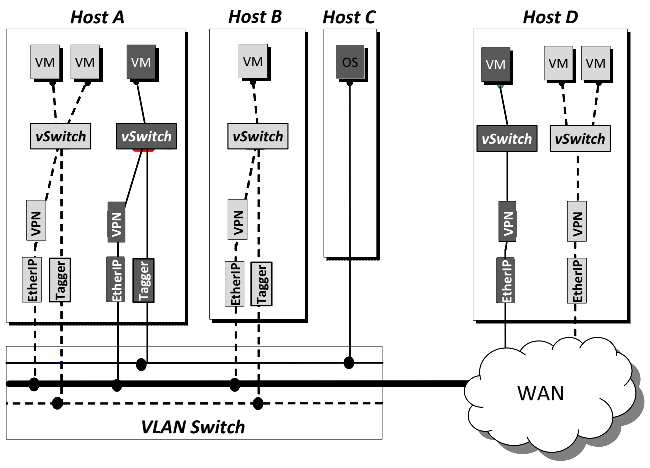

Figure 5: General vSwitch Architecture.

TVD. A single private virtual network is dedicated to each TVD, and network separation is ensured by connecting the VMs at the Ethernet level. Logically we provide a separate "virtual infrastructure" for each TVD in which we control and limit the sharing of network resources (such as routers, switches) between TVDs. This also provides the TVD owner with the freedom to deploy a wide range of networking solutions on top of the TVD network infrastructure. Network address allocations, transport protocols, and other services are then fully customizable by the TVD owner and work transparently as if the VMs were in an isolated physical network. To maintain secrecy and confidentiality of network data (where necessary), network communication is established over encrypted VPN tunnels. This enables the transparent use of untrusted networks between physical hosts that contain VMs within the same TVD to provide a seamless view of the TVD network.

In this section, we introduce the technologies we use to implement a security-enhanced VNET infrastructure for TVD owners. The concept of virtual switching is central to our architecture, which is then protected by existing VPN technologies that provide data confidentiality and integrity where needed. The VNET infrastructure acts as the local enforcer of VNET policies. As described in Section 3.3.2, these policies are based on the high-level TVD policies and translated into network configurations by the TVD Proxy. The Proxy then deploys the whole VNET infrastructure with respect to the translated configuration.

4.2.1 Virtual Switching

The *virtual switch* (vSwitch) is the central component of the virtual networking infrastructure and operates similarly to a physical switch. It is responsible for network virtualization and isolation, and enables a virtual network to span multiple physical hosts. To do so, the vSwitch uses EtherIP [20] and VLAN tagging [19] to insert VLAN membership information into every network packet. The vSwitch also implements the necessary address-mapping techniques to direct packets only to those machines that host member VMs. Virtual switches provide the primitives for implementing higher-level security policies for networking and are configured by the higher-level TVD management layer.

Figure 5 illustrates an example architecture in which physical machines host multiple VMs with different TVD memberships (the light and dark shades indicate different TVDs). Hosts A, B, and D host virtual machines in contrast to Host C which is a physical machine. Furthermore, Hosts A, B, and C reside on the same LAN, and thus can communicate directly using the trusted physical infrastructure without further protection (e.g., traffic encryption). For example, the *light* VMs hosted on Hosts A and B are inter-connected using the local VLAN-enabled physical switch. In this case, the physical switch separates the TVD traffic from other traffic passing through the switch using VLAN tags. Similarly, the *dark* VMs hosted on Host A and the non-virtualized Host C are seamlessly inter-connected using the local switch. In contrast, connections that require IP connectivity are routed over the WAN link. The WAN cloud in Figure 5 represents the physical network infrastructure able to deal with TVD-enabled virtual networks; it can include LANs with devices capable of VLAN tagging and gateways to connect the LANs to each other over (possibly insecure) WAN links. For connections that traverse an untrusted medium, we employ EtherIP encapsulation to denote TVD membership and additional security measures (such as encryption) to ensure compliance with the confidentiality and integrity requirements.

4.2.2 Virtual Private Networking

In Figure 5, VMs hosted on Host D are connected to the other machines over a WAN link. A practical setting in which such a connection might exist would be an outsourced remote resource connected to the local data center through the Internet. As an example, lightly shaded VMs on Host D connect to the lone VM on Host B over this untrusted link. In this setting, we use a combination of EtherIP encapsulation and VPN technology to ensure the confidentiality and integrity of the communication. To do so, we use point-to-point VPN tunnels with OpenVPN that are configured through the TVD Proxy from the TVD policies. This enables reconfiguration of the topology and the involved VPNs within a TVD from a single administration point, the TVD Master.

TVD policies distributed from the TVD Master to the TVD Proxy also include the secret key for the VPN along with other VPN-specific settings. On a physical host, the VPN's endpoint is represented as a local virtual network interface (vif) that is plugged into the appropriate vSwitch controlled by the TVD Proxy. The vSwitch then decides whether to tunnel the communication between VMs, and if so, uses the VPN module to establish the tunnel and access the VPN secret for traffic encryption and decryption.

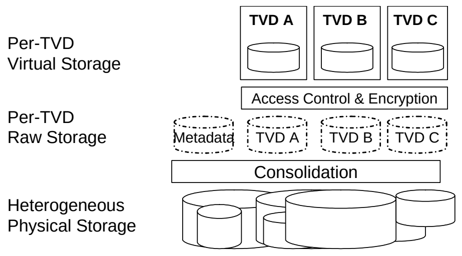

Figure 6: Security Enforcement for Virtualized Storage.

4.3 Virtual Storage Infrastructure

We focus on a simplified security management of virtualized storage. Broadly speaking, storage virtualization abstracts away the physical storage resource(s). It is desirable to allow a storage resource to be shared by multiple host computers, and also to provide a single storage device abstraction to a computer irrespective of the underlying physical storage, which may be a single hard disk, a set of hard disks, a Storage Area Network (SAN), etc. To satisfy both requirements, storage virtualization is typically done at two levels. The first level of virtualization involves aggregating all the (potentially heterogeneous) physical storage devices into one or more virtual storage pools. The aggregation allows more centralized and convenient data management. The second level of virtualization concerns the unified granularity (i.e., blocks or files) at which data in each pool is presented to the higher-level entities (operating systems, applications, or VMs).

Figure 6 shows our storage security enforcement architecture, in which existing heterogeneous physical storage devices are consolidated into a joint pool. This virtual storage pool is then subdivided into raw storage for each TVD. Each raw storage volume has an owner TVD that determines its policy (indicated by the labels TVD A, TVD B, and TVD C at the per-TVD raw storage layer in the figure). In addition, when a volume shall be shared among multiple TVDs, there is also a set of member TVDs associated with it. The access control and encryption layer helps enforce the storage-sharing policy defined by the owner TVD, e.g., enforcing read, write, create, and update access permissions for the member TVDs. This layer is a logical layer that in reality consists of the virtual storage managers (part of the security services) located on each physical platform. The virtual storage manager on each physical platform is responsible for enforcing the owner TVD's storage security policies (see Section 3.3.3) on these volumes. If a certain intra-TVD security policy requires confidentiality and does not declare the medium as trusted, the disk is encrypted using a key belonging to the owner TVD.[8] If conditions for (re-)mounting a disk have been defined, the disk is also encrypted and the key is sealed against the TCB while including these conditions into the unsealing instructions. The policy and meta-

[8]For efficiency reasons, we currently do not provide integrity protection.

data are held on a separate raw volume that is only accessible by the data center infrastructure.

An administrator of a domain may request a disk to be mounted to a particular VM in a particular mode (read/write). In Xen, the disk is usually mounted in the management machine Domain-0 as a *back-end device* and then accessed by a guest VM via a *front-end* device. The virtual storage manager on the platform validates the mount request against the policies of both the TVD the VM is part of and the owner TVD for the disk. Once mounted, appropriate read-write permissions are granted based on the flow control policy for the two TVDs, e.g., read access is granted only if the policies specified in the disk policy matrix allow the VM's TVD such an access to the disk belonging to the owner TVD.

4.4 TVD Admission Control

When a VM is about to join a TVD, different properties will be verified by the local TVD Proxy to ensure that policies of all the TVDs that the VM is currently a member of as well as of the TVD that it wants to join are not violated. If the verification is successful, then the VM will be connected to that TVD. The TVD admission control protocol is the procedure by which the VM gets connected to the TVD. In the case of a VM joining multiple TVDs, the admission control protocol is executed for each of those TVDs. We now describe the steps of the protocol.

We assume that the computing platform that executes the VM provides mechanisms that allow remote parties to convince themselves about its trustworthiness. Example mechanisms include trusted (authenticated) boot and the remote attestation protocol (see Section 2.2) based on TPM technology.

TVD Proxy Initialization Phase: To allow a VM to join a TVD, the platform hosting the VM needs access to the TVD policy, and upon successful admission, to TVD secrets, such as the VPN key. For this purpose, TVD Proxy services are started on the platform for each TVD whose VMs may be hosted. The TVD Proxy can be started at boot time of the underlying hypervisor, by a system service (TVD Proxy Factory), or by the VM itself, as long as the TVD Proxy is strongly isolated from the VM.

Pre-Admission Phase: When a VM wants to join a TVD that is going to be hosted on the platform for the first time, the TVD Master has to establish a trust relationship with the platform running the VM, specifically with the TVD Proxy. We call this step the *pre-admission* phase. It involves the establishment of a trusted channel (see Section 2.3) between the TVD Master and the TVD Proxy (or the TVD Proxy Factory). The trusted channel allows the TVD Master to verify the integrity of the TVD Proxy (Factory) and the underlying platform. After the trusted channel has been established and the correct configuration of the Proxy has been verified, the TVD Master can send the TVD policies and credentials (such as a VPN key) to the TVD Proxy.

Admission Control Phase: The Compartment Manager (part of the platform security services shown in Figure 4) is responsible for starting new VMs. The Compartment Manager loads the VM configuration and enforces the security directives with the help of the Integrity Manager (also part of the platform security services shown in Figure 4). The security directives may include gathering the VM state information, such as the VM configuration, kernel, and disk(s) that are going to be attached to the VM.

If the VM configuration states that the VM should join one or more TVDs, then the Compartment Manager interacts with the corresponding TVD Proxy(ies) and invokes TPM functions to attest the state of the VM. The TVD Proxy verifies certain properties before allowing the VM to join the TVD. More concretely, the TVD Proxy has to ensure that

- the VM fulfills the integrity requirements of the TVD;
- the information flow policies of all TVDs the VM will be a member of will not be violated;
- the VM enforces specific information flow rules between TVDs if such rules are required by the TVD policy, and that
- the underlying platform (e.g., the hypervisor and attached devices) fulfills the security requirements of the TVD.

Platform verification involves matching the security requirements with the platform's capabilities and mechanisms instantiated on top of these capabilities. For example, suppose that data confidentiality is a TVD requirement. Then, if hard disks or network connections are not trusted, additional mechanisms, such as block device encryption or VPN (respectively), need to be instantiated to satisfy the requirement.

TVD Join Phase: If the VM and the provided infrastructure fulfill all TVD requirements, a new network stack is created and configured as described in Section 4.2. Once the Compartment Manager has started the VM, it sends an attach request to the corresponding TVD vSwitch through the TVD Proxy. Once the VM is connected to the vSwitch, it is a member of the TVD.

5 Prototype Implementation

5.1 Overview

We realized a prototype of a physical data center implementing multi-tenant virtual data centers. Each one, usually owned by a different customer, is mapped onto a different TVD.

Our prototype is based on two classes of building blocks: Physical hosts providing VMs for the virtual domains and hosts running the data center services.

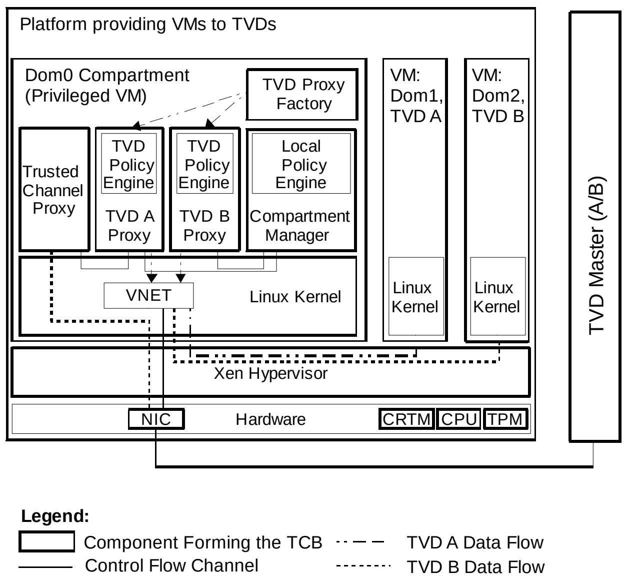

Figure 7: Xen Architecture for TVD.

Figure 7 shows a host of the first class, a single secure hypervisor platform (that we henceforth refer to as “the platform”) that is based on the Xen architecture [6] (Xen 3.1.1 and Linux kernel 2.6.22). Following the Xen model, the platform consists of a core hypervisor, the privileged management VM called Domain-0, and the guest VMs. We have also implemented the TVD Master role that runs on a second class host and maintains policies and membership for a given virtual domain.

The following components are provided by our platform. The *Compartment Manager* manages the life-cycle and the integrity of the guest VMs (also called compartments[9]). The *TVD Proxy* is the local representative of the *TVD Master* running on the platform and enforcing the policies of that TVD. Each TVD Proxy is created and spawned by a *TVD Proxy Factory*, whenever a TVD needs to be extended to a new host. The *Secure Virtual Network subsystem* creates, manages, and makes secure the virtual LAN for each TVD. The *Trusted Channel Proxy* implements the Trusted Channel needed for the pre-admission phase. In the current implementation, all these components run in Domain-0. Together with the Xen hypervisor and the other services in Domain-0, they constitute the Trusted Computing Base (TCB) of the platform, i.e., the integrity of these components must be ensured to guarantee the correct enforcement of the TVD policies.

The integrity measurements of the TCB components are performed during

[9]A compartment is a protected execution environment, subject to information flow control policies enforced by the hypervisor: a compartment can securely host a single process running with threads as well as full-fledged VMs.

the authenticated boot, when the chain of trust begun by the Core Root of Trust for Measurement (CRTM) is extended using TrustedGRUB [3], a TCG-enhanced version of the standard GRUB boot loader. The TVD Master implements one end of the Trusted Channel establishment protocol. The other end is the TVD Proxy Factory.

Figure 8 shows the simplified layout of our prototype data center. It has two virtual networks per customer: A management network for managing the customer's VMs within the TVD and a user data network for the actual communication between client VMs. Furthermore, the data center has a DMZ for external connections, a virtual data center management network (VDC) that is used for communication between our data center management components, and finally a SAN network that provides storage to the individual platforms.

Each virtual domain has an intra-domain management VM that runs the management software of the customer and connects to the management network. This management software interacts with a filtered XenAPI (called XenAPI') that hides infrastructure and machines belonging to other virtual domains and provides a virtual view of the data center. Each administrator in a virtual domain can then define and start any number of guest VMs that get connected to the corresponding data network.

5.2 Security Services

In [21], we have described a Xen-based prototype of our security services for integrity management and obtaining compliance proofs. The prototype enables the protection of security policies against modification and allows stakeholders to verify the policies actually implemented. The paper also describes multiple use cases, in which we demonstrated the policy enforcement and compliance-checking capabilities of our implementation. For example, we showed how to validate the configuration of the virtual networking subsystem on each host (assuming that a TPM is embedded in each host).

Here, we provide an overview of our security services implementation. The Compartment Manager (CM) is responsible for the VM life-cycle management. As the sole entry point for VM-related user commands, it communicates directly with the hypervisor and orchestrates the Integrity Manager (IM) and the secure device virtualization functions of the platform. These functions base on the standard Xen management tools.

The CM through the (IM) is responsible for verifying the integrity of the VMs. The root file system is made available to each VM, including Domain-0, through a pair of partitions or virtual disks. One of them is read-only, contains the initial root file system, and is measured together with the VM configuration file; the resulting whole integrity measurement of the VM is then accumulated into the TPM by extending one PCR. The other partition or virtual disk, which is read/write and empty at the beginning of the VM life, records the changes occurring on the root file system while the VM is running. This is done through the copy-on-write mechanism provided by *unionfs* [4, 1] which allows the stacking of multiple file systems and provides the operating system with a unified view. CM and IM are also responsible for guaranteeing the confidentiality of a VM's data (stored on the second read/write image) by encrypting it using *dm-crypt* [2], which is the Linux device mapper with support for encrypting block devices (e.g., physical or virtual disks and partitions). Next, the encryption key is sealed against the measurement of the TCB and of the VM (stored in a set of PCRs). We use this "sealed disk" scheme to protect the VM's confidential data. This scheme can be applied to all disks, except the root image, depending on the security requirements given.

The TVD policy defines, for each VM, the disks that need to be measured, the PCR(s) in which the measurements need to be stored, and the disks that need to be sealed. Once the policy has been executed and all disks have been prepared (measured/unsealed), the admission protocol involving the CM and the TVD Proxy (see Sections 4.4 and 5.3) follows. Then the CM requests Xen to start the VM. To manage VMs, the CM maintains the association between a running VM and the policy that was used to start it.

5.3 TVD Master, Proxies and Proxy Factories

The TVD policy of our prototype lists all VMs that can potentially be admitted to the TVD. Each VM is identified by the Xen domain identifier. For each VM, the policy specifies the required integrity measurement value. Only if the VM's actual measurement value (obtained by the CM) matches the required value the VM will be admitted to the TVD. The policy also specifies the MAC address assigned to the VM's virtual NIC, if the admission is successful. Moreover, it identifies the VLAN corresponding to the TVD, and the VPN keys needed for establishing secure connections within the TVD. Currently one shared key per TVD is used to protect all insecure links.

When a VM wants to be admitted to the TVD (i.e., this is stated in the configuration file of the VM), the related TVD Proxy is contacted by CM using a stateless TCP-based protocol called Compartment Admission Protocol (CAP). Since a platform can host VMs belonging to different TVDs, the CM contacts the TVD Proxy Factory to obtain the TVD Proxy end-point for the requested TVD. If such TVD Proxy is not running yet, the TVD Proxy Factory creates and spawns it. Before starting the VM, CM measures it (as explained in Section 5.2) and asks the TVD Proxy whether the VM can be admitted to the TVD by passing the identifier and the measurement of the VM. If the answer is positive, CM receives the MAC address specified in the policy from the TVD Proxy, creates the back-end network device (see Section 5.4 for further explanations about Xen back-end and front-end devices), and sets the MAC address for the front-end device. Finally, the CM requests the TVD Proxy to attach the VM to the virtual switch (vSwitch) of the VLAN corresponding to the TVD by specifying the identifier, measurement value, and the names of back-end devices for the VM being admitted. In the case of a negative response from the TVD Proxy, the CM can be configured to either start the VM even though it will not be connected to the TVD VLAN or not to start it at all.

5.4 Secure Virtual Network subsystem

The prototype implementation of our Secure Virtual Network subsystem has been documented in [10] and has been integrated in the prototype being presented in this paper. Our networking extensions consist of vSwitches, VLAN tagging, and LAN encapsulation modules. They are implemented as kernel modules in Domain-0, which also acts as the driver VM for the physical NIC(s) of each physical host.

To specify the particular vSwitch and the particular port in the vSwitch to which a VM's Xen back-end device must be attached, the Xen VM configuration file is used. This file is generated by CM after having received information (MAC address and VLAN identifier) from the TVD Proxy. We use additional scripts to specify whether a particular vSwitch should use one or both of the VLAN tagging and encapsulation mechanisms for isolating separate virtual networks.

The vSwitches maintain a table mapping virtual network devices to ports on a particular vSwitch. The encapsulation module implements EtherIP processing for packets coming out of and destined for the VMs. The VLAN segments associated with different TVDs and the corresponding vSwitches are assigned unique identifiers. The network identifier field in the EtherIP packets is set to the identifier of the vSwitch that the target VM is attached to.

The VLAN tagging module tags the packet with the VLAN identifier corresponding to the VLAN that the target VM is a part of. At the destination platform, the VLAN module removes the VLAN tags, and routes the packets to the appropriate vSwitch based on the VLAN tag.

5.5 Trusted Channel Proxies

The Trusted Channel between TVD Proxy Factory and TVD Master used during the pre-admission phase is set up by means of a pair of services called Trusted Channel proxies. They implement the Trusted Channel at the application layer via a TLS tunnel, made available to TVD Proxy Factory and Master once remote attestation has been successful. The remote attestation is done by performing the `TPM_Quote` operation, namely, digitally signing a set of PCRs and a challenge received from the remote attester using a TPM asymmetric key. The latter can be certified as Attestation Identity Key (AIK) by a Privacy CA. The result of the `TPM_Quote` operation (i.e. the signature over a set of PCR values), the actual PCR values and the AIK Public Key Certificate are sent by the TVD Proxy Factory to the TVD Master to be verified. If the verification is successful, then the remote attestation can be considered successful, and the two proxies start tunneling the incoming TCP packets through the TLS channel. An alternative implementation for attesting via Trusted Channel is documented in [5] and will be integrated in our prototype. This approach replaces the `TPM_Quote` operation with the usage of sealing and TPM certified keys.

6 Prototype Evaluation and Lessons Learned

6.1 Limitations of our Prototype

One goal of our prototype is to enable independent management of the different virtual domains. In principle, we aimed at allowing each TVD administrator to manage a given TVD independently of all other domains. This would require that a TVD owner can define policies, define images, and start them while being independent of others. In our prototype policies can be defined and machines can be started independently. However, images need to be stored in a central image repository. We implemented this by providing per-domain sub-directories for storing image files of the respective domains.

There are several additional open problems that our implementation has not covered so far. The first is globally unique identifiers. Our current prototype uses a naming authority. In the long run, TVDs should be federated without any mediation of a central authority, making an identification scheme like UUID [25] necessary to guarantee the uniqueness beforehand. Another class of open issues is related to the Integrity Measurement Architecture (IMA) implemented by each physical machine. The current scheme for measuring VM integrity is coarse-grained, because the entire file system is measured. It is a first attempt of measuring the VMs while allowing a persistent storage; however it has a big shortcoming: The measurements do not capture the modifications that occurred on the file system because they are stored on the second read/write virtual disk, which is never measured. Moreover, the VM is measured only prior to being started, and so far there is no support for run-time monitoring yet. In the long run, we will use a finer-grained integrity measurements, e.g., through a virtualization-enhanced version of the IMA proposed in [39] while using integrity-enhanced storage [11, 33] to protect data at rest.

Another part of our architecture that has not been fully implemented is the TVD Masters. Today, they only perform intra-TVD policy distribution. In the long run, they should enable trust brokering and delegation to allow trust establishment between multiple TVDs.

Finally, in this first implementation, all TVD components reside in Domain-0, the Xen-privileged VM. Following the approach of Domain-0 disaggregation[10] [31], the TVD Proxy and VNET components will be moved away from Domain-0 to run in dedicated and isolated VMs.

6.2 Performance Evaluation

We implemented our data center prototype using HP ProLiant BL25p G2 blade servers each fitted with two AMD Opteron processors running at 2 GHz, 8GB system memory and a Gigabit Ethernet card. We now discuss the performance of our prototype. Most of our components implement or support management tasks. They are dedicated to automate the secure set-up of platforms, virtual

[10]The OpenTC consortium is pursuing this approach to reduce the weight of the trust assumptions on Domain-0.

Operation		**System Measured** **Prototype**	**Xen**
Management	Start	3.601s	3.332s
	Stop	2.371s	0.460s

Table 4: Performance Measurements of our Prototype

Throughput	Linux	VLAN Tagging	Xen Bridging	EtherIP
TX (Mbps)	932	883	872	847
RX (Mbps)	932	881	870	851

Table 5: NetPerf Benchmark: Guest VM to Guest VM Throughput.

machines and domains, if possible with minimal performance impact on the running system.

Management In Table 4 compares the boot-up and shut-down delay between virtual machines using the unmodified Xen implementation and our components. Averaged over 235 measurements, our components add some 10 percent to the the original time-to boot. The delay is caused by measuring the configuration, attesting to the VM, transferring and checking configuration measurements, and (in case of success) attaching the VM to the corresponding network. Stopping a virtual machine requires 2.4s instead of 0.5s for the original Xen implementation. Here, the overhead is caused by the fact that the current implementation of the compartment manager polls the VM in rather long intervals to verify a successful shut-down.

Networking We obtained the throughput results using the `NetPerf` network benchmark and the latency results using the `ping` tool. Using the former benchmark, we measured the Tx (outgoing) and Rx (incoming) throughput for traffic from one guest VM to another guest VM on the same physical host. To do so, we ran one instance of the benchmark on one guest VM as a server process and another instance on the second guest VM to do the actual benchmark.

We report the throughput results for different networking schemes in Table 5. The figures show that the throughput results for both VLAN tagging and EtherIP schemes are comparable to that of the standard Xen (bridge) configuration. As expected, VLAN tagging yields the best throughput in a virtualized system that outperforms the standard Xen configuration. Both Xen bridging and VLAN tagging perform better on the **Tx path**. For EtherIP, the major cost in the Tx path is having to allocate a fresh socket buffer (`skb`) and copy the original buffer data into the fresh `skb`. When first allocating a `skb`, the Linux network stack allocates a fixed amount of headroom for the expected headers that will be added to the packet as it goes down the stack. Unfortunately, not enough space is allocated upfront to allow us to fit in the EtherIP header; so, we have to copy the data around, which is very costly.

	Minimum	Average	Maximum	Mean Deviation
Bridged	0.136s	0.180s	0.294s	0.024s
VLAN	0.140s	0.212s	0.357s	0.030s
EtherIP	0.151s	0.246s	0.378s	0.034s

Table 6: Round-trip Times using Ping.

CPU Utilization	Linux	Xen (Dom0+DomU)
Encrypted	42%	45% (42+3%)
Unencrypted	5%	13% (10+3%)

Table 7: CPU Utilization of Virtual Disks at 30MB/s.

In the Rx path, there is no packet-copying overhead for the EtherIP approach; the extra EtherIP header merely has to be removed before the packet is sent to a VM. As compared to VLAN tagging, in which packets are grabbed from the Linux network stack, EtherIP requires that packets are passed to and processed by the host OS IP stack before they are handed over to the EtherIP packet handler of the vSwitch code.

Table 6 shows the round-trip times between two guest VMs on a physical host for the bridged, VLAN, and EtherIP encapsulation cases obtained using the `ping -c 1000 host` command, i.e., 1000 packets sent. The results show that the average round-trip times for VLAN and EtherIP encapsulation are 17.8% and 36.7% higher than that of the standard Xen bridged configuration.

Storage Timing the set-up of storage has been part of the management evaluation. We now evaluate actual run-time performance of virtual disks.

We compared three set-ups using an IBM Thinkpad T60p: Disk access from Domain-0, disk access in a Linux without Xen, and disk access from a guest VM. For each of these three systems we compared encrypted and unencrypted access.

We first measured read/write throughput. A first observation was that in all cases, the disk performance limited our overall performance, i.e., encryption did not result in a performance penalty in a single-VM system (all 6 set-ups provided approx 30MB/s throughput).

As a consequence, we then measured the overall CPU utilization (i.e. for Domain-0 and a guest VM) of the different set-ups (see Table 7). This table points out that encrypting a disk at 30MB/s requires 42% CPU under Linux and 45% under Xen while servicing a guest VM. This shows that the utilization is similar to a plain Linux. The fairly high CPU utilization substantially limits the usage of encryption in a data centers. Fortunately, encryption in a data center can often be replaced by physical security or other measures. The only exceptions are removable media that are moved between locations.

6.3 Lessons Learned

Embedding integrity verification mechanisms in a distributed security architecture creates serious challenges. Conceptually, many of them can be addressed with property-based attestation. However, property-based attestation depends on well-defined and potentially institutionalized processes for validating the behavioral equivalence of different configurations. Certifying that two configurations have identical properties is currently a manual and labor-intensive exercise, which is costly and does not scale beyond single TVD owners or data center administrators.

While the migration of VMs between different physical hosts is well understood, the migration of a complete trust context associated with a VM has proved to be difficult. The latter type of migration requires the migration of not only the virtual networking and storage devices (with associated cryptographic keys, if necessary), but also of a virtual TPM, if present, which will be rooted in different hardware TPMs prior to and after the migration. During the migration process, the integrity of all these components has to be guaranteed while managing the handing-off of device access in a transactional fashion. Note that the importance of securing this transition has been further emphasized by recently published attacks on virtual machines in transit.

Building a security API that is at the same time flexible, usable, and manageable has proved to be more difficult than expected. A key reason for this difficulty is the requirement that the API should be easily adaptable to other hypervisor architectures and to workstation scenarios with GUI interfaces. While addressing each of these requirements separately is feasible, their combination comes with many trade-offs.

Yet another type of trade-off concerns our aim of decomposing the Xen architecture into multiple security services each running in dedicated tiny VMs while reducing the reliance on Domain-0, the privileged management VM. While such decomposition is advantageous from a security perspective, it tends to reduce flexibility. The availability of a full-fledged Linux management VM with access to all subsystems enables easy extensibility and rapid prototyping (scripting, adding devices, firewalls, VPNs etc), and also corresponds to the expectations of many Xen users. In general, however, considerations of usability tend to favor design decisions that are sub-optimal from a strict security perspective.

A final lesson learned was that measuring performance of virtual systems is not straightforward. We first used `iostat` to retrieve CPU usage data from `/proc/stat` under Linux and in Domain-0. This wrongly indicated that Xen needs half the CPU as compared to Linux. We then used the Xen tool `xentop` that gathers the data via hyper-calls to the hypervisor. Since this represents the time given to each VM (Domain-0, guest VM) by the hypervisor, the resulting data was no longer distorted by the virtual notion of time given to the VMs in Xen.

7 Conclusion

Securing the access to data on persistent media and during transfer over the network is a serious problem in distributed virtual data center and cloud computing scenarios. We described a framework based on TVDs and Trusted Computing for secure network and storage virtualization that includes mechanisms for verifying the integrity of the hypervisor, VMs, and security policy enforcement points. The concept of TVDs is rigid enough to allow consistent policy enforcement across a group of virtual domain elements, while being flexible enough to support policy-controlled interactions between different TVDs. TVD policies and configurations are 'backward-compatible' in supporting options that could be taken for granted in non-virtualized data centers. For example, cohosting of specific customer services with those of other data center customers on the same physical platform could be inhibited if so desired. By incorporating hardware-based Trusted Computing technology, our framework allows the creation of policy domains with attestable trust properties for each of the virtual domain nodes.

The inclusion of integrity measurement and management mechanisms as part of the physical platform's TCBs provides both data center customers and administrators with a much needed view of the elements (hypervisors, VMs, etc.) that are part of their virtual infrastructure as well as information on the configurations of those elements. Our framework can be used to obtain information about the elements on a 'need-to-know' basis without having to introduce all-powerful roles of administrators with access to every aspect of a platform.

Our performance evaluation shows that the overhead produced by changes and extensions to the original implementation is modest. Automated set-up adds a short delay to boot-time, VLAN tagging and IP tunneling lead to an increase in packet latency of 5 resp. 10 percent, while VLAN tagging provides slightly improved network throughput. Although the performance overhead of encrypting a single disk is non-negligible, the observed impact on the disk transfer rate tends to be small, since the predominant limiting factor is the capacity of the disk I/O channel. However, the performance of disk encryption in Xen is comparable to disk encryption under Linux.

Acknowledgements

We would like to thank our anonymous reviewers for providing great and detailed feedback that helped us to substantially improve the article. We would like to thank in particular Cataldo Basile from Politecnico di Torino, Italy, for valuable input on the policy framework, Michael Steiner from IBM's Watson Research Center for feedback on policy refinement, and Marcel Winandy of Ruhr-Universität Bochum for feedback on the TVD implementation.

This article is based on input from many members of the OpenTC project consortium. This work has been partially funded by the European Commission as part of the OpenTC project [32] (ref. nr. 027635). It is the work of the authors alone and may not reflect the opinion of the entire project.

References

[1] Aufs – Another Unionfs. `http://aufs.sourceforge.net/`.

[2] dm-crypt: a device-mapper crypto target. `http://www.saout.de/misc/dm-crypt/`.

[3] TrustedGRUB. `http://sourceforge.net/projects/trustedgrub`.

[4] Unionfs: A Stackable Unification File System. `http://www.am-utils.org/project-unionfs.html`.

[5] Frederik Armknecht, Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, Gianluca Ramunno, and Davide Vernizzi. An efficient implementation of Trusted Channels based on Openssl. In *STC '08: Proceedings of the 3rd ACM workshop on Scalable Trusted Computing*, pages 41–50, New York, NY, USA, 2008. ACM Press.

[6] P. T. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP-2003)*, pages 164–177, October 2003.

[7] Stefan Berger, Ramón Cáceres, Dimitrios Pendarakis, Reiner Sailer, Enriquillo Valdez, Ronald Perez, Wayne Schildhauer, and Deepa Srinivasan. TVDc: managing security in the trusted virtual datacenter. *SIGOPS Operating Systems Review*, 42(1):40–47, 2008.

[8] A. Bussani, J. L. Griffin, B. Jansen, K. Julisch, G. Karjoth, H. Maruyama, M. Nakamura, R. Perez, M. Schunter, A. Tanner, L. van Doorn, E. V. Herreweghen, M. Waidner, and S. Yoshihama. Trusted Virtual Domains: Secure foundation for business and IT services. Research Report RC 23792, IBM Research, November 2005.

[9] A. Cappadonia C. Basile and A. Lioy. Algebraic models to detect and solve policy conflicts. In I. Kotenko V. Gorodetsky and V.A. Skormin, editors, *MMM-ACNS 2007*, volume 1 of *CCIS*, pages 242–247. Springer-Verlag, 2007.

[10] Serdar Cabuk, Chris Dalton, HariGovind V. Ramasamy, and Matthias Schunter. Towards automated provisioning of secure virtualized networks. In *Proc. 14th ACM Conference on Computer and Communications Security (CCS-2007)*, pages 235–245, October 2007.

[11] Dwaine E. Clarke, G. Edward Suh, Blaise Gassend, Ajay Sudan, Marten van Dijk, and Srinivas Devadas. Towards constant bandwidth overhead integrity checking of untrusted data. In *IEEE Symposium on Security and Privacy*, pages 139–153. IEEE Computer Society, 2005.

[12] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. Internet Engineering Task Force: `http://www.ietf.org/rfc/rfc4346.txt`, April 2006. Network Working Group RFC 4346.

[13] N. Dunlop, J. Indulska, and K. A. Raymond. A formal specification of conflicts in dynamic policy-based management systems. DSTC Technical Report, CRC for Enterprise Distributed Systems, University of Queensland, Australia, August 2001.

[14] H. Hamed E. Al-Shaer, R. Boutaba and M. Hasan. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications, IEEE*, 23(10):2069–2084, Oct. 2005.

[15] Zhi Fu, Shyhtsun Felix Wu, He Huang, Kung Loh, Fengmin Gong, Ilia Baldine, and Chong Xu. IPSec/VPN security policy: Correctness, conflict detection, and resolution. In *POLICY*, pages 39–56, 2001.

[16] Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N. Asokan. Beyond Secure Channels. In *STC '07: Proceedings of the second ACM workshop on Scalable Trusted Computing*, pages 30–40. ACM Press, 2007.

[17] John Griffin, Trent Jaeger, Ron Perez, Rainer Sailer, Leendert Van Doorn, and Ramon Caceres. Trusted Virtual Domains: Toward secure distributed services. In *Proc. 1st Workshop on Hot Topics in System Dependability (Hotdep-2005)*, Yokohama, Japan, June 2005. IEEE Press.

[18] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic Remote Attestation - virtual machine directed approach to Trusted Computing. In *USENIX Virtual Machine Research and Technology Symposium*, pages 29–41, 2004. also Technical Report No. 03-20, School of Information and Computer Science, University of California, Irvine.

[19] IEEE. Standards for local and metropolitan area networks: Virtual bridged local area networks. Technical Report ISBN 0-7381-3662-X, IEEE, 1998.

[20] IETF. EtherIP: Tunneling Ethernet Frames in IP Datagrams, 2002. RFC 3378.

[21] Bernhard Jansen, Harigovind Ramasamy, and Matthias Schunter. Policy enforcement and compliance proofs for Xen virtual machines. In *4th International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE-2008)*, pages 101–110. ACM Press, New York, 2008.

[22] Y. Katsuno, M. Kudo, Y. Watanabe, S. Yoshihama, R. Perez, R. Sailer, and L. van Doorn. Towards Multi–Layer Trusted Virtual Domains. In *The Second Workshop on Advances in Trusted Computing (WATC '06 Fall)*, Tokyo, Japan, November 2006. From http://www.trl.ibm.com/projects/watc/program.htm.

[23] S. Kent and K. Seo. Security Architecture for the Internet Protocol. Internet Engineering Task Force: `http://www.ietf.org/rfc/rfc4301.txt`, December 2005. Network Working Group RFC 4346. Obsoletes: RCF2401.

[24] Ulrich Kühn, Marcel Selhorst, and Christian Stüble. Realizing property-based attestation and sealing with commonly available hard- and software. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 50–57, New York, NY, USA, 2007. ACM.

[25] P. Leach, M Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. Internet Engineering Task Force RFC 4122, July 2005.

[26] Emil Lupu and Morris Sloman. Conflicts in policy-based distributed system management. *IEEE Transaction on Software Engineering*, 25(6):852–869, November 1999.

[27] Rich MacDonald, Sean Smith, John Marchesini, and Omen Wild. Bear: An open-source virtual secure coprocessor based on TCPA. Technical Report TR2003-471, Department of Computer Science, Dartmouth College, Hanover, NH, USA, 2003.

[28] John Marchesini, Sean W. Smith, Omen Wild, and Rich MacDonald. Experimenting with TCPA/TCG hardware, or: How I learned to stop worrying and love the bear. Technical Report TR2003-476, Department of Computer Science, Dartmouth College, 2003.

[29] John Marchesini, Sean W. Smith, Omen Wild, Josh Stabiner, and Alex Barsamian. Open-source applications of TCPA hardware. In *20th Annual Computer Security Applications Conference*, pages 294–303, Washington, DC, USA, December 2004. ACM, IEEE Computer Society.

[30] Craig Mundie, Pierre de Vries, Peter Haynes, and Matt Corwine. Trustworthy computing. White paper, Microsoft Corporation, October 2002.

[31] D. G. Murray, G. Milos, and S. Hand. Improving Xen security through disaggregation. In *4th International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE-2008)*, pages 151–160. ACM Press, New York, 2008.

[32] Open Trusted Computing (OpenTC) Project. The OpenTC Project Homepage, 2008. `http://www.opentc.net/`.

[33] Alina Oprea, Michael K. Reiter, and Ke Yang. Space-efficient block storage integrity. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2005)*, San Diego, CA, February 2005. Internet Society.

[34] Jonathan Poritz, Matthias Schunter, Els Van Herreweghen, and Michael Waidner. Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical Report RZ 3548, IBM Research, May 2004.

[35] D. Pendarakis R. Yavatkar and R. Guerin. A framework for policy-based admission control. RFC 2753, January 2000.

[36] A-R. Sadeghi and C. Stüble. Property-based Attestation for Computing Platforms: Caring about Properties, not Mechanisms. In *Proc. 2004 Workshop on New Security Paradigms (NSPW-2004)*, pages 67–77, New York, NY, USA, 2005. ACM Press.

[37] David Safford. Clarifying misinformation on TCPA. White paper, IBM Research, October 2002. Available at `http://www.research.ibm.com/gsal/tcpa/tcpa_rebuttal.pdf`. See also [38] and `http://www.research.ibm.com/gsal/tcpa/`.

[38] David Safford. The need for TCPA. White paper, IBM Research, October 2002. Available at `http://www.research.ibm.com/gsal/tcpa/why_tcpa.pdf`. See also [37] and `http://www.research.ibm.com/gsal/tcpa/`.

[39] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[40] Trusted Computing Group (TCG). `www.trustedcomputinggroup.org`.

[41] Trusted Computing Group (TCG). TCG TPM specification version 1.2 revision 103. `https://www.trustedcomputinggroup.org/specs/TPM/`, July 2007. See also [43] and `http://www.trustedcomputing.org/`.

[42] Trusted Computing Group (TCG). About us. `https://www.trustedcomputinggroup.org/about/`, May 2008.

[43] Trusted Computing Platform Alliance (TCPA). Main specification version 1.1b. `https://www.trustedcomputinggroup.org/specs/TPM/`, February 2002. See also [41] and `http://www.trustedcomputing.org/`.

[44] Andrea Westerinen. Terminology for policy-based management. RFC 3198, November 2001.

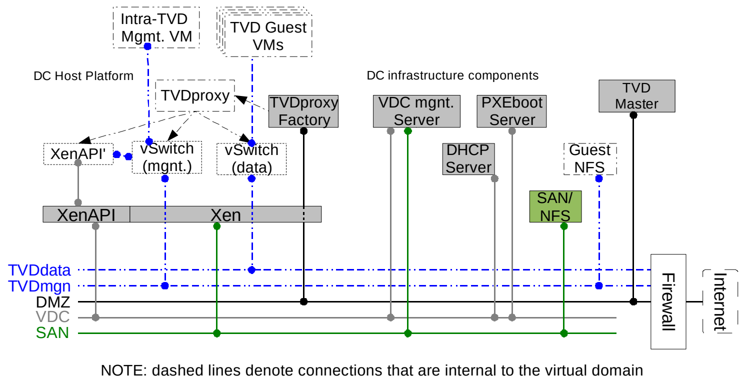

Figure 8: Layout of our prototype virtual data center.