RC23792 (W0511-075) November 9, 2005
Computer Science

IBM Research Report

Trusted Virtual Domains: Secure Foundations for Business and IT Services

Anthony Bussani[2], John Linwood Griffin[1], Bernhard Jansen[2], Klaus Julisch[2], Güenter Karjoth[2], Hiroshi Maruyama[3], Megumi Nakamura[3], Ronald Perez[1], Matthias Schunter[2], Axel Tanner[2], Leendert van Doorn[1], Els A. Van Herreweghen[2], Michael Waidner[2], Sachiko Yoshihama[3]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]IBM Research
Zurich Research Laboratory
8803 Rüschlikon, Switzerland

[3]IBM Research
Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Executive Summary

Trusted Virtual Domains (TVDs) represent a new model for achieving IT and business security. TVDs address critical heterogeneity and complexity issues in existing models, they provide quantifiable security and operational management for business and IT services, and they simplify overall containment and trust management in large distributed systems.

The key innovation in TVDs is a focus on overall security goals required within *service domains*—collections of complete systems that work together to provide a service—as opposed to point hardware and software solutions. This emphasis on satisfying service-oriented goals is a step toward enabling the flexible deployment of secure services in on demand environments.

Within a TVD, high-level security and operational policy statements are systematically mapped into the configuration of the individual hardware and software components that together perform a service. For example, a TVD for a payroll-processing service would transform business-level policy statements such as "Employees' personal information in HR records must only be disclosed to authorized parties" into platform-specific directives for information flow and access control. These directives are then used to configure the protected execution environments that host the HR information service.

The TVD model represents a departure from the design of many conventional secure operational models. For example, TVDs are designed to provide an explicit and autonomously measurable quantification of whether the overall security goals are achieved, prior to (or during) the processing of a service. The application developer is relieved from the burden of implementing and verifying security-related functions for service processing, such as the creation of protected communication channels, as such functions are provided by the TVD infrastructure. Also, the specification of security goals in TVDs proceeds according to the requirements of the application or service to be run, instead of being individually specified on a per-user or per-system basis.

This white paper discusses the high-level design of TVDs from the point of view of our customers. The paper also describes two case studies for TVD deployment.

Contacts

For more information on TVDs, please contact any of:

Charles Palmer, Watson Research Center (ccpalmer@us.ibm.com)
Hiroshi Maruyama, Tokyo Research Lab (maruyama@jp.ibm.com)
Michael Waidner, Zurich Research Lab (wmi@zurich.ibm.com)

Table of Contents

1. Background and motivation for Trusted Virtual Domains

1.1 Inadequate frameworks for developing secure on demand services

Current approaches to security management are poorly suited to transition to on demand and eUtility business models, both at IBM and across the industry. This limitation constitutes a clear and present obstacle to sustaining IBM's leadership in service-oriented computing, and if not addressed will negatively impact the uniqueness and value of the solutions IBM offers to its clients.

As IBM's customers transition to the integration of on demand, service-oriented computing solutions, IBM is focusing on enabling our customers to seamlessly coordinate and securely manage ever-larger sets of distributed, heterogeneous, and highly dynamic computer systems. Current approaches to security management require stakeholders (security officers, application developers, users, and administrators) to manually combine separately deployed techniques such as authentication services, firewalls, virtual private networks, antivirus software, and remote monitoring agents. This paradigm will not adapt to a larger-scale, on demand environment, because of at least three key limitations:

"Securing the physical perimeter" is losing its effectiveness as a technique. IT environments are becoming increasingly heterogeneous and distributed, rendering traditional security measures at the physical or network boundary ineffective. One computer system cannot rely on the safe behavior of another system simply because both reside in the same machine room. Most communication protocols today use encryption techniques, which reduces opportunities for traffic monitoring or content filtering at the boundary.

There is no end-to-end focus on tying together the individual components used to secure systems. The increasing complexity of IT systems makes it difficult to keep them in a known good configuration. The trustworthiness of applications relies not only on the application itself but also on the integrity of its execution environment, including the operating system and other components. Legacy operating systems and execution environments do not provide useful mechanisms for verifying integrity or establishing trust in the overall system.

Security verification already relies too much on user and administrator involvement. Implementing business-level security requirements with the appropriate IT configuration in a large, heterogeneous environment is very labor-intensive and error-prone. Whenever systems are reconfigured—be it because new applications are deployed, hardware is upgraded, or the user base changes—the security properties and architectures are affected, and operational parameters must often be readapted manually.

1.2 TVDs: A composable infrastructure for scalable secure services

Trusted Virtual Domains (TVDs) address today's challenges toward managing the secure configuration, deployment, and operation of distributed, scalable, on demand services. The primary security goals of TVDs are to provide an operating environment for such services that has verified containment and trust properties, and to simplify the role of humans in specifying and validating these properties.

The three limitations described in the previous section correspond to general problems in current systems involving inadequate containment, insufficient trust, and unbounded complexity. TVDs are designed to directly address these three problems:

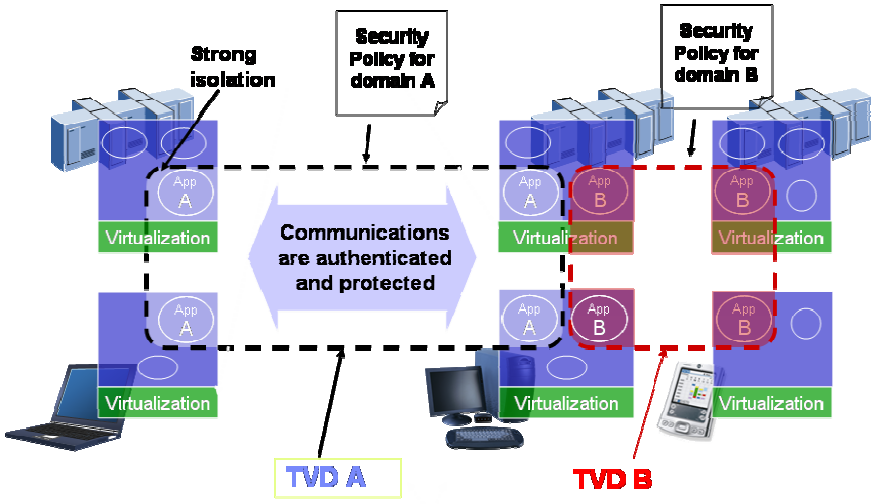

Figure 1. Distributed services are being performed in each TVD. Each service is divided into four processing members, each of which runs in isolated virtual execution environments across four heterogeneous machines.

Containment. TVDs use virtualization and overlay technologies to form a protection layer around each of the computing entities used to perform a service, regardless of the physical machine or network topology configuration of those entities. The resulting workload execution environments are contained in several desirable ways: Their internal execution is isolated from any unintentional or malicious side-effects of external applications. External applications are protected from damage due to a misbehaving internal application. Moreover, communications among internal components are automatically protected from interception or interference by an external entity.

Trust. Trust is the quality leading to the belief that a remote system will behave as expected. An inherent property in TVDs is the codification of trust among computing entities that potentially are composed of heterogeneous hardware and software components, are geographically and physically widely separated, and are not centrally administered or controlled. By leveraging artifacts of the traditional security infrastructure (such as digital signatures, certificates, and assurance statements) and building upon emerging trusted computing technologies, TVDs convey trust evaluations and guarantees for each entity within the domain.

Simplification. In a TVD, security policy statements are initially specified at an abstract level on the basis of the service to be performed. These statements are methodically decomposed, and uniformly enforced and verified across the layers of decentralized hardware and software resources in the TVD. Measurable properties are drawn together and analyzed as part of an explicit global security measurement. The result is a security perspective of business and IT services that is coherent and manageable—and greatly simplified for the user and administrator—that addresses the scalability and composability requirements of the on demand operating environment.

Now is the time to develop and deploy an integrated end-to-end approach toward foundational security in on demand services. Such an approach must be simple in the face of hidden complexity—it must enable human users and administrators to reason about security decisions in understandable terms, while simultaneously being powerful enough to coordinate many machines across a vast physical and geographic homogeneity. TVDs fulfill these requirements.

2. Design overview

2.1 Design objectives: Secure processing and communications for services

Conceptually, a TVD is composed of two elements: a set of distributed, trustable virtual processing environments and a communications channel to connect and provide an isolation boundary around the environments. The TVD infrastructure ensures that the configuration of both the environments and the channel match the secure operational policy specified for the services running inside the TVD.

The design of TVDs is based on the concept of security domains. A security domain is a computing environment that uniformly enforces a secure operational policy across all its members. Such a policy includes statements involving such concepts as authentication, authorization, access control, and assurance, as discussed in Section 2.2. In the context of TVDs, this design simplifies the user's and administrator's experience: each can treat a TVD as if it were composed of an isolated network within which all the nodes are mutually trustable (to the extent specified by the policy).

TVDs are constructs built upon isolation technologies such as hypervisors or software-based virtual machines. This means that the perceived topology of hardware and software components visible inside a TVD may not necessarily reflect the topology of the physical machines and networks that compose that TVD.

Figure 1 demonstrates a sample TVD configuration. Here, each service is composed of four processing members distributed across four machines. Each member is strongly isolated from other non-member processes residing on the same physical system. The TVD members communicate with each other across an abstract channel that provides for the authentication and authorization of all members, and protects the security and integrity of the communications among the members.

From an application programmer's perspective, a TVD can be considered to be a "virtual intranet." In other words, from this perspective, a TVD is an environment where each processing component is well maintained and trusted, where intradomain communications are protected by default, and where extradomain communications are regulated by firewalls based on the domain's policy. If desired, the application may also be made aware of more complex details of the TVD membership—for example, the application might transmit data at a certain level of integrity or fidelity depending on its knowledge of whether its communications partner is a member of the application's TVD.

2.2 Secure operational policy for TVDs

Secure operational policy is stated at several levels, depending on the requirements of the services running inside the TVD. Policy statements may involve functional requirements, such as containment and access control; quality requirements, such as the minimum acceptable trust and assurance levels to be attained; and implementation/mechanism mappings, describing specific hardware and software configurations that meet the functional and quality requirements. The main constraint on policy statements is that each statement must be verifiable.

Policy statements for TVDs are not limited to access control statements, in contrast to the policy statements made by many traditional security models. Instead, several types of policy statements may be associated with a TVD:

> *Functional requirements* specify limitations on what can (must) and cannot (must not) happen. An example of a security functional requirement is "Only entities authenticated as managers are authorized to access this data"; an example of an operational functional requirement is "Each execution environment must have access to at least 200 GB of local protected disk storage".
>
> *Quality requirements* specify quantitative metrics for adherence to process standards, such as "Each participant must be certified to be ITCS 104 compliant," or the level of shared trust among TVD members, such as "The integrity of each platform must be measured by TPM hardware".
>
> *Implementation/mechanism mappings* describe hardware and software configurations that are known to be acceptable for mapping functional and especially quality requirements onto real systems. "The x86 Open Hypervisor version 9.87 provides a platform that is currently certified to be ITCS 104 compliant."

The secure operational policy for a TVD is enforced for each of its members. The policy statements may define functionality either to be verifiably implemented by the individual members or to be enforced by the TVD infrastructure itself. Several policy statements that are typically expected to be applied to TVDs are shown in Table 1.

Table 1. Example policy statements for TVDs.

Statement Type	Example
Ownership	The EE is owned or created by a given party.
Information flow	Confidential information must never leave the domain.
Access control	Confidential information must only be accessed by users with the appropriate authorization.
Channel protection	All communications in the channel must be authenticated and encrypted using strong cryptographic techniques.
User authentication	All users of the domain must be properly authenticated.
Monitoring and auditing	All security-critical domain operations must be recorded in a secure audit log.
Platform assurance	The OS used must have at least EAL-4 certification, and the software integrity must be verified by TCG-based attestation.
Third-party trust	Only a specific set of trusted third parties must be used in certain roles (such as certificate authorities).
Operational policy	The platform must be administered by a BS-7799 certified operator.

To maintain the integrity of the TVD, each candidate member must demonstrate adherence to the secure operational policy when it joins the TVD. (Note that this can be an active or a passive operation: in one extreme, a member may be unaware that it joined a TVD—a condition we term *implicit membership*—as long as its affiliation with the TVD can be externally evaluated to meet the TVD's policy.) To ensure that adherence is computable, we note that each policy statement must be *verifiable*, meaning that there must exist a mechanism for other TVD members to remotely verify the adherence of the candidate member to the policy statements with reasonable accuracy.

2.3 Architectural components and interconnections

A TVD contains architectural components that support policy enforcement, TVD management, and TVD operation. For enforcement, each hardware node hosts one or more containers, where each container supports one or more execution environments (EEs). Logistically, the run-time management of a TVD is handled by either third-party domain controllers or by software running in individual EEs inside the TVD.

Conceptually, a TVD is composed of a set of trusted execution environments (EEs) that communicate via an abstract communications channel. Specifically, a TVD contains one or more physical platforms called *nodes*. Each node hosts one or more software *containers* that coordinate the overall platform security and the secure operation of one or more *EEs*. The relationship among nodes, containers, EEs, and domain controllers (discussed below) is illustrated in Figure 2.

Containers provide the trust basis for the EEs by using hardware or software techniques to establish a trust basis for the node itself. Verification of policy statements may stem from attestations or assertions enforced by the containers, by the individual EEs, or both. Containers also manage the creation and destruction of EEs, resource provisioning for EEs (such as computational time and I/O metering), and auditing of EE operation.

The *channel* is an abstract notion for exchanging information among the components in a TVD. This includes data traffic for the service and administrative traffic for the TVD itself. The channel is not tied to any particular mechanism—for example, channels may be implemented using IPSec-enabled network connections or may be implemented using shared secure secondary storage. For containment, containers may be tasked with mediating resource sharing and communication among EEs, or software inside the EEs can be responsible for channel creation and validation.

Life-cycle management operations of TVD include such activities as attestation, provisioning, monitoring, policy distribution and administration, and member registration. These operations can either be serviced by one or more third-party *domain controllers* that are not otherwise part of the TVD, or by one or more EEs that take on the specific additional role of a domain controller.

2.4 Interfaces to the core architectural components

Containers and EEs are the building blocks for TVDs. Containers manage the life cycle of the EEs that compose the TVD, while EEs provide the trusted execution environment for the services being performed. Multi-implementation-compatible APIs are available for container management by a domain controller, EE management by its container or a domain controller, communications between an EE and another entity, and the code executing inside the EE.

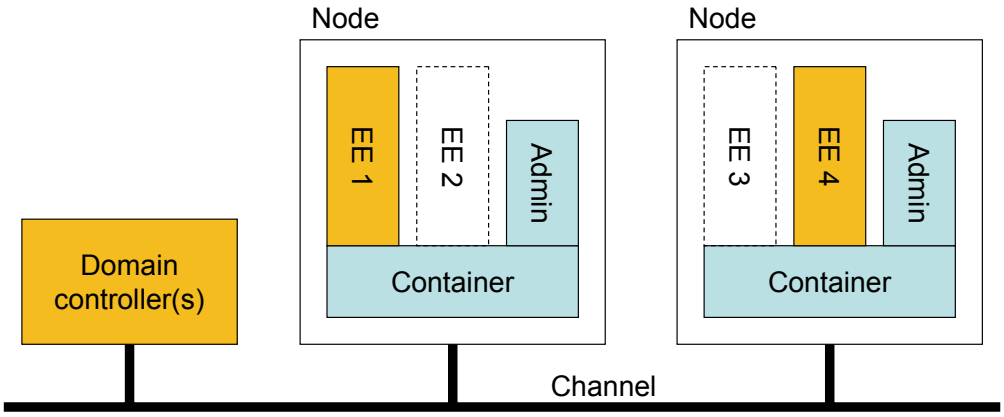

Figure 2. Architectural components of a TVD. In this example, a TVD has been formed using EE 1 and EE 4. Note that although two other EEs (2 and 3) coexist on the same container, they are strictly isolated from EEs 1 and 4 and from the channel shown.

EEs are similar to ordinary virtual execution environments. EEs include an application-independent layer that controls the program being executed or interpreted by the virtual environment. This layer, which we term the EE Monitor, may include such components as a secured OS and/or middleware. The specific architecture is illustrated in Figure 3. These are trusted execution environments in that the EE Monitor or the EE's container are specifically designed to address and verify trust issues as part of its adherence to the secure operational policy.

An EE's life-cycle is driven by its container, as discussed in Section 2.3. To accomplish this, APIs are available for EE management by its container, and for container management by a domain controller or an alternate entity. The EE Monitor also provides an API for verifying the integrity of the node, up to the level of the EE and its contents. By calling into this API, other TVD elements can determine the integrity of the platform and confirm any relevant policy statements.

Information flow management is an important aspect of the TVD concept. Using TVDs, the application developer is relieved from the burden of implementing authentication and authorization mechanisms to verify the access of remote parties during a communication, and is also relieved from worrying about the security and integrity of communications across the channel, because these details are taken care of by the TVD infrastructure. To achieve this, all communication by the application must go through well-defined APIs. Communication between peer members in a TVD is generally unrestricted, as all members are verified to adhere to the same policy. Communication between a TVD member and a non-member is generally disallowed but may be permitted by the appropriate policy statements.

TVD-agnostic applications. A TVD container may control the EE life-cycle and enforce EE-level policies by using its enforcement mechanism. This allows existing TVD-agnostic applications to gain many of the benefits of running in a TVD environment. Moreover, existing applications may have been initially designed decades ago for a particular environment—such as a single disconnected system with a small set of expert users—but over time these applications are migrated into unforeseen new environments such as onto Internet-visible servers. TVDs can retain the visibility of such a new environment while simultaneously returning the applications to the "secure enclosure" for which they were designed.

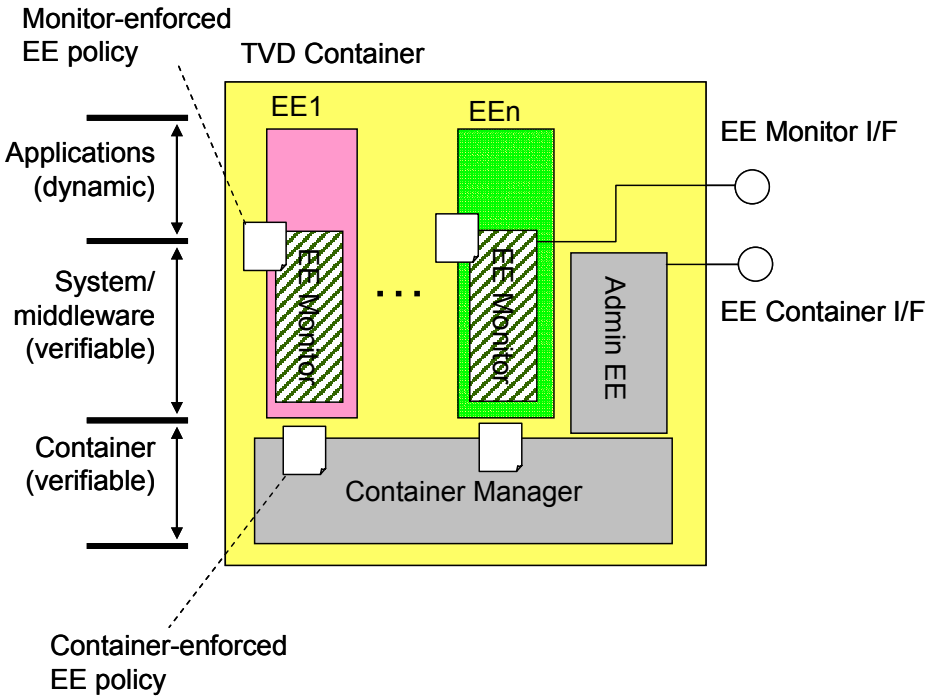

Figure 3. Internal structure of a container, and interfaces to the EE.

TVD-aware applications. Applications may be designed to use one of several predefined APIs that utilize TVD-level context in application-level logic. APIs for TVDs are defined as abstract, platform-independent interfaces, with each configuration-dependent implementation providing mappings between the abstract API and the platform-specific interface. A TVD API would include interfaces for creating, destroying, and provisioning EEs; for EEs to join or depart a TVD; for testing other containers and TVDs for their compliance to a TVD's policy; and for accessing TVD-enabled resources such as secure channels or secure persistent storage. For example, a TVD-aware on-line shop application may limit purchase orders from non-TVD-member EEs to a maximum of $100 while allowing TVD-member EEs unlimited transactions.

In addition, virtualization makes it possible to take a snapshot of a runtime environment. This enables code mobility and recovery; i.e., the run-time state of an EE can be recorded and then migrated to other physical nodes, e.g., to simplify a long-lasting transactional process. In the case of a fatal event, an EE may be rolled-back to an earlier state that is known to be trusted.

2.5 Mechanism independence of the TVD model

The TVD abstraction is designed to apply across a heterogeneous set of environments, including the presence or absence of trusted hardware, hypervisors, security-enhanced operating systems, OSGi middleware, and native-execution or interpreted processing models. Depending on the environment, the functionality of the TVD will remain constant, but the levels of containment and trust offered by the TVD will change.

We envision that the TVD model can accommodate a diversity of IT systems while supporting a unified secure operational policy among them. Our current focus is on policy statements relating to the security properties of *containment* and *trust*. The levels to which these properties are achievable and verifiable depend on the hardware and software mechanisms available to a particular TVD.

Containment properties include those that mediate information flow and communication, and those that isolate members from damage. The strongest isolation is provided by placing individual execution environments on disparate nodes. Hypervisor technologies such as the Open Hypervisor provide very strong isolation among execution environments on the same node, while secured OSes and middleware technologies such as J2EE and OSGi can provide isolation among applications running in the same EE.

Trust properties involve verifying an EE's adherence to the policy statements. One example of a verification technology is Trusted Computing Group (TCG)-based attestation. TCG attestation—both binary and property attestation—involves hardware-based verification of software integrity. Verification could also be based on an implicit trust of the platform's middleware, or could involve a combination of TCG-based attestation and such other integrity verification mechanisms as the Tripwire change auditing tool or the IBM Workstation Security Tool (WST).

Table 2 illustrates several possible combinations of mechanisms for implementing TVDs. The hypervisor-based isolation coupled with TCG-based verification in Cases 1 and 2 provides very strong levels of containment and trust for TVDs. Java-based isolation is weak compared with other alternatives, but its use allows rapid TVD deployment on current IT infrastructures and provides a reasonable migration path to stronger mechanisms. Case 5 represents today's implementation of the TVD concept using physical separation, whereas Case 6 illustrates mechanisms that could potentially be further enabled or well suited to TVD environments—e.g., purpose-specific component-based applications which include only the operating system support that is specifically required; component-based hypervisors that provide application-specific isolation; and *streams* which are perhaps unidirectional self-protecting communication entities.

Table 2. Mechanism alternatives for implementing TVDs.

	Isolation	Verification	EE Monitor	Applications	Channels
Case 1	Hypervisor	TCG attestation	COTS OS	OS native apps	VLAN + IPSec
Case 2	Hypervisor	TCG attestation	OS + OSGi	OSGi bundles	VLAN + WSS
Case 3	SELinux	TCG attestation	SELinux	Linux native apps	IPSec
Case 4	J2EE	Implicit trust	J2EE	J2EE	WSS
Case 5	Physical	TCG attestation	COTS OS	OS native apps	VLAN + IPSec
Case 6	Component-based hypervisor	TCG attestation	Library OS	Component-based applications	Streams

3. Case study #1: System management in strategic outsourcing

The goal of this case study is to allow robust monitoring and management of client workstations in strategic outsourcing. It aims at damage control and to lower the management costs in large customer installations of workplace PCs.

This case study implements Case 1 in Table 2, where a secure hypervisor is used to provide isolation and TCG-based attestation is used for verification. For simplicity, we do not describe details of the servers and other infrastructure components.

Consider a large installation of workplace PCs and notebooks, which are assigned to employees. Business requirements often dictate that such computers be used for a multiplicity of tasks, including Internet communication, displaying email, and sharing word-processing documents. This presents several well-known business risks, including the running of unlicensed or unauthorized software on the system (including viruses, worms, and spyware) or even the improper configuration of the system. TVDs mitigate these risks and reduce the human-visible management effort and costs in this environment.

Figure 4 depicts a scenario in which IBM Global Services manages the PCs of two outsourcing customers. The PCs are managed using two consoles that are accessible from the IBM intranet via the services backbone (SBB). GSNI/ITS104 policies separate the two customer environments into two logical zones, where each zone is defined and encapsulated by a TVD. In this scenario, the yellow zone is an IBM-controlled management zone that is dedicated to servicing a customer. The red zone is customer controlled and hosts the customer's workstations and servers.

As shown in Figure 4, portions of the individual yellow and red zones coexist on servers in a virtualized "infrastructure consolidation" environment. The TVD infrastructure ensures that isolation requirements are preserved, both between customers and between the customers and the IBM internal network. For example, there are four EEs running on the Management/Gateway machine of the operator. A hypervisor and a gateway ensure strong isolation between this customer and IBM. The VLAN topology and the flow policies guarantee that the management console can only connect to other machines within this TVD. The applications running in these TVDs enjoy all the usual management benefits and cost reductions of virtualization, including resource provisioning and migration of software components among different machines in different topological locations.

The use of TVD also improves damage control and security functions. A main feature of the TVD concept is that the Domain Controller continuously keeps track of the security and trust state of all domain members. Therefore the Domain Controller knows which applications are running on the domain members. If unauthorized software (not licensed in the best case or a virus/spy-ware in the worst case) is detected, the Domain Controller or the management EE can react with predefined actions. The actions can be the application of special firewall rules to the host, or, more drastically, the hibernation of the infected client operating system for later inspection. The advantage over today's virus scanner is that the TPM measurement system, which is part of the TVD concept, cannot be switched off by any malicious code before the code is registered, and that it works explicitly and not implicitly. In abstract, only allowed code will not raise an alarm (and not trigger the execution of an action) on the Domain Controller.

Example: Software integrity policies

Each TVD may have an integrity policy that defines which integrity properties are required for this TVD. For example, the yellow TVD will only be allowed to run IBM-approved software. For the red (customer) TVDs, each customer can define which software is allowed to be executed. By using two virtual machines

on each client, the management machine (yellow) can implement automated management and provisioning, which normally requires hardware access (e.g. reboot of a frozen machine can be done remotely at any time). As the red zone is maintained by the client and cannot be controlled as tightly, the integrity controls may be relaxed as compared with those of the yellow zone, where only authorized software is allowed to run.

Example: Information flow control policies

TVDs may also have a flow control policy that describes communication to and from the EEs in the TVD. The policies for the TVDs in this scenario are defined using a role-based model in which each EE is assigned a specific role. The EE roles are listed in Table 3. The TVD policy defines both which roles are allowed to exist in the TVD and which flow control rules are assigned to each role.

The flow control matrix for this scenario is shown in Table 3. In the common case where an EE attempts to communicate with another EE in the same TVD, no information flow restrictions are imposed. Two policy examples for cross-TVD communication by EEs are shown: *Policy Rule—P*, where extradomain point-to-point connections are allowed to specific destinations (e.g., those identified as taking on a gateway role), and *Policy Rule—G*, where extradomain information flows are only permitted when they first pass through a gateway.

The information flow control policies described in this example are relatively simple IP-level connectivity rules for network firewalls. Depending on the strength of the TVD or application requirements, additional policies could be realized by running detailed monitoring, data tracking, and filtering software inside the gateway EEs themselves.

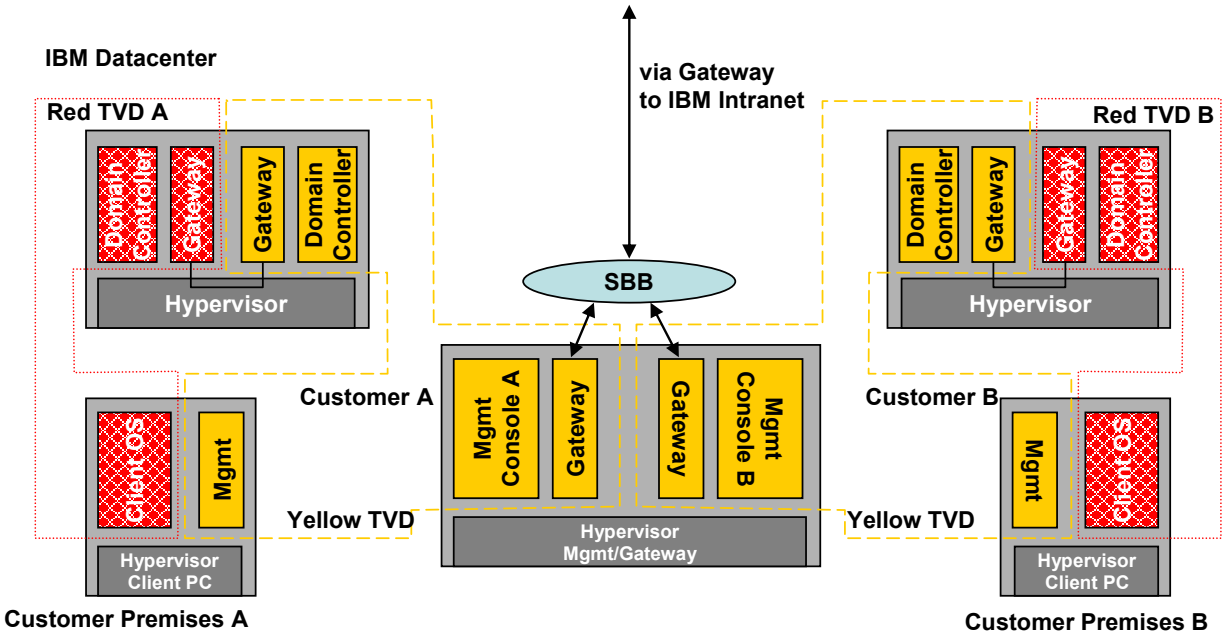

Figure 4. System architecture for case study #1.

Table 3. Information flow control policies.

TVD membership and role for originating EE	Communications with yellow-zone EEs	Communications with red-zone EEs
Yellow: Domain Controller	Unrestricted	No flow
Yellow: Gateway	Unrestricted	Policy Rule—P
Yellow: Management Console	Unrestricted	Policy Rule—G
Red: Domain Controller	Policy Rule—G	Unrestricted
Red: Gateway	Policy Rule—P	Unrestricted
Red: Client OS	No flow	Unrestricted

4. Case study #2: Software as a service

The second case study is about a hypothetical retail company called tMart that wants to use an outsourced business process service for their employees' payroll. The service company, Paychecks.com, runs the service. A TVD with a common security policy is formed across the two organizations' IT infrastructure.

This case study implements Case 2 in Table 2, which is similar to the previous case study except that OSGi bundles are used in lieu of native OS applications.

In this scenario, each employee of tMart can enter their sign in and out records through a desktop PC, a special punching machine, or a mobile PC (if the employee is on the road). Also there are a number of contractors who use PCs in their own office to enter the records. A store manager at tMart is to approve the entered data, and Paychecks.com calculates payrolls and issue checks. This data is reflected to the accounting system of tMart and also to an affiliated bank. In addition, Paychecks.com needs a few other connections to tMart's backend systems, such as to the HR system in order to obtain the employment status of each employee.

For the data entry through a PC, Paychecks.com provides a special application program to be downloaded and installed. This application helps users to enter data more easily as well as provides additional functionalities such as generating tax forms.

To accomplish these goals, Paychecks.com creates a new domain, "Paychecks4tMart" (shown in green in Figure 5), and sets policies for the domain that incorporate tMart's company regulations for any software running in tMart's intranet. The use of TVDs is a value proposition that preserves the goals of all parties involved in these actions:

The tMart CIO. The Paychecks4tMart TVD allows the CIO to monitor the isolation between the tMart intranet applications running on Paychecks.com's servers and any other non-tMart software running on those servers. Also, the CIO may monitor the policy configuration set by Paychecks.com as well as the ongoing policy enforcement by the TVD members.

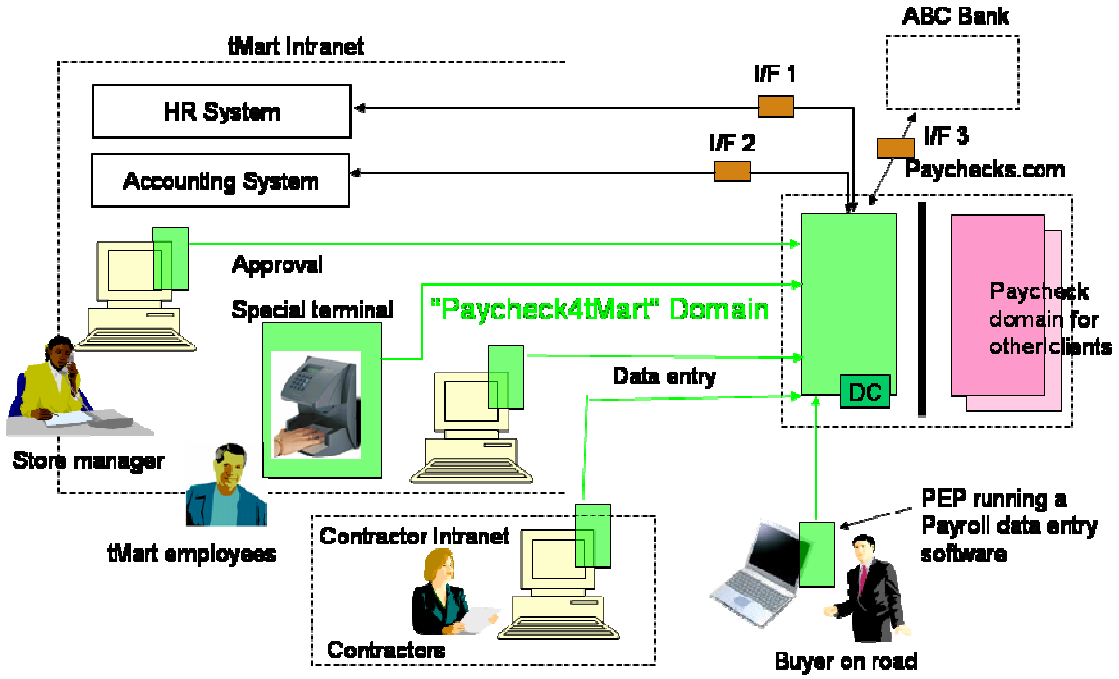

Figure 5. tMart scenario for case study #2.

Paychecks.com. The use of TVDs reduces the internal support cost for Paychecks.com because they are able to export and support a single virtual environment regardless of their heterogeneous physical infrastructure, and they also benefit from the uniform policy configuration and enforcement provided by the TVD infrastructure.

The tMart employees. The employees benefit by the greater protection afforded to their personal payroll information both at data-entry and data-transmission time. Also, the uniform virtualized execution environments may provide a simpler, migratory user experience in which the user's current state is automatically migrated by the TVD infrastructure to the user's currently active machine.

5. TVD challenges and next steps

TVDs provide a long-term vision for heterogeneous computing systems and execution environments that offer strong and verifiable security guarantees, which will serve as the basis for managing relationships between businesses and other entities in a hyper-connected world. As this vision begins to materialize, new applications and programming paradigms will emerge, and systems will even be designed with TVD support in mind—e.g., trusted computing capabilities will be embedded into controllers and subsystems (networking and storage), and policy enforcement mechanisms will be implemented in firmware and hardware. While TVDs will make security management of workloads and relationships easier, and will make for a safer computing ecosystem through the their containment capabilities, these benefits will likely come at the cost of additional resources—more memory, bandwidth, storage and raw computing power—enforcing the constant drive for more powerful as well as more distributed systems.

In the medium term, TVDs will provide the basis for On Demand resource control and metering across virtual enterprises, enabling a true eUtility model that is agile and can be trusted by all parties—customers and vendors alike. Server farms will be able to optimize resource usage like never before while at the same time reducing overhead and total cost of ownership through the autonomic self-managing capabilities and guarantees afforded by TVDs.

We continue to focus our efforts on realizing the full potential of TVDs. Three of the most interesting opportunities in this space include achieving scalability, enabling policy specification and management, and maintaining simplicity.

Scalability of the distributed TVD infrastructure itself will be a significant challenge for the research and development communities. The potential plethora of attestation statements coming from each component of each TVD member might easily overwhelm centralized domain controllers, not to mention TVDs whose members are self-managed—i.e., those that negotiate trust, containment, and policy enforcement themselves. Moreover, the sheer volume of state and audit data will itself become a management and storage issue unless intrinsic logging and retrieval mechanisms are built in to the TVD infrastructure.

Policy specification and management issues represent another significant challenge. One issue involves the creation of a uniform policy language that can express a variety of security requirements in a comprehensive manner. Another issue involves translating each TVD-level policy into platform-specific policies, which represents a crucial step toward supporting cross-platform applications in heterogeneous environments. A third issue involves the type of assurances that can be made by trusted computing hardware and software—in particular, determining how the measurements provided by these components can be evaluated to determine the policy compliance of the configuration parameters and execution states of the TVD members.

Simplicity of the human interaction with TVDs will be a third significant challenge. Historically, adding security features to systems has increased the user-visible complexity and reduced user convenience. For TVDs to be successful, the architecture must enable the human to manage larger and more complex system configurations than is currently possible, without increasing the individual efforts required of the TVD administrators, the application developers, or the users.

6. Conclusion

The authors believe that deployment of TVD technology will significantly increase the level of security visible to IBM's customers while reducing the complexity of security management. We do not claim that TVD technology is a "silver bullet" for solving security problems—there still are many technical challenges and uncertainties on the road ahead—but the TVD concept represents a quantum leap over today's best-effort systems: security must no longer be an afterthought, but rather must be integrated into every piece of an IT system. With the maturity of technologies such as secure hypervisor software and TCG hardware, we now have the opportunity to take a holistic approach to the problem of commoditizing security. TVDs will represent a true step toward the goal of making security that "just works" in our products and for our customers.

Authors

This paper was collaboratively developed by members of IBM Research at the Tokyo Research Laboratory, the Watson Research Center, and the Zurich Research Laboratory. Contributors include Anthony Bussani, John Linwood Griffin, Bernhard Jansen, Klaus Julisch, Günter Karjoth, Hiroshi Maruyama, Megumi Nakamura, Ronald Perez, Matthias Schunter, Axel Tanner, Leendert van Doorn, Els Van Herreweghen, Michael Waidner, and Sachiko Yoshihama.

The authors gratefully acknowledge Charlotte Bolliger, Peter Capek, Hugo Krawczyk, J. R. Rao, David Safford, and Michael Steiner, whose advice and suggestions greatly improved this paper.