DTSTM: Dynamic Tree Style Trust Measurement Model for Cloud Computing

Zhen-ji Zhou [1], Li-fa Wu [1], Zheng Hong [1], Ming-fei Xu [1], Fan Pan [1]
[1] Institute of Command Information System, PLA University of Science and Technology, No.1 Haifu Street, Nanjing, Jiangsu, China
[e-mail: zhou_zhenji@163.com, wulifa@vip.163.com, hongzhengjs@139.com, hixumingfei@163.com, dynamozhao@163.com]
*Corresponding author: Li-fa Wu

Received September 1, 2013; revised December 16, 2013; accepted January 11, 2014; published January 29, 2014

Abstract

In cloud computing infrastructure, current virtual machine trust measurement methods have many shortcomings in dynamism, security and concurrency. In this paper, we present a new method to measure the trust of virtual machine. Firstly, we propose "behavior trace" to describe the state of virtual machine. Behavior trace is a sequence of behaviors. The measurement of behavior trace is conducted on the basis of anticipated trusted behavior, which not only ensures security of the virtual machine during runtime stage but also reduces complexity of the trust measurement. Based on the behavior trace, we present a Dynamic Tree Style Trust Measurement Model (DTSTM). In this model, the measurement of system domain and user domain is separated, which enhances the extensibility, security and concurrency of the measurement. Finally, based on System Call Interceptor (SCI) and Virtual Machine Introspection (VMI) technology, we implement a DTSTM prototype system for virtual machine trust measurement. Experimental results demonstrate that the system can effectively verify the trust of virtual machine and requires a relatively low performance overhead.

Keywords: trust measurement, trusted computing, cloud computing, virtual machine, behavior trace

This work was supported by the National Nature Science Foundation of China under Grant No.61070173; The Nature Science Foundation of Jiangsu China under Grant No.BK2011115 and No.BK20131069; The opening Foundation of Laboratory of Military Network Technology. The authors would like to thank the anonyous reivewers for their valuable comments which improved the quality of this paper.

http://dx.doi.org/10.3837/tiis.2014.01.018

1. Introduction

Virtualization technologies have been widely applied in cloud computing to package computing, network and storage resources which are provided to tenants. It is curcial to measure whether the cloud computing platform runs buggy, malicous application codes or is improperly configured. Thus, how to build a trustworthy virtual environment has become a key challenge to ensure the security of cloud computing platform [1] and it has attracted wide attention of researchers.

Traditional security management methods for virtual environment usually depend on software level security and lack a trustable base [2]. It is hard to achieve a trusted virtual machine with traditional manner in cloud computing environment. Regarding this problem, an effective solution [3] is to combine trusted computing with cloud computing and provide trusted cloud service in a verifiable manner. Santos [4] initially proposed trusted cloud computing and designed a Trusted Cloud Computing Platform (TCCP). The platform starts from a trusted platform module which built in hardware of cloud platform, and conducts trust measurement level by level. In this way, a trusted virtual environment with ensured security is finally built up. Similar studies were provided by researchers in [5][6][7][8][9]. These studies are based on the Trust Measurement Scheme by TCG [10], which is used to ensure trustworthiness of traditional PC platform at booting stage. They do not accommodate a dynamic and multi-tenant virtual environment such as cloud computing. The major defects of these solutions are listed as follows:

- Static measurement. In cloud computing, virtual machines are usually used on demand. The programs running in user domains may change trustworthiness of the whole virtual machine. Current methods which use static measurement can only ensure trustworthiness of Virtual Machine Monitor (VMM) during booting stage [11] [12]. A method that can accurately describe the state of running virtual machine is needed.
- Weak concurrency. A VMM often monitors a system domain and a large number of user domains. Chain style measurement model can hardly deliver real-time and accurate trustworthiness to user domains with complicated structures [11]. Moreover, some user domains may constitute a trust domain according to specific security policy based on business demands [13][14], which results in a more complicated relationship of trust. As a result, an automatic and centralized trust measurement mechanism for virtual machine is required.
- Poor security. Traditional trust measurement methods such as IMA [15] can not resist the attack in kernel mode. For example, IMA depends on kernel to ensure the correctness of verification results when adopting Linux Security Model mechanism to implement integrity verification. Thus, measurement of virtual machine trust in cloud computing infrastructure requires the ability of kernel attack resistance.

Regarding the above problems, we present Dynamic Tree Style Trust Measurement Model (DTSTM) for cloud computing. The model measures behavior trace instead of virtual machine state which is difficult to describe, and reduces complexity of virtual machine trust measurement while ensuring its security at runtime stage. Behavior trace is a sequence of behaviors, we formally define it in section 2. In order to solve security and concurrency problems, our model measures system domain and user domain on the basis of configuration

and behavior trace. Moreover, DTSTM ensures security of the measurement module itself by separating measurement module from monitored virtual machine through the application of SCI and VMI technology. We leverage real-time monitoring of kernel, key data, configuration and application behavior of user domain to offer dynamic trust measurement of virtual machine.

The remainder of this paper is organized as follows: In Section 2, we formally describe a trust measurement theorem of virtual machine state based on the concept of behavior trace, which builds the theoretical founation of our work. In Section 3, we introduce the structure and workflow of DTSTM and analyze security of this model. Section 4 presents a DTSTM-based trust measurement system and evaluates effectiveness and performance of our implementation. In section 5, we discuss related work. Section 6 draws conclution and summarizes future work.

2. Virtual Machine Trust Validation

In this section, we firstly propose the concept of behavior trace, and then describe virtual machine state on the basis of behavior trace. Thereafter, a measurement method of virtual machine behavior in cloud computing infrastructure is provided. Finally, two virtual machine trust measurement methods are inferred from the virtual machine trust validation theorem.

2.1 Description

According to the definition of TCG, trustworthiness of a computing platform state depends on whether the platform's behavior complies with its anticipated policy [10], therefore the state of a virtual machine is affected by its behavior. In virtual machine trust measurement, any virtual machine behavior that does not conform to anticipated policy will damage trustworthiness of the virtual machine. Thus, virtual machine trust validation can be realized through measurement of the virtual machine behavior. For the purpose of description, we give relevant definitions as follows.

DEFINITION 1 (**Behavior trace**) Behavior trace $R=b_1b_2\ldots b_n$, where b_1, b_2, …, b_n represents virtual machine behavior. Behavior $b\in B=(S\times O\times A)$ means an operation initiated by a subject and performed on an object. $S=\{s_1, s_2, \ldots, s_n\}$ is a set of behavior subjects including the programs started by the user, $O=\{o_1, o_2, \ldots, o_n\}$ is a set of behavior objects. A single object $o\in O$ can be a resource such as a document, a program and an equipment, $A=\{r, w, e\}$ is a set of access operations such as read, write and execute.

DEFINITION 2 (Trusted behavior) A specific behavior b of virtual machine is trusted means behavior b complies with the anticipated policy of trusted behavior from the inquiry party.

DEFINITION 3 (Virtual machine state) Virtual Machine State Set $N=(R\times E\times H\times F\times J)$ indicates a collection of active subject behaviors and behavior attributes in a certain state. In the expression, $n\in N$ indicates a single state within Virtual Machine State Set, and active subjects in State n are denoted as S_n. R indicates behavior trace, and behavior trace of State n is denoted as R_n. $E=\{e_1, e_2, \ldots, e_n\}$ indicates a set of expected values of system integrity measurement, and a set of expected values for trust in State n is denoted as E_n. $H(s)$ is a function that measures integrity of Active Subject s. F is a function that checks policy conformance of the subject behavior. If Behavior b conforms to security policy, $F(b)=True$; otherwise, $F(b)=False$. For example, a policy p contains two rules r_1 and r_2. r_1 is user a_1 can read file f_1. r_2 is user a_2 can't read file f_1. If a_1 reads file f_1, F determine the behavior complies with p. If a_2 reads file f_1, F determine the behavior does not comply with p. J is a function to

determine whether a state is trustable. If State n is trustable, $J(n)=Trust$; otherwise, $J(n)=Untrust$.

2.2 Measurement

Virtual machine platform is composed of various hardwares and softwares with certain functional attributes, which are called components. Thus, to measure virtual machine behavior means acquiring relevant attributes of component behavior. There are two types of components in virtual machine: system component and application component. The former provides basic system service for user components while the latter mainly implements various tasks issued by the user. For different features of these two types, the strategies and methods to measure their behaviors are also different.

1) System component. System component is mainly composed of kernel module, system dynamic link library and relevant configuration files. When a virtual machine starts, system kernel will be loaded first. At this moment, virtual machine state can be regarded as the initial state for absence of user during booting stage. Meanwhile, booting sequence of system kernel, code module and input/output is relatively fixed. Thus, trustworthiness of the state can be determined by integrity. There is a set of Platform Configuration Registers (PCR) within TPM, and each PCR is associated with a specific event state of a certain system during booting stage. When an event occurs, hash values corresponding to the virtual machine program will be calculated and expanded to PCR. After the booting stage, hash values will be used to verify integrity of the system component.

2) Application component. Application component consists of executable program, application dynamic link library and relevant configuration files. Since component has diverse inputs and often executes in a concurrent mode, the composition of operating environment is complicated. Applications implement user task through a series of behaviors which may affect trustworthiness of the system. Therefore, trustworthiness of a behavior lies not only on the integrity of file component itself but also input data and environment. To determine whether a component will cause detrimental effect to system means verifying whether the component can cause damage to virtual machines. In order to determine whether application environment of virtual machine is trustable, it is required to verify the sensitive behaviors of all application components in runtime stage.

However, each component may produce sensitive behaviors that would affect the system. For example, both a trusted component and a malicious one may read and modify sensitive resources of the system. Thus, it is not reasonable to determine whether a component is trustable only on the basis of existence or absence of sensitive behaviors. If the behavior trace of component can be analyzed in a meticulous way and a comprehensive description of behavior trace is provided, it is possible to objectively discover whether a component produces malicious behavior. To precisely measure the state of current virtual machine, a prerequisite is to acquire behavior trace of the components.

2.3 Validation

Based on system component and application component behavior, following validation policy is defined to determine whether a virtual machine is trustable.

DEFINITION 4 (Trust validation policy of virtual machine state) In a certain state, if all active subjects in the virtual machine are trusted subjects and the behavior traces of all subjects comply with the security policy, the virtual machine state is trustable, namely:

$$\forall s((s\in S_n)\wedge(H(s)\in E_n))\wedge\forall b((b\in R_n)\wedge(F(b)=True))\Rightarrow J(n)=Trust$$

The definition indicates that trusted behavior of a trusted subject ensures a trusted virtual machine state. Any untrusted subject or behavior would result in untrusted virtual machine state. To ensure effective implementation of measurement, report and determination mechanism of behavior, all active subjects of trusted virtual machine state must be trusted subjects. Therefore, trusted system behavior will not affect trustworthiness of the virtual machine state.

As previously mentioned, the virtual machine state is affected by system behavior. Transfer function $Q(b, r_i)=r_{i+1}$ indicates that occurrence of system behavior b renders it possible for virtual machine state to transfer from r_i to r_{i+1}. More generally, if R represents a system behavior trace, $Q(R, r_i)=r_{i+1}$ indicates that occurrence of system behavior trace R renders it possible for virtual machine state to transfer from r_i to r_{i+1}. The theorem below can be inferred from Definition 4.

THEOREM (Trust validation theorem of virtual machine state) If virtual machine state r_t is trustable at the moment t and each system behavior b_i ($1\le i\le n$) within system behavior trace of virtual machine ($R=b_1b_2\ldots b_n$) is trustable after moment t, the virtual machine state $r_{t+1}=Q(R, r_t)$ after the occurrence of system behavior trace R is trustable.

Since state space of virtual machine is usually infinite, it is difficult to directly determine whether a state is trustable or not. The trust validation theorem only needs to confirm whether the virtual machine state at a certain moment and each system behavior after that moment are trustable. This measurement method is more feasibile in a real system and trust validation can be effectively performed for any system state.

DEFINITION 5 Assuming B^T indicates a set of system behaviors related to the trustworthiness of a virtual machine, R_1 and R_2 are two system behavior traces. R_2 is a trust relevant behavior trace of R_1 if the following conditions are simultaneously satisfied.

— For any system behavior a in R_1, if $a\in B^T$, a also belongs to R_2;

— For any system behavior b in R_1, if $b\notin B^T$, b does not belong to R_2;

— The sequence of system behaviors in R_2 is consistent with those in R_1.

For example, $B^T=\{b_1, b_2, b_3\}$, $R_a=b_4b_2b_1b_3b_5$, $R_b=b_2b_1b_3$, R_b is a trust relevant behavior trace of R_a. Only b_1, b_2 and b_3 can affect trustworthiness of the virtual machine state while b_4 and b_5 do not. The behaviors that do not change the trust of virtual machine state can be omitted.

The inference below can be made based on the above definitions and trust validation theorem of virtual machine state.

INFERENCE Assuming R_s is trust relevant behavior trace of system behavior trace R, if virtual machine state r_t at the moment t is trustable and all system behaviors in R_s are trustable, virtual machine state $r_{t+1}=Q(R, r_t)$ after occurrence of system behavior trace R is trustable.

The inference indicates that in case of determining whether a virtual machine state is trustable, it only needs to verify the trustworthiness of the system behavior(e.g., writing system files, updating system policy) which changes the trust of virtual machine state. In the foregoing example, we only have to analyze R_b instead of R_a. This manner makes it possible to reduce the number of objects to be measured and decrease calculation required to verify the virtual machine state.

Based on the trust validation theorem and its inference, two different measuring methods are applied to booting stage and runtime stage of the virtual machine respectively.

1) Integrity-based measuring method. Measure integrity of VMM, relevant executive program of system domain and initial state of configuration file after the booting of platform to determine trustworthiness of the behavior measurement mechanism.

2) Behavior trace based measuring method. Measure state at one moment and system behavior trace after that moment to determine whether the virtual machine state is trustable.

3. Dynamic Tree Style Trust Measurement Model

In this section, we firstly analyze the shortcomings of current measurement schemes, and then provide DTSTM model which combines integrity-based measurement and behavior trace based measurement according to the features of cloud computing infrastructure. Finally we present workflow of the virtual machine measurement and analyze the security of DTSTM.

3.1 Challenges

A patch of the system, an upgrade or an alteration of an application, can change trustworthiness of a trust computing platform. A virtual machine which can be trusted at the booting stage may not be trusted at the runtime stage. Thus, dynamic trust measurement is required.

In virtual environment, a real machine is equipped with a VMM component which executes several virtual machines, and a large number of application components execute on the virtual machines. The method provided by TCG specification can only conduct static measurement of traditional PC platform state, and lack the ability to dynamically measure trustworthiness of a running virtual machine. Additionally, trust measurement in the method assumes a one-to-one relationship which can hardly be developed into a one-to-many parallel relationship in virtual environment.

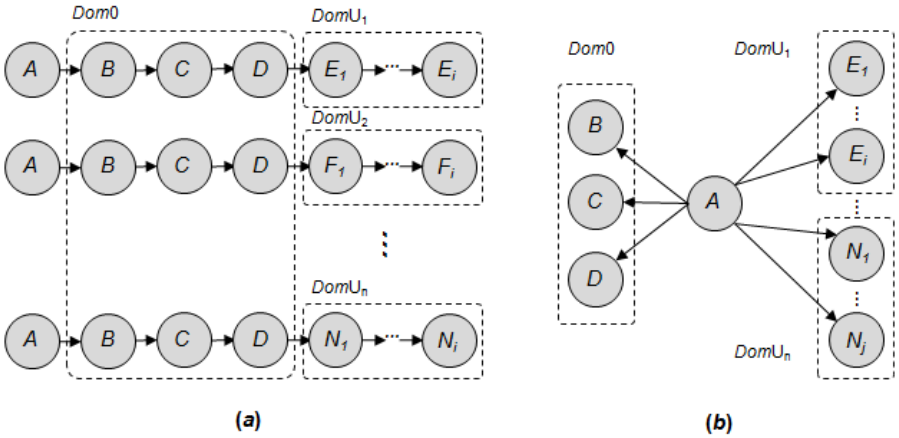


Fig. 1. Chain Style and Star Style Measurement Model

Sadeghi [16] and Chen [17] separately introduced the property-based attestation which employs TPM to measure the security properties without revealing the exact configurations of a target platform, but it is a tough nut to define property in cloud computing infrastructure. A direct solution for dynamic virtual machine measurement in cloud computing infrastructure is to expand a single chain style measurement [18][19]. When a new user domain is created, we can use Flicker [18] or TrustVisor [19] to restart measurement from bottom. The procedure is shown in Fig.1 (a). Therein, A is the root node; B to D are component nodes of system domain $Dom0$; E_1 to E_i refer to all component nodes within user domain $DomU_1$. The specific procedure goes in this way: Root node A measures nodes B, node B measures nodes C, node C measures nodes D. If they are verified trustable, node D measures nodes E_1, node E_1 measures node E_2 and so on. However, in cloud computing environment, events such as boot of virtual

machine and load of module rarely occur, while creation and deletion of user domains are frequent. For this reason, the simple expansion method assumes low efficiency, and simultaneous measurement of several user domains may fail [20] (e.g. during measurement, $Dom\mathrm{U}_1$ would lock up the whole virtual computing environment from $Dom\mathrm{U}_2$ to $Dom\mathrm{U}_n$).

If a trust domain is built up based on star style model [21][22], all measurement is to be finished by one node, as shown in **Fig. 1 (b)**. Node *A* is center node responsible for all measurement, and it would first measure *B*, *C* and *D*. If the nodes are verified trustable, node A will then measure E_i to N_j. However, there are a great variety of components operating in multi-tenant environment. To measure all the other nodes, center node would become complicated with poor expandability, as the center node *A* must accurately identify all components in $Dom\mathrm{U}_1$ to $Dom\mathrm{U}_n$. The implementation is extremely difficult in a multi-tenant environment like cloud computing.

To sum up, there are two challenges for measuring the trust of virtual machine in cloud computing infrastructure: (1) dynamism of measurement. If software and hardware configuration and the state of the virtual machine change during runtime stage, it should be guaranteed that trustworthiness of the changed virtual machine will be measured; (2) concurrency of measurement. There are multiple operating virtual machines that belong to different trust domains. To avoid DoS attack, it should be prohibited to lock up other trust domains for measurement of one virtual machine.

3.2 Structure

To address the above problems, we provide a dynamic tree style trust measurement model called DTSTM which separates trust relationship between system domain and user domain. The trustworthiness of user domain will be measured by measurement module within system domain. If both of them are trustable, trust will be delivered to different user domains thus forming a parallel trust relationship. The structure of DTSTM is shown in **Fig. 2**.

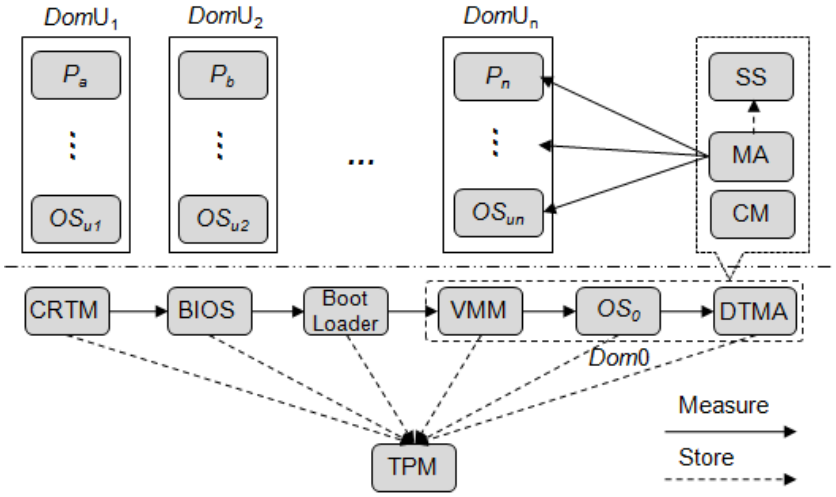


Fig. 2. The structure of DTSTM

The integrity-based measurement mentioned in last section is adopted during booting stage of virtual machine. Since the loading sequence of modules is fixed, possibility of the module change is low, and the system is relatively stable. Behavior trace based measurement is applied to runtime stage for the users of virtual machine usually execute concurrently and the components change frequently.

DTSTM is mainly composed of two parts. The upper part includes six components which are Core Root of Trust for Measurement (CRTM), BIOS, Boot Loader, System domain Kernel OS_0 and Dynamic Trusted Measurement Agent (DTMA). Before DTMA executes, DTSTM builds up the trust chain based on traditional chain style measurement. When the model starts up, CRTM will first measure the integrity of BIOS. If BIOS is trustable, bootloader and VMM will be measured until the trust boundary is expanded to *Dom*0. TPM will store the measuring results for whole process.

The lower part includes components of DTMA, User Domain Kernel OS_{un} and user application component P_n. When trust is delivered to DTMA, DTMA will start new $DomU_i$ and dynamically monitor the behavior of User Kernel OS_{ui} and User Component P_i. If sensitive behavior is triggered, security attributes of all components will be measured and the results will be stored.

DTMA is a key component of DTSTM. It measures trustworthiness of the user domain from system domain and ensures security of the measurement module itself. DTMA mainly consists of three sub-modules:

(1) MA (Measurement Agent). MA measures integrity of the user components during booting stage of a virtual machine, monitors behavior of the components during runtime stage and conducts trust measurement for relevant components in case of sensitive operation and update of components.

(2) SS (Secure Storage). SS provides secure storage service. To collect trustable evidences for software, it is inevitable to store and transfer evidences. The acquired evidence can be effectively protected using secure storage mechanism of TPM. SS aims to maintain the operating environment of TPM for each virtual machine including virtual machine relevant keys and data, physical PCR bound to each virtual machine and each virtual machine relevant Virtual PCR (VPCR). As for acquired trustable evidence of software, it will be signed by TPM thus verifying the trustworthiness of its source.

(3) CM (Component Manager). CM manages installation, registeration, deletion, unload, upgrade, etc, and stores certificate attributes of the registered components. CM aims to deal with update requests of components in the virtual machine. If the security of the component changes, it will trigger MA to measure the components again. CM uses a component list to store register information and attribute certificate information of components. The same component is shared by multiple virtual machines. If the component is updated, CM will modify the item of component list.

3.3 Workflow

When measuring user application components, it is necessary for DTSTM to measure trustworthiness of the relevant components. All components except root component have a parent component and all component logics form a tree structure, as shown in **Fig. 3**. When a virtual machine is built on the trusted cloud computing platform, SS will establish a VPCR for this virtual machine to describe configuration and status of associated components while CM will set up a component tree which corresponds to this context. When a virtual machine instance executes, CM will compute the component tree according to the relevant components. Moreover, when a component is updated or the status of a component is changed due to hostile attack, MA will be activated to measure the component tree and update the measurement results.

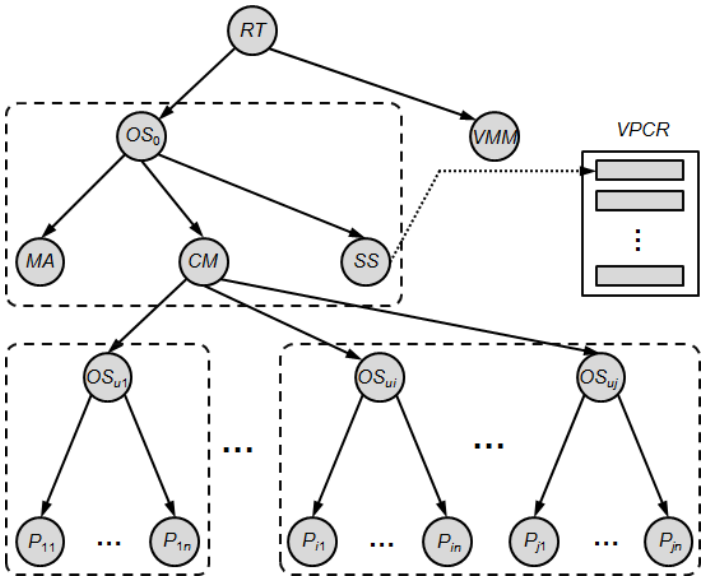


Fig. 3. The structure of component tree

The measurement process mainly includes construction and measurement of component tree, which is described as follows:

(1) Construction of component tree. The component tree uses a list structure *Nodes* for storage, with each node structure consisting of component identification (*id*), component level (*level*) , child node pointer array (*children*) and parent node pointer (*parent*). The *Nodes* is a global variable. When a node is created, we assign an *id* to it. The *id* is a number that uniquely identifies a node. The *level* of root component is 0., the *level* of the child node will increase by 1 on the basis of its parent's level when it is created. The *children* point to all child nodes which originate from a node. The *parent* point to the node of its parent. The construction algorithm of component tree is shown in **Algorithm 1**.

Algorithm 1. Construction Algorithm of Component Tree

Input : *Nodes* - array[0...*N*−1] of nodes
Output: *Root* - Root of the component tree

```
1.  BuildComponentTree (Nodes)
2.      Sort the points in ascending order of level for Nodes
3.      for each p∈ Nodes in ascending order of level do
4.          curNode := GetNode(p)
5.          relateNodes := GetRelateNode(p)
6.          for each q∈ relateNodes in increasing order of level do
7.              adjNode := GetNode(q)
8.              parNode := GetParentNode(q)
9.              if (Nodes[curNode]→level > Nodes[adjNode]→level) then
10.                 Nodes[parNode]→addChild(Nodes[curNode])
11.                 Nodes[curNode]→level := Nodes[adjNode]→level
12.                 Nodes[curNode]→parent := Nodes[adjNode]→ parent
13.             else
14.                 SetRelateNode(adjNode, curNode)
15.             end if
16.         end for
17.     end for
18.     Root := GetLowestNode(p)
19. return Root
```

Input of the algorithm is the component node array (*Nodes*) and the result is root of the component tree (*Root*). Firstly, arrange node array of the component in an ascending sequence according to its node level (Line 2), and then select one node successively. The *GetRelateNode* function is invoked to get a node's relevant node array (Lines 3 - 5). Compare *level* of the relevant nodes with that of the node to be inserted, and if the latter is higher, transfer the node to be the neighbor node of the relevant nodes (Lines 6 - 12), or else, establish relationship between this node with its relevant nodes (Line 14). As shown in **Fig. 4**, level of the relevant node N_i^0 is lower than that of N_{i+1}^m, so insert the node below N_{i-1}^j, and adjust its level to i. Lastly, select the node with lowest level from node array as root of component tree, and return (Lines 18 - 19).

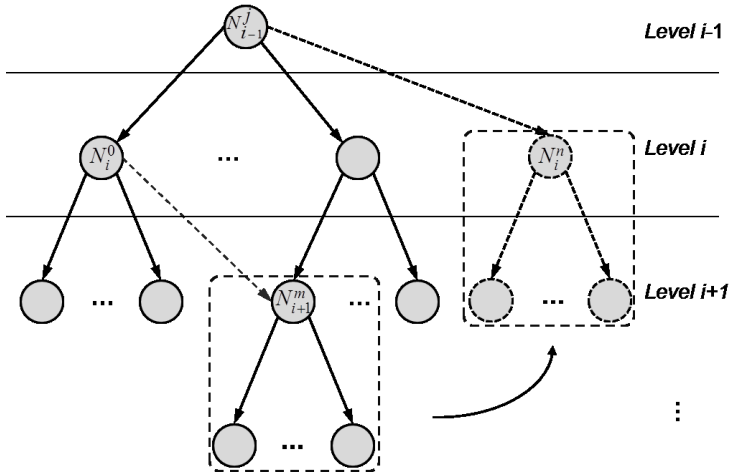


Fig. 4. The construction of component tree

(2) Measurement of component tree. No matter a component update is triggered by a normal user or a hostile attack, the trust of the computing environment alters. DTSTM conducts real-time monitoring of changes of the components. When the status of a component changes, the measurement algorithm of component tree will be invoked to verify its trustworthiness. The measurement algorithm of the component tree is shown in **Algorithm 2**.

Algorithm 2. Measurement Algorithm of the Component Tree

Input : *Nodes* - array[0...*N*−1] of nodes, *mNode* – measured node
Output: *Trust* – trust of *monNode*

```
1.  MeasureComponent (Nodes, mNode)
2.      rootNode := GetRootNode(mNode)
3.      childNodes := GetChildNode(rootNode)
4.      Sort the points in ascending order of level for childNodes
5.      for each p∈childNodes in ascending order of level do
6.          adjNode := GetNode(p)
7.          if (Nodes[adjNode]→trust ≠ TRUE) then
8.              mLevel:= GetLevelNode(mNode)
9.              if (Nodes[adjNode]→level < mLevel) then
10.                 b := GetBehave(adjNode)
11.                 Nodes[adjNode]→trust := H(adjNode)∈E_n && (F(b)==True)
12.                 SetRelateTrust(adjNode, Nodes[adjNode]→trust)
13.             end if
14.         end if
15.     end for
16.     Trust := Nodes[mNode]→trust
17. return Trust
```

Input of the algorithm is the component node array (*Nodes*) and the measurement node (*mNode*). Output is the trustworthiness of the measurement node. Firstly, acquire the root node and all child nodes below the root node, and arrange these nodes in an ascending sequence according to their levels (Lines 2-4). Next, select the node's child node successively to check its trustworthiness. If the node is not trusted and its level is lower than the measurement node's level (Lines 8 and 9), measure whether the subject and behavior of the child node are both trusted (Line 11) according to the validation theorem and update trustworthiness of the relevant nodes (Line 12). Lastly, return the measured result.

An example is shown in **Fig. 5**. When N_{i+1}^m is altered, firstly, measure trustworthiness of the node. Secondly, trace back to bottom node N_{i-1}^j through parent node N_i^l; and then, measure trustworthiness of N_i^0 and N_i^n respectively. In case of any change, update trustworthiness of N_{i-1}^j, N_i^l and N_{i+1}^m during the backtracking process. Lastly, return the measured result.

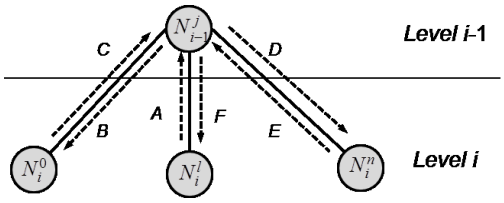


Fig. 5. The measurement of component tree

3.4 Security Analysis

This section will further make a quantitative security evaluation of DTSTM. For the purpose of formal description of the model, components related to DTSTM are defined in **Table 1**:

Table 1. Main elements

Notation	Element and implication
S	S={p|p is any component}
S^{U}	S^{U}={p|p is any untrusted component}
S^{T}	S^{T}={p|p is any trusted component}=$S-S^{\mathrm{U}}$
S^{M}	S^{M} ={p|$p\in S^{\mathrm{U}}$ and contains malicious codes}
S^{I}	S^{I}={p|$p\in S^{\mathrm{U}}$ and does not contain malicious codes}=$S^{\mathrm{U}}-S^{\mathrm{M}}$
S^{V}	S^{V}={p|$p\in S^{\mathrm{T}}$ and contains vulnerable codes}
S^{S}	S^{S}={p|$p\in S^{\mathrm{T}}$ and does not contain vulnerable codes}=$S^{\mathrm{T}}-S^{\mathrm{V}}$
S_{env}	Sen_v={p|p is any component operating in env}, $env\in ENV$={RT, OS_0, OS_{U}}
$P(p)$	$p\in \mathrm{S}_{\mathrm{env}}$: probability that p damages the security of its environment (env)
$P^{\mathrm{M}}(p)$	$p\in \mathrm{S}_{\mathrm{env}}$: probability that p contains malicious codes in env
$P^{\mathrm{V}}(p)$	$p\in \mathrm{S}_{\mathrm{env}}$: probability that p contains vulnerable codes in env
$Size(p)$	Source codes lines of component p, indicating the scale of p

Note: CRTM, BIOS, BootLoader and VMM is simplificated as RT; OS_0 means kernel of system domain and OS_{U} means kernel of user domain.

The following formulas are based on these expressions:

$$S = S^T \cup S^U = (S^V \cup S^S) \cup (S^M \cup S^I) \tag{1}$$

$$P(p) = P^M(p) + P^V(p) \tag{2}$$

Based on Formula (1) and Formula (2), we obtain that as components in operating environment *env* grow in number, the security of *env* decreases gradually. This conclusion can be expressed with the formula below:

$$P(S'_{env}) = \sum_{p \in S'_{env}} P(p) < P(S''_{env}) = \sum_{p \in S''_{env}} P(p), S'_{env} \subset S''_{env} \tag{3}$$

Sam *et al*. [23] proposed that if software design quality, code complexity and code implementation quality are almost equal, the vulnerability of software p is approximately in direct proportion to *Size* (p) and a security vulnerability remains uncovered every one thousand lines of codes in average. Therefore, $P^{V}(p)$ can be simplified as Formula (4) below, where α is an empirical constant.

$$P^{V}(p) = \alpha \times \frac{Size(p)}{\sum_{p_i \in S_{env}} Size(p_i)}, p \in S_{env} \tag{4}$$

As for traditional multi-task operating system (OS), $P(S_{OS})$ can be expressed as the following formula:

$$\begin{aligned} P(S_{OS}) &= P(S^{T}_{OS} \cup S^{U}_{OS}) = P(S^{T}_{OS}) + P(S^{U}_{OS}) \\ &= P(S^{S}_{OS}) + P(S^{V}_{OS}) + P(S^{M}_{OS}) + P(S^{I}_{OS}) \\ &= P(S^{V}_{OS}) + P(S^{M}_{OS}) + P(S^{I}_{OS}) \end{aligned} \tag{5}$$

As all components in definition S^{S}_{OS} are trusted and do not contain any vulnerable codes, $P(S^{S}_{OS})$ is omitted in the final results of Formula (5).

The formulas below point out aspects regarding the security of user domain and system domain in DTSTM:

$$P(S_{OS_U}) = P(S^{U}_{OS_U}) = P(S^{M}_{OS_U}) + P(S^{I}_{OS_U}) \tag{6}$$

User domain which consists of various application components may contain malicious codes and thus is untrusted, as shown in Formula (6).

$$P(S_{OS_0}) = P(S^{T}_{OS_0}) = P(S^{V}_{OS_0}) + P(S^{S}_{OS_0}) = P(S^{V}_{OS_0}) \tag{7}$$

The system domain of DTSTM only contains verified management components. Most verified codes of the componets do not contain vulnerabilities, so the security of the system domain mainly depends on the components which suffer from vulnerabilities, as shown in Formula (7).

In system operating environment, only DTSTM RT, system domain Dom0 exchange data with other user domains, and following formula can be obtained:

$$S^{V}_{OS_0} = S_{RT} \cup S_{Dom0}, Size(RT) + Size(Dom0) << Size(OS) \tag{8}$$

As DTSTM software is much smaller than a traditional OS in scale, it is superior in reliability. Based on the Formulas (3), (4), (7) and (8), the following inequality can be obtained:

$$P(S_{OS_0}) = P(S_{RT}) + P(S_{Dom0}) << P(S_{OS}) < 1 \tag{9}$$

The focus of DTSTM is the security of system domain. The probability that isolated codes in DTSTM virtual machine lower the security of system domain can be depicted as follows:

$$P(S_{OS_U} \mid S_{OS_0}) = P(S_{Dom_U}) \times P(S_{OS_0}) \cong P(S_{OS}) \times P(S_{OS_0}) \tag{10}$$

$P(S_{OS_U} \mid S_{OS_0})$ indicates that components in S_{OS_U} break through the security protection of OS_U, OS_0 simultaneously. Based on the Formula (9) and Formula (10), we obtain

$$P(S_{OS_U} \mid S_{OS_0}) << P(S_{OS}) \tag{11}$$

From Inequality (11), it can be seen that DTSTM significantly improves the security of system domain.

4. Implementation and Evaluation

We implement a DTSTM-based system for virtual machine trust measurement on the basis of VMM-level SCI [24] and VMI [25]. The effectiveness of the trust measurement system is verified with four groups of malicious code samples, and the measurement overhead is evaluated with lmbench test set.

4.1 System Overview

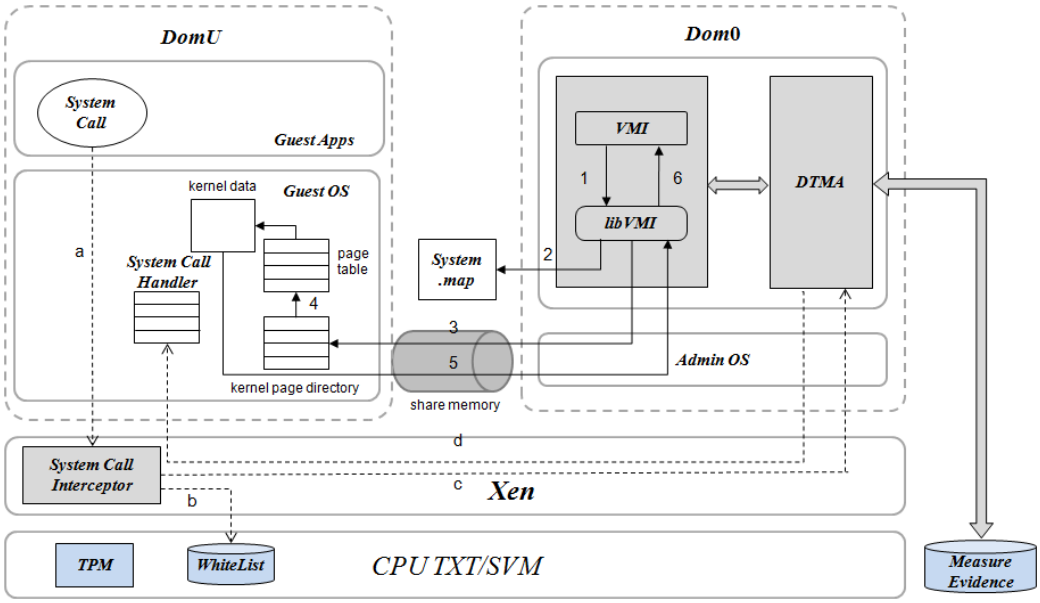


Fig. 6. DTSTM-based trust measurement system architecture

Fig. 6. shows the architecture of DTSTM-based system for trust measurement. SCI module intercepts system call events in user domain, reconstructs system call sequence related to the loading of the executable program, and transfers the sequence to DTMA module after comparing it with the white list. DTMA module acquires behavior of the virtual machine based on system call sequence and system call context extracted by VMI module. It verifies whether the called subject file is complete and whether the behavior accords with the anticipated policy. The verification results will be saved to provide trust measurement evidence for user.

4.1.1. SCI Module

In modern OS, user program accesses kernel through system call. When user program initiates a system call, it will first move system call parameters to the relevant register and then

execute the interrupt instruction to trap into kernel. However, in a virtualization environment, VMM must intercept the system calls before measuring user behavior.

In [24], hardware virtualization technology is used to intercept system calls at VMM level. We, however, intercept system calls of virtual machine on the basis of [24] and using the white list technology. In DTSTM, SCI is located at Virtual Machine Monitor (VMM) and responsible for intercepting the system calls. It acquires system call events, maintains a white list for system call according to measurement requirements, and matchs system call numbers with the white list to identify whether it is necessary to measure the behavior. The main working steps are listed as follows:

1) SCI turns off direct call of *Dom*U system call, and *Dom*U traps into VMM level when initiating a system call. After intercepting the system call, SCI will set entry address of the call as an illegal address and return to *Dom*U;

2) When executing codes at this address, *Dom*U will generate a page fault, thus initiate vmexit instructions and trap into VMM level again. SCI will intercept this VMM exception handling process;

3) SCI can acquire system call numbers according to the values of EAX and match them with the white list. If the measurement is necessary, it will send the numbers to MA. MA will identify the behavior by matching the system call IDs and the context;

4) After the execution, entry address of the original system call will be recovered for normal execution of the procedure.

Through the above processes, DTSTM is able to monitor the behaviors in user domain such as load of software module, file read and write, call of sensitive operations and realize measurement of user domain through analysis of their behaviors.

4.1.2. VMI Module

In order to ensure the security of measurement module itself and the authenticity of key data, we use VMI to acquire data of user domain. In this way, the measurement module is independent of the security of user domain kernel.

When MA takes control of the system calls, VMI module acquires contextual information of the system calls from user domain. As there is a semantic gap between user domain and system domain, VMI adopts LibVMI [26] to access any physical memory location of the user so as to realize introspection of user space data. LibVMI runs in *Dom*0 and accesses the original data of memory of user domain through XenControl.

MA module in *Dom*0 makes use of VMI to acquire the memory data of *Dom*U through following six steps:

1) VMI makes requests to acquire key data of user and kernel spaces;

2) LibVMI finds the requested kernel address and virtual address through System map;

3) VMM maps the kernel page directory (KPD) to the memory space of *Dom*0 and makes use of KPD to find the correct page table (PT);

4) VMM maps PT to the memory space of *Dom*0 and finds the correct address of data page in PT;

5) VMM maps the data page to the memory address space of *Dom*0 and returns it to the LibVMI library;

6) LibVMI returns the pointer and offset of the data sheet with read/write permissions to VMI.

Through the above procedures, MA module can acquire data structures of system kernel symbols in *Dom*0, e.g. task_struct. Based on the data structures, MA module acquires the process list, module list and other information to judge whether there is any malicious behavior.

4.2 System Evaluation

The security analysis of DTSTM in Section 3.4 shows that this model is superior to traditional chain style model in security. In this section, we verify effectiveness and overhead of our system.

Our prototype system runs on a 2.8GHz Intel Core i5 processor with 4GB memory and a 500G 7200RPM Seagate hard disc. The system is based on a Xen virtualization platform, consisting of tboot1.7.3 (for bootloader), Xen4.1.4 (for VMM) and Linux3.2.0 kernel (for system domain kernel), with both WindowsXP and Ubuntu12.04 virtual machines installed on Xen.

4.2.1 Effectiveness

In order to demonstrate the advantages of DTSTM in dynamic measurement, we compare it with Xen and IMA [15]. According to the supported OS and code type, four representative malicious sofewares are selected, i.e. poisonivy-rat [27], hxdef [28], lrk5 [29] and adore-ng [30]. The software samples are shown in **Table 2**.

Table 2. Samples for effectiveness test

Sample	Supported OS	Type	Binary Size(KB)
poisonivy-rat	Windows	backdoor	2,092
hxdef	Windows	rootkit	68
lrk5	Linux	backdoor	3,223
adore-ng	Linux	rootkit	21

The results of effectiveness test on DTSTM are shown in **Table 3**, where "√" indicates that malicious codes are detectable while "—", undetectable.

Table 3. Effectiveness test results

Sample	poisonivy-rat		hxdef		lrk5		adore-ng	
	Booting	Runtime	Booting	Runtime	Booting	Runtime	Booting	Runtime
Xen	—	—	—	—	—	—	—	—
IMA	√	—	√	—	√	—	√	—
DTSTM	√	√	√	√	√	√	√	√

1) User-level programs. Poisonivy-rat and lrk5, two popular backdoors, are used to verify effectiveness of DTSTM by measuring the integrity of application programs in Windows and Linux environments. Poisonivy-rat and lrk5, realize self-starting by tempering with normal programs, and accept remote connection when executed. In the experiment, Xen does not perform trust verification and thus can not discover all samples. IMA can not detect malicious codes executed in runtime stage though it is able to discover samples in booting stage. DTSTM, however, discovers loading of malicious programs and measures their integrity, and then identifies the programs are malicious by comparing them with the existing fingerprints in fingerprint database, thus directly terminates the loading process.

2) Kernel-level programs. Attacks usually invade kernel of the OS by loading malicious kernel module. We adopt hxdef and adore-ng, two well-known kernel-level rootkits of

Windows and Linux, to verify effectiveness of DTSTM kernel module. As in user application program experiments, Xen can not detect malicious codes and IMA can only detect malicious codes during booting of the system, while DTSTM can detect samples during runtime. We add hxdef and adore-ng to white list of DTSTM and load them to kernel. In the experiment, hxdef and adore-ng are detected by DTSTM as soon as they maliciously temper with kernel module after booting, which indicates DTSTM can detect attacks on kernel module both effectively and timely.

4.2.2 Performance

In order to evaluate the influence of DTSTM on system performance, we respectively test running of system calls and application programs.

1) System call test. We adopt lmbench as the test set. As DTSTM is implemented based on system call interception and analysis, test indexes associated with system calls are specifically selected from lembench and compared with standard Xen to analyze the performance overhead introduced by DTSTM. **Table 4** gives corresponding test results.

Table 4. System call test results (us)

Micro Benchmark	null	open/close	read	write	fork+exec
Xen	0.82	21.82	2.19	1.24	2303
DTSTM	1.23	23.46	2.62	2.28	4378

It is found that DTSTM has little influence on open/close and read/write operations, but the performance overhead is obvious under fork+exec test indexes. The fork+exec consists of three steps, namely system call interception, program path position and program file measurement, and the process of computing hash values of executable files takes a long time. The required time for DTSTM to measure files is almost two times as much as that of standard Xen. However, hash values of executable files are usually computed only once before running and fork+exec operations hold a relatively low proportion in daily programs, and DTSTM can ensure integrity of the programs compared with Xen, the overhead is acceptable.

2) Application program test. In order to further evaluate the performance of DTSTM, performance overhead of 6 common application programs are tested, as shown in **Table 5**:

Table 5. Samples for application program test

Item	Command
Getpid	getpid-2000.sh
Decompression	tar zxf linux-source.tar.bz2
Compression	tar zcf linux-source
File Copy	cp /usr/linux-source /linux-source
Kernel Build	make
Linux Boot	Linux boot

Fig. 7 shows the application program test results of Xen and DTSTM. From the point of overall trend, DTSTM exerts some influence on system performance. The main reason is DTSTM needs to intercept system calls at upper level by forcing a page fault to make current execution flow trap into VMM level.

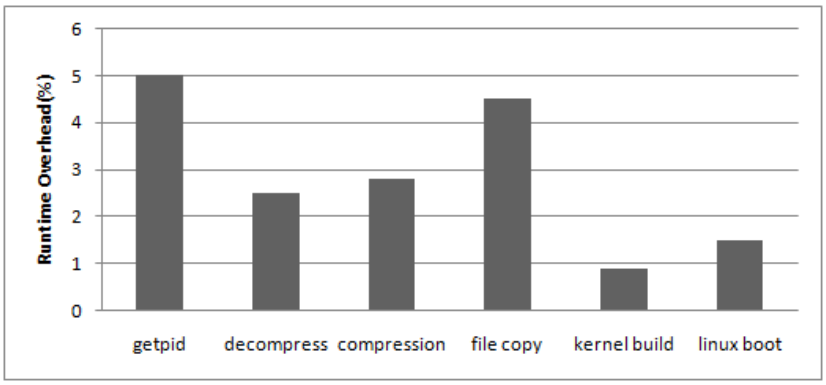


Fig. 7. Application program test results of DTSTM

As DTSTM needs to intercept system calls, the performance overhead of getpid test program is relatively high. For compression and decompression programs which refer to computation-intensive work, DTSTM only introduces 2.7% performance overhead. The file copy requires a relatively large system overhead because it is I/O-intensive and includes a lot of read and write operations and DTSTM intercepts write system calls to prevent modification of loaded software files. The kernel compilation, though taking a long time, brings about a relatively small system overhead as it calls few types of compilation programs. All executable programs of DTSTM verification will be loaded during booting of Linux, so the boot test performs general overhead of DTSTM. It can be seen from the figure that the performance overhead of DTSTM is approximately 1.5%, a relatively low result.

5. Related Work

Trust measurement of virtual machine is considered as one of the key challenges in cloud computing security. Santos et al [4] proposed a trusted cloud computing platform (TCCP) which provides a closed box execution environment to ensure confidentiality of the running virtual machine and allows the users to attest to the IaaS provider and determine whether or not the service is secure before they launch their virtual machines. TCCP has to measure all loaded modules such as configuration information of hardware and software of the platform, thus generating redundancy [31].

The measurement method proposed in [9] is based on platform attribute to reduce redundancy, but the platform attribute is an abstract concept which is always difficult to describe and define. DTSTM, however, uses behavior trace to describe the status of virtual machine, which is accurate and easy to implement.

Chain style measurement proposed by TCG is applicable to traditional PC platform, with no consideration for virtualization environment. vTPM [11] virtualizes entity TPM and constructs multiple trust chains above VMM layer. The method can be considered as a simple extension on the basis of chain measurement. However, it can not solve the relevance problem of trust relationship between virtual machines in cloud computing environment.

ETPM [23] uses a root node which located at central position to measure other nodes. The star style measurement is complex and poor extensibility, though it improves measurement security. DTSTM, however, adopts tree style measurement according to the characteristics of cloud computing, which improves expansibility and concurrency of measurement.

BonaFides [7] periodically measures integrity of the virtual machine kernel and files. This kind of measurement system lacks flexibility, as its measurement timing is determined by

system designer rather than the actual user. If virtual machine is damaged after measurement, the measurement will give wrong results. DTSTM, however, intercepts system calls in real time, thus preventing security problems caused by inconsistency of measurement and running.

6. Conclusions and Future Work

The main approach of implementing secure and trusted cloud computing is to effectively create a trusted virtualization environment, and ensure trustworthiness of the software in virtual machine. In view of the characteristics and requirements in cloud computing environment, we propose a model named DTSTM, which reduces complexity of trust measurement by separating system domain and user domain with different methods. DTSTM solves the problems of traditional measurement schemes in terms of dynamism, security and concurrency, and improves practicability and expandability simutaneously. Moreover, experimental results indicate practicability and high performance of the model.

There are two main directions to which our measurement model can be extended. First, it is necessary to prove the trusted status of platform to user after platform measurement of virtual machine. Traditional attestation scheme mainly aims at a single host machine while a large number of virtual machines are included in cloud computing system, so there are such problems as single point failure and low attestation efficiency. For this reason, it is important to improve the efficiency and security of the attestation. Second, it is crucial to discover security policy violations in the model. In case of violations among these security policy, trustworthiness of the software can not be judged. Therefore, it also requires in-depth study to effectively implement policy violation discovery in cloud computing infrastructure.

References

[1] Top Threats to Cloud Computing. http://www.cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

[2] Y. Chen, V. Paxson and R. H. Katz, "What's New About Cloud Computing Security," *University of California, Berkeley, Tech*, vol. 20, January, 2010. Article (CrossRef Link)

[3] D. G. Feng, M. Zhang, Y. Zhang and Z Xu, "Study on cloud computing security," *Journal of Software*, vol. 22, no. 1, pp. 71-83, January, 2011. Article (CrossRef Link)

[4] N. Santos, K. Gummadi and R. Rodrigues, "Towards trusted cloud computing," in *Proc. of the 2009 conference on Hot topics in cloud computing*, September, 2009. Article (CrossRef Link)

[5] J. K. Frank, "Private virtual infrastructure for cloud computing," in *Proc. of the 2009 conference on Hot topics in cloud computing*, September, 2009. Article (CrossRef Link)

[6] J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger and P. McDaniel, "Seeding clouds with trust anchors," in *Proc. of the 2010 ACM workshop on Cloud computing security workshop*, pp. 43-46, October, 2010. Article (CrossRef Link)

[7] R. Neisse, D. Holling and A. Pretschner, "Implementing trust in cloud infrastructures," in *proc. of 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 524-533 , May, 2011. Article (CrossRef Link)

[8] S. Butt, C. H. Lagar, A. Srivastava and V. Ganapathy, "Self-service cloud computing," in *Proc. of the 2012 ACM conference on Computer and communications security*, pp. 253-264, October, 2012. Article (CrossRef Link)

[9] N. Santos, R. Rodrigues, K. Gummadi and S. Saroiu, "Policy-sealed data: A new abstraction for building trusted cloud services," in *Proc. of the 2012 USENIX Security*, August, 2012. Article (CrossRef Link)

[10] TCG Specification Architecture Overview. https://www.trustedcomputinggroup.org/

[11] R. Perez, R. Sailer and L. Van-Doorn, "vTPM: Virtualizing the Trusted Platform Module," in *Proc. of the 15th USENIX Security Symposium*, pp. 305-320, July, 2006. Article (CrossRef Link)
[12] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum and D. Boneh, "Terra: A Virtual Machine-Based Platform for Trusted Computing," *ACM SIGOPS Operating System Review*, vol. 37, no. 5, pp. 193-206, October, 2003. Article (CrossRef Link)
[13] E. Shi, A. Perrig and L. V. Doorn, "BIND: A Fine-Grained Attestation Service for Secure Distributed Systems," in *Proc. of the 2005 IEEE Symposium on Security and Privacy*, pp. 154-168, May, 2005. Article (CrossRef Link)
[14] S. Berger, R. Caceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer and D. Srinivasan, "TVDc: managing security in the trusted virtual datacenter," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 1, pp. 40-47, January, 2008. Article (CrossRef Link)
[15] S. Reiner, X. L. Zhang, T. Jaeger and L. Van-Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *Proc. of the 13th USENIX Security Symposium*, pp. 16-32, August, 2004. Article (CrossRef Link)
[16] A. Sadeghi and C. Stble, "Property-based attestation for computing platforms: caring about properties, not mechanisms," in *Proc. of the 2004 workshop on New security paradigms*, pp. 67-77, September, 2004. Article (CrossRef Link)
[17] L. Chen, R. Landfermann, H. Loehr, M. Rohe, A. Sadeghi and C. Stble, "A Protocol for Property-Based Attestation," in *Proc. of the 1st ACM Workshop on Scalable Trusted Computing*, pp. 7-16, November, 2006. Article (CrossRef Link)
[18] J. McCune, B. Parno, A. Perrig, M. Reiter and H. Isozaki, "Flicker: An Execution Infrastructure for TCB Minimization," *ACM SIGOPS Operation System Review*, vol. 42, no. 4, pp. 315-328, April, 2008. Article (CrossRef Link)
[19] J. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor and A. Perrig, "TrustVisor: Efficient TCB Reduction and Attestation," in *Proc. of the 2010 IEEE Symposium on Security and Privacy*, pp. 143-158, May, 2010. Article (CrossRef Link)
[20] D. G. Feng and Y. Qin, "Research on Attestation Method for Trust Computing Environment," *Chinese Journal of Computers*, vol. 31, no. 9, pp. 1640-1652, September, 2008. Article (CrossRef Link)
[21] N. Petroni and T. Fraser, "Copilot-A coprocessor-based kernel runtime integrity monitor," in *Proc. of the 13th conference on USENIX Security Symposium*, pp. 179-194, August, 2004. Article (CrossRef Link)
[22] B. Zhao, H. G. Zhang, J. Li and S. Wen, "The system architecture and security structure of trusted PDA," *Chinese Journal of Computers*, vol. 31, no.1, pp. 82-93, January, 2010. Article (CrossRef Link)
[23] W. Sam, A. Paul and P. Amit, "A software flaw taxonomy: aiming tools at security," in *Proc. of the 2005 workshop on Software Engineering for secure system*, pp. 1-7, January, 2005. Article (CrossRef Link)
[24] A. Dinaburg, P. Royal, M. Sharif and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proc. of the 15th ACM conference on Computer and Communication Security*, pp. 51-62, October, 2008. Article (CrossRef Link)
[25] T. Garfinkel and M. Rosemblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," in *Proc. of the 2003 Network and Distributed Systems Security Symposium*, pp. 191-206, February, 2003. Article (CrossRef Link)
[26] LibVMI. https://code.google.com/p/vmitools/
[27] Poison ivy – remote administration tool. http://www.poisonivy-rat.com/
[28] Hacker defender. http://en.pudn.com/download46/sourcecode/hack/detail154363_en.html
[29] Linux rootkit 5. http://www.ussrback.com/UNIX/penetration/rootkits/
[30] Adore-ng rootkit. http://stealth.openwall.net/rootkits/
[31] T. Jaeger, R. Sailer and U. Shankar, "PRIMA: policy-reduced integrity measurement architecture," in *Proc. of the 11th ACM Symposium on Access Control Models*, pp.19-28, June, 2005. Article (CrossRef Link)\

Appendix A – Abbreviation

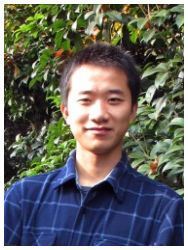
BIOS	Basic Input Output System
CM	Component Manager
CRTM	Core Root of Trust for Measurement
DTMA	Dynamic Trusted Measurement Agent
DTSTM	Dynamic Tree Style Trust Measurement Model
IMA	Integrity Measurement Architecture
MA	Measurement Agent
OS	Operating System
PCR	Platform Configuration Registers
SCI	System Call Interceptor
SS	Secure Storage
TCCP	Trusted Cloud Computing Platform
TCG	Trusted Computing Group
TPM	Trusted Platform Module
VMI	Virtual Machine Introspection
VMM	Virtual Machine Monitor
VPCR	Virtual PCR

Zhen-ji Zhou was born in Shuyang, Jiangsu, China in 1985. He received his B.E. and the M.S degree in computer science and technology from PLA University of Science and Technology in 2008 and 2010 respectively. Now he is a Ph.D. candidate in the university. His research interests include information security and cloud security.

Li-fa Wu was born in Qichun, Hubei, China in 1968. He received his Ph.D. from Nanjing University in 1998. He is currently a professor in PLA University of Science and Technology. His research fields concern network security, protocol engineering and satellite communication.

Zheng Hong was born in Yingtan, Jiangxi, China in 1979. He received his Ph.D. from PLA University of Science and Technology in 2007. Now he is an associate professor in the university. His research fields concern network security and protocol reverse engineering.

Ming-fei Xu was born in Jining, Shandong, China in 1989. He received his B.E. from AnHui University of Science and Technology in 2011. Now he is a M.S candidate in PLA University of Science and Technology. His research interests include network security and trusted computing.



Fan Pan was born in Wuhu, Anhui, China in 1987. He received his B.E. and the M.S degree in network engineering from PLA University of Science and Technology in 2007 and 2009 respectively. Now he is a Ph.D. candidate in the university. His research interests include protocol reverse engineering and software testing.