The 9th International Conference for Young Computer Scientists

A Generalized Trusted Virtualized Platform Architecture

Anbang Ruan Qingni Shen Yuanyou Yin
School of Software and Microelectronics. Peking University. Beijing, China.
anbang@pku.edu.cn qingnishen@ss.pku.edu.cn yuanyou1206@gmail.com

Abstract

Problems of overall safety management, appropriate load balance, and the need for easy-to-use emerge in an environment containing multiple Trusted Virtualized Platforms. We proposed the Generalized Trusted Virtualized Platform architecture, GTVP, which combines multiple physical platforms as a trusted union. GTVP first establishes trust relationship among all platforms, and then synchronizes their resource and security information for unified management. Moreover, GTVP supports fast and secure migration to resolve the overall load-balance issue. Host OS (as in Xen) of GTVP is divided into five control domains for minimizing TCB and Guest OS of certain application (called as Lazy Box) cut into components for rapid deployment and upgrade. As a result, administrators can manage multiple platforms in a similar way as in a single platform and get the benefits of security, efficiency and easy-to-use while obtaining transparency and flexibility. Three scenarios are demonstrated to show their efficiency in the GTVP architecture.

Keywords: Trusted computing platform, virtualization, migration, trusted virtual machine manager.

1. Introduction

The application of the combination of Virtualization Technology [1] and Trusted Computing Technology [2] becomes popular nowadays. Several studies had achieved this combination, such as Terra [3] and OpenTC [4]. However, most works focus on a separate physical platform. Most enterprise applications require the collaboration of more than one physical platform to provide users with trusted and consistent services. The multi-platform environment faces following three problems.

Security Management Many security threats derive from inappropriate management. For multiple hardware platforms' collaboration, administrators have to deal with details such as creating secure communication channels and guaranteeing trusted sharing and isolation.

Load Balance In a virtualized environment, administrator can migrate virtual machines from over-loaded platform to under-loaded ones for overall load balancing [5]. However, administrator's control, whether directly (manually) or indirectly (scripts executed), is frequently needed, which may bring security threats. Moreover, fast migration for the short-time load balance requirements is another critical issue.

Easy-to-Use In order to deploy applications on a Trusted Virtualization Platform, users should first configure virtual machines, then the Guest *OSes* and at last the applications. After that, he/she needs to adjust security policies in the *TVMM* (Trusted Virtual Machine Monitor) and in the Guest *OS*. Meanwhile, for upgrading and patching, users need to concern every *VM* [6]. Thus, rapid deployment and application management for multiple hardware platforms are also very important issues.

We propose a unified platform -- *GTVP* (Generalized Trusted Virtualized Platform) to tackle above problems. *GTVP* is the combination of separate virtualized platforms (called member platforms or members). These platforms have established trusted relation mutually, and possess all other members' resource and security information. They cooperate via security protocols and manage hardware resources, management strategy and security policies in a uniform manner. Thus, *GTVP* provides a series of unified, transparent and trusted services for its upper application layer.

In *GTVP* architecture, administrator needs only to configure local security policies and launch the connecting process to a target *GTVP*. *GTVP* automatically extends trust chain to the new member and synchronizes its resource and security information with all other members. Henceforth, *GTVP* protect all subsequent communication between members, and synchronize all information updates. *GTVP* automatically migrate applications or *VM*s among members for load balancing, while guaranteeing security by strictly control the entire migration process under security policies. Finally, *GTVP* achieves Easy-to-use property by Lazy Box

978-0-7695-3398-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ICYCS.2008.508

technique, which assembles a minimal run time environment for application dynamically according to previous configured resource and security profiles.

We first incise host *OS* (or Domain 0 in *Xen* [7]) into five domains presiding over different security functionalities. We then disaggregate the guest *OS* (or Domain U in *Xen*) into small *OS* Components, which are building blocks for Lazy Boxes. We also provide methods to optimize dynamic migration, configuration and deployment, such as *OS* Component Cache and parallel migration technologies. Finally, we propose attestation related mechanisms, such as attestation expiration count, attestation information sharing and transferring mechanism to ensure continuous integrity and enhance attestation efficiency.

The rest of this paper is organized as follows. Section 2 introduces *GTVP* architecture, as well as *G-TVMM*, Control Domains and Lazy Box. Section 3 introduces key technologies, such as platform connection, effective migration mechanism, continuous attestation mechanism and secure policy management mechanism. Section 4 describes three scenarios of *GTVP* according to platform design goals. Section 5 discusses related work, and there will be conclusion and future work introduced in section 6.

2. *GTVP* Security Architecture

The following four design principles guide our design of *GTVP* architecture: **(1) Secure:** *GTVP* architecture should be constructed in accord with *TCG* specifications and be designed with security protocols for trusted connection and communication between physical platforms. **(2) Efficient:** Dynamical load balance mainly under the control of secure and fast migration technology in *GTVP* should optimize the overall efficiency and satisfy most applications' requirements on best-effort. **(3) Simple:** *GTVP* should reduce complexity of deployment and improve transparence or easy-to-use, making the operation on a *GTVP* as similar as that on a commodity *OS*. **(4) Flexible:** *GTVP* should leverage security, efficiency and simplicity by flexible policy configuration and maintenance mechanism to support maximum types of security requirements.

Figure 1 outlines our general architecture of *GTVP*. It is a three-layer-structure: Hardware Layer, Virtualization Layer and Application Layer. The Hardware Layer may comprise various sets of hardware with different architectures, at least one of which contains *TPM* [2]. *GTVP* links (Trusted) Virtual Machine Monitors *(TVMM* and *VMM)* from all members via secure connection to form its Virtualization Layer -- Generalized *TVMM* (*G-TVMM).* Control Domains above *G-TVMM* is effectively external implementation of some *G-TVMM* functionalities. *G-TVMM*, Control Domains and all underlying hardware devices compose the *TCB* of the *GTVP*. *GTVP* encapsulates applications in three types of Virtual Machines, named Boxes in *GTVP* terminology, namely *Open Box*, *Close Box* and *Lazy Box*. We derive the former two from Terra [3], and design *Lazy Box* especially for *GTVP*. We will describe *G-TVMM*, Control Domains and Boxes in the following three sections respectively.

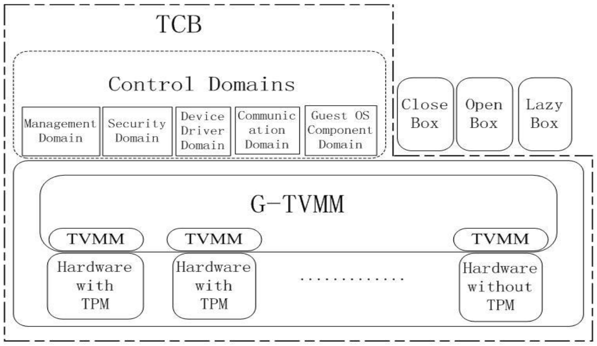


Figure 1. *GTVP* architecture

2.1 *G-TVMM*

G-TVMM plays three roles in *GTVP*. First, it implements the interfaces of *GTVP*. For applications, *G-TVMM* abstracts virtual hardware device interfaces, guarantees isolation and enforces effective resources sharing and communication among applications. For administrators, it provides interfaces for management of virtual machines, security, attestation, migration and overall platform. Further, *G-TVMM* resides directly upon hardware layer. It manages and dispatches hardware resources uniformly, guaranteeing secure and effective access. It provides integrity attestation, cryptology services and key management service from *TPM* for the upper layer. Finally, *G-TVMM* enforces communication of member platforms' *TVMMs* or *VMMs*. *G-TVMM* completes the following tasks for members: information synchronization (hardware resources status, security and management information); hardware resources requisition and concession; mutual attestations; secure migrations, etc.

G-TVMM is the core component in *GTVP* architecture; hence, its security and efficiency directly influence the overall platform. We leave only basic functionalities inside *TVMM*, and extract others out as independent functional components. This decision has following benefits: (1) least privilege control, since auxiliary functions reside outside *TVMM*, one's failure will not lead to the crash of others; (2) low complexity, the scale of *TVMM's* source code is reduced, which makes formal proving of the *TVMM* realizable. Meanwhile, it is convenience for integrity attestation of platform, because only the attestations of necessary components are enough.

Xen [7] is a typical implementation of this approach. Its *VMM* implements only three basic functions, namely *CPU* scheduling, memory management and inter-domain sharing and communication. However, all the remaining functions are implemented in a single privileged domain (named Domain 0). In recent years, *Xen* community makes important steps in incising Domain 0 [8]. They have extracted device drivers form Domain 0, constructing a separate domain -- *DDD* (Device Driver Domain). We embrace *Xen*'s philosophy, and make advance steps: we further incise the remaining Domain 0 into *MD* (Management Domain) and *SSD* (Security Service Domain), and then add *CD* (Communication Domain) and *OSCD* (*OS* Component Domain). These five domains together constitute *GTVP*'s Control Domains.

2.2 Control Domains

Device Driver Domain manages Backend Device Drivers for every *VM*. Backend Device Driver is the "real" driver for physical hardware. On the other side, the Frontend Device Driver is the driver in virtual machines but only forwards request to and receives result from the corresponding Backend Device Driver. This mechanism guarantees isolation between applications and device drivers.

Management Domain is mainly responsible for basic management of *VM*s, such as creation, destroy, suspend and recovery. Administrator can also modify *VM*s' configuration, explicitly assign them for attestation or migration. Moreover, *MD* collects usage information of local hardware resource, including network loading, memory occupation, *CPU* running status and so on. Both administrators and other domains can use these information, for instance, *CD* could use them to make migration decision. *MD* is the interface and entrance for administrator to control the overall platform.

Security Service Domain manages and enforces security policy. It defines policy transition rules for local platform to synchronize its policy with overall platform. In addition, security services domain utilize trusted physical devices, e.g. *TPM* to provide extra security services, such as encryption and decryption, key management and integrity attestation. This domain employs various mechanisms to guarantee the efficiency of integrity attestation process.

***OS* Component Domain** manages *OS* components for Lazy Box. It automatically instantiates components by pre-defined configuration files. *OSCD* maintains a component cache, which stores components used recently. It enables fast migration and rapid deployment. We will describe it in the following section. On the other side, the higher frequency the component used the more possibility of being attacked. Hence, we assign an attestation expiration count for each component. *GTVP* attests to or reload the component whose used time exceeds the count. As a result, GTVP focus attestation efforts on components with most security needs. For example, we assign lower counts to components with high-level sensitivity, so they can have higher attestation frequency.

Communication Domain connects member platforms and synchronizes their configurations. It implements a set of security protocols for inter-members communication, namely *HSP* (Handshake Protocol), *CSP* (Configuration Synchronization Protocol), *RRP* (Resources Requirement Protocol), *SMP* (Secure Migration Protocol) and *SCP* (Secure Configuration Protocol). *HSP* manages the process of a platform joining or leaving *GTVP*. *CSP* synchronizes the resource configuration and corresponding security and management information of local platform to all other members. For dynamic information synchronizing, *GTVP* adopts a requesting approach, which is accomplished by *RRP*. If local platform is over-loaded, it multicast request for hardware resources, and migrates some of its *VM*s to appropriate platforms. *S*ecure and efficient migration is implemented by *SMP*. Finally, *SCP* synchronizes the secure policy of local platform with all other members according to the pre-defined policy transition rules.

2.3 Virtual Machines

Virtual Machines are the run-time environments for applications. *GTVP* supports three kinds of *VM*s: Open Box, Close Box and Lazy Box. Open Box is a typical virtual machine equipped with commodity *OS* and provides appearance of general-purpose platform. Close Box is a virtual machine with specialized executing environment and especially *OS* configured explicitly for particular application. Lazy Box provides applications a minimum executing environment by combining appropriate *OS* Components from *OSCD* dynamically. It enables the features of safe and rapid deployment, convenient upgrade, secure and efficient migration and attestation.

For rapid deployment, administrator provides application image and a list of dependency, e.g. version of *OS* kernel, various dependent libraries, etc. *GTVP* first resorts to *OSCD* for necessary dependencies, then from other members, and at last, acquires remaining from administrators. *GTVP* attest to all components, guaranteeing their integrity. When application runs for the first time, *GTVP* combine necessary components to form a Lazy Box for encapsulating the specific application. Each time when application encounters a dependency-missing-fault, *GTVP* activates related components in *OSCD*, and links them with the Lazy Box. For efficient migration, *GTVP* construct a identical Lazy Box at target member, and simply migrate the dynamic part of application, e.g. *CPU* status, stacks and heaps from memory, etc. *GTVP* protects the entire process of migration by attestation mechanism, security protocol and security policies. For efficient attestation, *GTVP* attest to only the needed components, reducing attestation overhead, while gaining security by attesting secure concerning components more frequently. We will examine migration and attestation related mechanism in Section 3.3 and 3.4 respectively.

3. Methods for Constructing *GTVP*

3.1 Platform Connection

GTVP seamlessly connects all the members in three steps, forming a unified platform. Firstly, it connects every member's *TCB* to form its virtual *TCB* or *vTCB*, which shields bottom details for the upper layer, such as hardware configurations and network topology. Each member preserves all members' configuration. This requires: (a) Mutual trust be established between members. In *GTVP* terminologies, it is the Horizon Extension of trust chain. During the stage of platform establishment, *GTVP* utilizes *HSP* to establish trust relationship, which utilizes remote attestation mechanism from *TCG*. During normal operation, *GTVP* protects connection between members with various security protocols. (b) Platforms must synchronize resource information in a safe and reliable way. *GTVP* utilizes *CSP* for safe and efficient information transmission, including hardware configuration, security information, trust and other security information.

Secondly, *GTVP* combines security information of all members, forming a unified global security configuration, which includes the security level of all resources among every member and security policy of every member. We utilize *SCP* for unified management and enforcement of overall security policy.

Lastly, from the application point of view, in accordance with its resource demand, *GTVP* migrates it among the members dynamically. We will investigate migration mechanism in the next section.

3.2 Enforcing dynamic and fast Migration

The core of *GTVP* is its efficient and secure migration mechanism. *GTVP* enables applications to access resources among all members by migrating applications to the member that possesses the needed resources. There are mainly two types of migration techniques: virtual machine migration [5] and process migration [9]. The first transfers the entire virtual machine. It supports both checked-point migration (i.e. suspending-copy-awakening) and live migration. However, its overhead is too high for short-term load balancing. Process migration technique adds an external pack to the process, which reduces process's dependence to local platform. Hence, migration is performed by transferring the entire pack directly. The overhead is relative small but it suffers the pain of platform heterogeneous.

The migration mechanism in *GTVP* takes the advantages of both techniques while avoiding their shortcomings. First, *GTVP* still migrates virtual machine. It acts as a middle-ware between *VM* and hardware, hence it provides a homogeneous platform. Meanwhile, we derives the techniques developed in [10] to incise the virtual machine into pieces (*OS* components), the source member transfers only the needed components, reducing the transferring overhead.

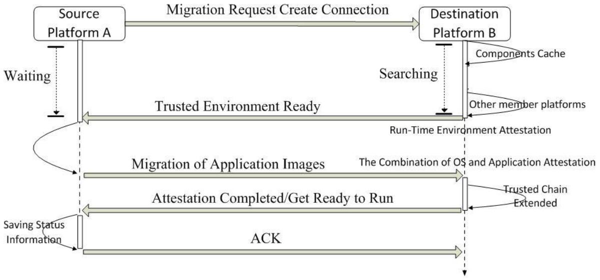


Figure 2. Migration process

Figure 2 demonstrates the migration process. Target member can have multiple sources for *OS* components: it first searches the needed components in component cache, and then it searches the static local image files. Further, it sends requisition to other members for needed components. Because the needed components may exist in more than one member, the target can select different members as sources for different components, realizing multi-source parallel migration.

All members have attested to each other before migration, and the entire migrating process is protected by security protocol. When the migration completes, *GTVP* attest related components and extend the Vertical trust chain. Thus, *GTVP* guarantees the security of migration. When *GTVP* accomplishes all transferring and attestation, it generates the specific Lazy Box at the target member.

Since the application on platform is relatively stable, components increase at a relatively slow speed. In addition, with cache mechanism, components can quickly distribute to every members of the platform. In the vast majority of time, *GTVP* simply migrates application and it corresponding state. If applications are further divided into static and dynamic parts, *GTVP* can just transfer the dynamic one (running states), which will further reduce the migration cost.

3.3 Continuous and Valid Attestation

The attestation techniques [2] in current trusted platform have two major drawbacks: (1) Large granularity. In most of time, we only need to attest parts of the system. (2)Lack of continuous guarantee. *TCG's* attestation mechanism is performed at system initializing or application loading. It cannot guarantee that the process will always be in an integrated state.

GTVP only need to attest those needed *OS* components. In addition, fine-grained control can enhance attestation efficiency by parallel attesting. On the other hand, *GTVP* assigns different *OS* Components different attestation expiration counts according to their security levels. Components will no longer be considered safe and must be attested or loaded again when their invoking times exceed theirs count. Hence, for applications or components with higher security needs, *GTVP* assigns them higher attestation frequency. Otherwise, *GTVP* saves attestation efforts for efficiency.

Attestation information can be shared among members as long as trust relationship has been established. As a result, applications residing on different members do not need to attest each other from the bottom step-by-step. As long as they are trusted by their local platform, they can trust each other. For example, there is an application A on platform B and C on platform D. At the time of loading, A and C were attested to their local member respectively, i.e. B and D, since B and D have reached trust relationship, A and C can trust each other naturally.

Any member can send their trust information to others, as long as they trust mutually. Thereby *GTVP* reduces duplicate attestations. However, this will bring potential safety problems, e.g. components with vulnerability would be trusted by more members. This flaw can be make up by attestation expiration mechanism – according to security needs, administrators can choose the intensity of component attestation. This is another proof of *GTVP* 's flexibility.

3.4 Security Policy Management

Security policy is the soul of *GTVP* because it controls the three most important functionalities: resource management, migration decision, and attestation intensity. *GTVP* enforces strict supervision upon applications' access to resources once the administrator have labeled the subjects and objects and specified the policy. Furthermore, *GTVP* automatically migrates applications to appropriate members when local resources are unavailable or insufficient, as long as those applications have permission to access target resources. Again, this action is supervised in accordance with security policy. In addition, as described above, object's security level also affects the attestation expiration count, which further affects the attestation intensity.

When a new platform is joining a *GTVP*, its local security information will be synchronized with every member. There are two typical methods to achieve this synchronization [11]: (1) the new platform is inherently dedicated to a particular *GTVP*. Therefore, it configures its security information in accordance with that *GTVP*. This method facilitates the platforms with lower mobility needs, i.e. they only connect to a designated *GTVP*. (2) The new platform has its own pattern of security information, and predefines transition rules for each targeted *GTVP*. This method is applicable to the platforms with higher mobility needs, e.g. a notebook needs to join different *GTVP* at different time.

In order to differentiate members in the *GTVP*, we propose the concept of platform identity. Platform identity is determined by the identification of the current user of the local member. It represents the current security status of the local member, i.e. it is a reference security level. It indicates the highest security level the subjects and objects in the member can obtain, and it restricts the inward migration of applications with higher security level. Furthermore, the identity could also aid clarifying the relationship among members. In some scenarios, members with higher security identity can gain more control. For example, when we organize members with a star topology, the central unit can be the member with the highest identity.

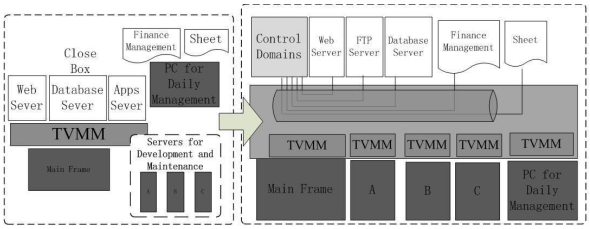
4. Case Study

In this section, we will demonstrate concrete examples for how GTVP satisfies all three requirements

4.1 Dynamic Load Balancing

Tradition Platform: Suppose a company supplying web services. It employs a mainframe hosting its three servers, namely a Web server, a database server and an application server, with each server running inside an individual *VM*; three computers with middle-processing capability, with each hosting two *VMs* deployed with services development and testing environment respectively; and a commodity *PC* for daily management (Figure 3(a)).

Requisitions for services tend to concentrate in certain time of the day, and machines for development and testing may only be over loaded during the process of compiling, debugging and testing. For overall load balancing, administrator may migrate appropriate applications among different platforms. Either manual or script-controlled will introduce extra manage complexities and security vulnerabilities.



a: Tradition Method b: GTVP Method

Figure 3. Dynamic Load Balancing

GTVP: *GTVP* deploys application on arbitrary member, and migrates them dynamically (Figure 3(b)). For example, when web server acquires more processing capability, *GTVP* migrate it to a member with sufficient resources and appropriate security attributes. *GTVP* balance the loading of overall machines dynamically, reducing hardware cost and management complexities.

In addition, once administrator discovers the needs for adding computing capability, what he needs to do is just connect a new machine it to the *GTVP*. *GTVP* takes efforts to re-balance the overall platform loading. The entire process is just like to "hot-plug" new computing power to the *GTVP*. Furthermore, platform can also contract as needed, e.g. for maintenance or cost saving purpose.

4.2 Easy-to-use

Tradition Platform: Following previous scenario, there are three servers, three development environments, three testing environments and a management environment, each of which needs a *VM*. Therefore, there would be ten *VM*s to deploy. Administrator needs to configure each virtual machine's virtual resources and *OS*, which brings much management burden. On the other hand, when the underlying infrastructure needs patch of update, administrators have to repeat these tasks for every *VM*.

GTVP: With Lazy Box, *GTVP* automatically generates *VM* and Guest *OS* for applications. Users could install application in a similar way as in traditional *OS*, e.g. Linux. He or she only has to provide an extra dependency list and specifies performance and security configuration. In addition, patching and upgrading in GTVP become convenient, because administrators just need to replace certain components with a patched or upgraded one in OSCD. For example, when we need to upgrade certain Lazy Boxes' OS kernel, instead of deploying, installing, and rebooting to switch to it on each Box, we deploy the new kernel in arbitrary member, and simply change the dependency configuration of candidate Boxes. We then inform GTVP to perform the transition, which simply builds the new environment, and migrates related dynamic status.

4.3 Security Management

Tradition Platform: Administrators have to take more consideration for lower layer details, such as the security connection and the resources sharing among different machines. As the dimension of machines grows, the management complexity soon becomes tremendous, which introdoces both manage burden and security vulnerabilities.

GTVP: Firstly, *GTVP* provides a uniform *vTCB* for administrators, which hides the details of the underlying connection and guarantees that all members connect each other seamlessly and reliably. Secondly, *GTVP* makes efforts to alleviate the management burden (for easy to use). Administrator only needs to configure local security information and *GTVP* synchronizes it with all members automatically. *GTVP* guarantees the continuous integrity of the entire platform (security) and dynamic load balancing (efficiency). Meanwhile, administrators could adjust the various aspect of *GTVP*, such as scheduling strategy, memory-allocating algorithm, migration and attestation strategy, and the granularity of *OS* components, achieving maximum flexibility.

5. Related Work

Terra [3] achieves the combination of trusted computing and virtualization technology by use of TVMM that partitions a tamper-resistant hardware platform into multiple, isolated virtual machines. The software stack in each VM can be tailored from the hardware interface up to meet the security requirement of its applications. TVMM functions to guarantee securities such as security of root, authentication and trust road. Unfortunately, Terra only concerns the scenario of single-platform. We adopted the Open Box and Closed Box from Terra, and added Lazy Box for our purpose.

Virtual Infrastructure (VI) [12] decouples the entire software environment from its underlying hardware infrastructure. It enables the aggregation of multiple servers, storage infrastructure and networks into shared pools of resources that can be delivered as needed. It achieves the uniform management and dynamic load balancing. However, it emphasizes efficiency rather than security, while GTVP protects every aspect of the platform by security mechanism. Furthermore, the migration in GTVP can be more efficient, because it only migration the dynamic parts of applications, which may be rather small, instead of copying the entire VM file in VI.

Public resource computing [13] involves an asymmetric relationship between projects and participants. Computing distributes among participants. Most participants are individual PC owners. BOINC [14] is a typical public resource-computing platform. It supports redundant computing, cheat-resistant accounting, and support for user-configurable application graphic. However, it has no control over participants, and cannot prevent malicious behavior. In GTVP, we demonstrate a virtual layer as middleware for integrity attestation of platforms and trust establishing.

6. Conclusion and Future Work

GTVP can layer upon more than one hardware platform and guarantee their trust relationship with techniques provided by TCG. It first extends the trust chain horizontally, then synchronizes information among these members and extends the trust chain vertically to applications. Henceforth, administrators can manage the overall platform in a uniform manner, without concerning complicated underlying details while gaining security guarantees. GTVP automatically combines necessary *OS* Components to form a Lazy Box for the target application. Thus, it supports rapid application deployment and management, alleviating administrators burden maximally. GTVP achieves rapid migration by first assembling the identical Lazy Box at the target member, and then migrates the dynamic parts, with migration decision made in accordance with overall security policies, and the entire migration process protected by security protocols. Finally, GTVP's TCB was reduced for least privilege. We extracted functionalities from G-TVMM, creating five Control Domains.

For future works, we will first design the five protocols in detail, prove their security, and evaluate their effectiveness and performance. Then we will device the algorithms in OS Component Cache, and techniques for locating and loading OS Components. In addition, we will examine Failure Tolerance techniques.

Acknowledgment

This paper is supported by the Foundation of Science and Technology of Huawei of China under Grant No.YJCB2008001TS.

References

[1] R. Goldberg, "Survey of Virtual Machine Research," *IEEE Computer*, 7(6), June 1974.

[2] Trusted Computing Group, Trusted Platform Module (TPM) specifications, Technical Report, *https://www.trustedcomputinggroup.org/specs/TPM,*2006.

[3] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh, "Terra: a virtual machine-based platform for trusted computing", *In ACM Symposium on Operating Systems Principles (ASOSP)*, 2003, pages 193～206.

[4] Dirk Kuhlmann, Rainer Landfermann, HariGovind V. Ramasamy, Matthias Schunter, "An Open Trusted Computing Architecture-Secure Virtual Machines Enabling User-Defined Policy Enforcement", *Research Report RZ3655*, IBM Research Division, August, 2006.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt and A.Warfield, "Live Migration of Virtual Machines", In NSDI, 2005.

[6] T. Garfinkel and M. Rosenblum, "When Virtual is Harder than Real: Security Challenges in Virtual Machine-based Computing Environments", *in Proc. 10th Workshop on Hot Topics in Operating Systems (HotOS-X)*, May 2005.

[7] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauery, Ian Pratt, Andrew Warfield, "Xen and the Art of Virtualization", *in Proceedings of the nineteenth ACM symposium on Operating systems principles*, Bolton Landing, NY, USA, 2003, 164～177.

[8] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. "Safe hardware access with the Xen virtual machine monitor", *in 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure(OASIS),* Oct 2004.

[9] F. Douglis and J. Ousterhout, "Transparent Process Migration: Design Alternatives and the Sprite Implementation", Software - Practice and Experience, August 1991.

[10] Melvin J. Anderson, Micha Moffie, Chris I. Dalton, "Towards Trustworthy Virtualization Environments: Xen Library OS Security Service Infrastructure", Trusted Services Laboratory, HP Laboratories Bristol, April 30, 2007.

[11] Reiner Sailer, Trent Jaeger, Enriquillo Valdez, Ram´on C´aceres, Ronald Perez, Stefan Berger John, Linwood Griffin, Leendert van Doorn, "Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor", IBM T. J. Watson Research Center, Hawthorne, NY 10532 USA.

[12] Virtual Infrastructure Overview *http://www.vmware.com/technology/virtual-infrastructure.html.*

[13] J.P.R.B. Walton, D. Frame and D.A. Stainforth, "Visualization for Public-Resource Climate Modeling", *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization*, 2004.

[14] David P. Anderson, "BOINC: A System for Public-Resource Computing and Storage", Space Sciences Laboratory University of California at Berkeley.