11-2008

Model-Driven Remote Attestation: Attesting Remote System from Behavioral Aspect

Liang GU
Peking University

Xuhua DING
Singapore Management University, xhding@smu.edu.sg

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Yanzhen ZOU
Peking University

Bing XIE
Peking University

See next page for additional authors

DOI: https://doi.org/10.1109/ICYCS.2008.349

Citation

GU, Liang; DING, Xuhua; DENG, Robert H.; ZOU, Yanzhen; XIE, Bing; SHAO, Weizhong; and MEI, Hong. Model-Driven Remote Attestation: Attesting Remote System from Behavioral Aspect. (2008). *2008 ICYCS 9th International Conference for Young Computer Scientists: November 18-21, 2008, Zhang Jia Jie, Hunan, China: Proceedings*. 2347-2353. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/314

Author
Liang GU, Xuhua DING, Robert H. DENG, Yanzhen ZOU, Bing XIE, Weizhong SHAO, and Hong MEI

Model–Driven Remote Attestation: Attesting Remote System from Behavioral Aspect

Liang Gu[1,2], Xuhua Ding[2], Robert H. Deng[2], Yanzhen Zou[1], Bing Xie[1], Weizhong Shao[1], Hong Mei[1]
[1] Key Laboratory of High Confidence Software Technologies, Peking University,
{guliang05, zouyz, xiebing, wzshao, meih}@sei.pku.edu.cn
[2] School of Information Systems, Singapore Management University,
{xhding, robertdeng}@smu.edu.sg

Abstract

Remote attestation was introduced in TCG specifications to determine whether a remote system is trusted to behave in a particular manner for a specific purpose; however, most of the existing approaches attest only the integrity state of a remote system and hence have a long way to go in achieving the above attestation objective. Behavior–based attestation and semantic attestation were recently introduced as solutions to approach the TCG attestation objective. In this paper, we extend behavior–based attestation to a model–driven remote attestation to prove that a remote system is trusted as defined by TCG. Our model–driven remote attestation verifies two compliance requirements to prove the trustworthiness of a remote system: expected behavior compliance and enforced behavior compliance.

Keywords: *Trusted computing, remote attestation, security policy*

1 Introduction

Remote attestation was first introduced as an important feature in trusted computing and supposed to attest the trustworthy of remote platform configurations. A trusted-platform device attests its state by reporting its integrity state, for example, the values in the registers inside the Trusted Platform Module (TPM) chip [1]. This feature enables a challenger to have certain confidence about the integrity state of the remote platform. The TCG specifications introduce a layered integrity measurement mechanism to measure the platform integrity from hardware to applications and authenticate remote platform to the challenger based on the primitive attestation function provided by TPM. With the *authenticated boot* or *secure boot* introduced in TCG specifications, a platform can prove the genuineness of both hardware and software at the system initialization stage. It provides a primitive attestation service to support higher level remote attestation mechanisms, e.g., remote attestation of applications. Several remote attestation mechanisms have been proposed based on this primitive attestation service in trusted computing, e.g., [2, 3, 4, 5, 6].

However, research in remote attestation is still in its infancy and many technical challenges need to be overcome. The first challenge is closing the gap between existing remote attestation solutions and TCG's objective. In TCG specifications, trust is defined as "the expectation that a device will behave in a particular manner for a specific purpose". In other words, for the purpose of proving that remote platforms or programs are trustworthy, remote attestation is required to prove that these targets behave as expected. However, existing remote attestation solutions mostly check only the configuration or integrity state of remote platforms or programs. These solutions can not achieve the attestation objective as specified in TCG's trust definition. The second problem is that most of the exiting attestation mechanisms have not specifically considered how to attest programs at runtime in a dynamic environment effectively and efficiently. This is because the runtime environments of programs are dynamic and difficult to predict. In TCG's primitive integrity report mechanism, TPM simply measures and records all configuration states of the platform. However, the specific semantic meanings of these giant numbers of configuration states have not been specifically studied. It is still not clear how to translate these system configurations into a trust decision in remote attestation [7]. The question is how challengers can employ the existing lower level attestation schemes to prove that a remote target behaves as expected.

In a typical modern operating system, all operations can be restricted by system policies. Some existing operating systems employ discretionary access control to protect sys-

978-0-7695-3398-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ICYCS.2008.349

tem resources. Recently, some works [8, 9, 10] tried to introduce *Mandatory Access Control* (MAC) to restrict system behaviors. These policies specify how subjects should behave and how objects may be accessed in an operating system. The behavior restrictions in a policy represent the system behavior expectation of policy producer. Today, most systems are under the control of their users or system administrators. Besides modifications made by system users, system compromise may also lead to changes of system security policy. SElinux [11] is a typical Mandatory Access Control implementation which supports dynamic polices. For the purpose of attesting remote systems, it is necessary for challengers to attest both real time system behaviors and their system access control policy enforcement. For different policy enforcement mechanisms, a specific behavior model can have different implementations. In a specific application, a challenger may only need to know whether the behavior model of attesting target satisfies their attestation objective, leaving the policy implementation details verification in an automatic and transparent approach at lower level.

Haldar et al. [4] proposed the concept of semantic remote attestation and Li et al. [12] proposed the concept of behavior based attestation. These approaches initiated the trend of attesting remote system from a behavioral aspect. However, these approaches are still at a primitive level. Haldar et al. [4] did not specify how to practically attest software system behaviors. Li et al. [12] only dealt with static policy and just attested system behaviors records. We extend the behavior based attestation and semantic remote attestation to a more practical approach: model-driven remote attestation, which attests the system behaviors and system policy enforcement mechanism to prove whether a remote target behaves as expected. By extracting behavior model from enforced policy or generating policy instances of expected behavior model, the challenger verifies whether the behavior model dictated by an enforced policy is trustworthy. By building an attestation service to monitor and record the dynamic enforcement of policy and system behaviors, the challenger verifies these measurements to check whether the expected behaviors in the enforced policy are correctly executed. Our model–driven attestation closes the gap of attestation objectives between existing attestation schemes and final expectation of TCG specification, and can also be applied on various platforms.

2 Background

2.1 Trusted Computing

TCG has established a serial of specifications about the trusted computing [1]. There are some upcoming technologies of trusted computing, e.g., AMD's Secure Virtual Machine (SVM) architecture [13] and Intel's Trusted Execution Technology(TXT) [14]. The trusted computing requires following core features:

- Trust root: Trusted Platform Module (TPM), which is a specially designed and implemented chip, functions as the trust root and employs cryptographic functions to support other mechanisms in the trusted platform;
- Secure boot or authenticated boot: Secure boot is a layered booting mechanism in which the lower layers verifies the integrity state of upper layers before the control is transferred from the lower to the upper. The transfer is only allowed when upper layer is of integrity. In case of being compromised, the boot process terminates. While the authenticated boot only records the measurements of each layers at booting time and leave the integrity verification in the process of attestation.
- Memory curtaining: A strong, hardware-enforced memory isolation feature to guarantees the integrity of executing processes and prevents tampering from other unauthorized processes.
- Secure I/O: Provides a secure path from the keyboard to an application and from the application back to the screen.
- Sealed storage: A mechanism to protect data by keys based in part on the identity of the software requesting to use them and in part on the identity of the computer on which that software is running.
- Remote attestation: A process to prove the trustworthiness of remote parties including hardware and software.

2.2 Remote Attestation

With other fundamental features, trusted computing platforms employs attestation mechanism to authenticate themselves including their hardware and software. The attesting platform employs TPM's integrity measurement to measure the platform and securely storages these measurements. On receiving an attestation request from a challenger, the attesting platform sends measurements back to the challenger. The challenger verifies these measurements to check the configuration of attesting platform. This attestation function provides a primitive service for attesting higher level applications. However, as the dynamically changed programs and theirs runtime environments, this primitive attestation mechanism is not enough to achieve the goal stated in the definition of trust by trusted computing [1], which is to prove whether a remote system behaves as expected. The

TCG attestation is also static, inflexible and inexpressive [4]. Its integrity report mechanism still has to face the program upgrades, program patches and revocation problems.

In order to solve these problems of TCG attestation, some remote attestation schemes were introduced. Integrity Measurement Architecture (IMA) [3] and property based attestation [15, 6, 16] only concerns the specific configuration states of remote platform. Shi et al. proposed BIND [17] which is a fine-grained attestation mechanism. However, BIND only attests some specific program code blocks' integrity. Besides lacking of behavior attestation, these attestation mechanisms still have problems as following [7]: the giant complexity of software system configurations; the specific semantic meanings of such software configurations have not been specifically studied. The semantic remote attestation [4] introduced a trusted virtual machine (TVM) based semantic monitoring and remote attestation scheme. It specified that attestation should attest dynamic system behaviors, not a particular binary. Li et al. [12] and Zhang et al. [18] both introduced the concept of behavior remote attestation. Zhang et al. [18] constructed a process tree which reflects the domination relationship among processes, and this process tree is used for remote attestation to identify suspect processes. However, their work did not concern all possible security behaviors such as data objects involved behaviors.

2.3 Security Policy

Operating system security mechanisms are foundations to enforce the separation of information based on confidentiality and integrity requirements to guarantee system security. System policy is a set of rules governing subjects and objects in system, and it specifically restricts the behaviors of subjects (processes and users) in system. For example, it specifies which subjects can access which objects. Commonly, there exist two categories of access control models: Discretionary access control(DAC) and Mandatory access control (MAC). Traditional operating systems employ DAC mechanisms. However, these DAC mechanisms are vulnerable to tampering and bypass, and malicious program can easily compromise the system security. These vulnerabilities of DAC models can be addressed by MAC mechanisms [19]. MAC policy is enforced over all subjects (processes) and objects (e.g., files, sockets) in system. The Security-Enhanced Linux (SELinux) [20] is one of the most promising solutions to enforce MAC policy in operating system. SELinux implements the flexible and fine-grained MAC architecture Flask [9] in the Linux kernel. It separates the policy decision-making logic from the policy enforcement logic which is encapsulated within a single component known as the security server with a general security interface. SELinux provides an example security server which implements a combination of Type Enforcement model , Role-Based Access Control (RBAC), and optional Multi-Level Security.

Different types of security models are widely applied in different applications. Meanwhile, different platforms may have different implementations for a specific security model. For some situations, challengers may only want to attest the higher level behavior model of system behaviors instead of checking all these lower level details. For example, a specific application program may only be concerned about its application level security policies by ignoring the implementation details in operating system level security policies. For these considerations, we introduce a model-driven remote attestation scheme to support a higher level behavior model attestation to prove that the remote target behaves as expected, or in other words, to prove whether the remote target is trustworthy. In our scheme, the attesting target can be a process, a sub system, a pure software system hosting on certain platform, as well as a whole platform. Remote attestation is supposed to prove whether the attesting target and its execution environment satisfy certain behavioral expectation.

3 Model-Driven Remote Attestation

3.1 Behavior Model

As the definition of trust in TCG specifications, the purpose of remote attestation should attest that a remote target behaves as expected. We consider two types of behaviors in our behavior model: attesting target's behaviors and policy change behaviors. Attesting target's behaviors are restricted by the policy enforcement mechanism. Policy change behaviors refer to policy creating, replacing or updating behaviors which dynamically change the system security policies and affect the behaviors of attesting target.

We use a state machine model [21] to depict the system behaviors and policy change behaviors. A system is a state machine M which is defined as $(U, S, SC, Out, Capt, CC, f_{out}, f_{do}, f_{cdo}, S_0, t_0)$:

- U: A set whose elements are subjects (processes,users), and its element is denoted as u;
- S: A set whose elements are system states, which are determined by the states of all its subjects and objects, and its element is denoted as s;
- SC: A set whose elements are process' state changing commands, and its element is denoted as sc;
- Out: A set whose elements are all possible outputs;
- $Capt$: A set of capability tables which specify the permission of subjects, and these capabilities are determined by the system policy;

- CC: A set of policy state changing commands which change the $Capt$, and its element is denoted as cc;
- $f_{out}(s, Capt, u)$: $S \times Capt \times U \rightarrow Out$, a function gives specific outputs when the system is with a specific state, a specific user, and a specific capability set;
- $f_{do}(s, Capt, u, sc)$: $S \times Capt \times U \times SC \rightarrow S$, a system state change function;
- $f_{cdo}(s, Capt, u, cc)$: $Cpat \times U \times CC \rightarrow Capt$, a capability set state change function;
- S_0: The initial machine state;
- t_0: The initial state of capability set;

Trustworthy Behavior Model A system policy specifies behaviors of all subjects in system, and all these behavior permissions are denoted as a set $Capt$. System behaviors are executions of a sequence of state change commands. Let C=$SC \bigcup CC$, and then a subset of C represents a behavior. Let Ab denote the set of all C's subsets, then the $Capt = \{U \times S \rightarrow Ab\}$. These behavior permissions represent the expected behaviors which the remote target is supposed to or not to carry out. The behavior models come into two kinds: expected behavior model and enforced behavior model. Expected behavior model is a behavior pattern that a remote target is supposed to behave as, which is also the expectation of challenger; enforced behavior model is the behavior pattern contained by the enforced policies in system at runtime, which is the exact enforced one.

An enforced policy is trustworthy only when the enforced behavior model complies with the expected behavior model of challenger. A behavior model $\mathcal{M}_1$ complies with another behavior model $\mathcal{M}_2$, only when $\mathcal{M}_1$ holds a more restricted security requirements than $\mathcal{M}_2$, or $\mathcal{M}_1$ holds the equivalent behavior model as $\mathcal{M}_2$, or $\mathcal{M}_1$ has exactly the same implementation as $\mathcal{M}_2$.

Trustworthy Behavior: At runtime, system enforces security policy to restrict system behaviors. Therefore attesting target's enforcement behaviors can be identified by the system state change function: $f_{do}(s, Capt, u, sc) : S \times Cpat \times U \times SC \rightarrow S$. An enforcement behavior of subject u in system is denoted as $b_s : s_{i+1} = f_{do}(s_i, Cpat, u, sc)$. Security policy may be changed and these kinds of behaviors can be represented by the capability set state change function: $f_{cdo}(s, Capt, u, cc) : Cpat \times U \times CC \rightarrow Cpat$. A policy change behavior of subject u in system is denoted as $b_p : Capt_{i+1} = f_{do}(s, Cpat_i, u, sc)$. For a state $s_i \in S$, if s_i is in a trustworthy state, and a subject u's behavior $b_s : s_{i+1} = f_{do}(s_i, Cpat, u, sc)$ is correctly executed, then the system state s_{i+1} is trustworthy. The correct execution of b means that the program codes of b's state change commands are correctly executed.

Trustworthy Behavior Patterns For a sequence of system behaviors $Seq = b_1 b_2 ... b_n$, if all these behaviors are correctly executed and the system starts from a trustworthy state s_0 and finishes at a state s_n, then s_n is trustworthy. Different kinds of behavior patterns can be transformed to a composition of behavior sequences. At runtime, remote target behaves in certain patterns. If these behavior patterns are correctly executed and the system starts from a trustworthy state, then the final state of the system is trustworthy and this execution process is trustworthy.

3.2 Behavior Model Attestation

For TCG specification, in order to prove whether a remote target is trustworthy or not, it is necessary to verify whether the remote target satisfies following two requirements.

- **Expected behavior compliance requirement**: The security policies enforced by system should be in compliance with the expected behavior models.
- **Enforced behavior compliance requirement**: The system behaviors should exactly follow the security policy and the state change commands (codes) of these behaviors should be correctly executed.

Expected Behavior Compliance The security policy enforced by a remote target may be changed dynamically. The enforced policy may also be different types and with varieties of implementations on different platforms. An ideal solution to check the compliance among different policy implementations is to evaluate their behavior model according to the expected behavior model. For all different policy implementations, they can be transformed to some low-level policy languages like XACML [22] to represent the same behavior model and support compliance check. For some applications, it is also possible to extract the behavior model from the policy implementation and represent the behavior model at an abstract level, like the behavior model in section 3.1. At time of verification, a challenger verifies the intermediate representation target policy or abstract behavior models according to the expected behavior model. For different types of abstract models, we may employ model transformation techniques to verify their equivalence, such as OMG's Query View Transformation (QVT) [23].

Another solution to check the compliance among different policy implementations is to directly compare the target implementations with standard expected implementations. For a specific expected behavior model, we may employ some automatic policy generation tools to get all its possible implementations on some specific platforms [24]. At time of attestation, a challenger may compare the enforced policy with all these expected policy implementations. If there is any one implementation which is exact the same as the enforced policy, the enforced policy is trustworthy.

Enforced Behavior Compliance Enforced behavior compliance requires that the actual behaviors (program executions) are correctly executed and comply with the enforced behavior model. In order to achieve the enforced behavior compliance, it is necessary to guarantee the correct function of policy enforcement components and correct execution of behavioral commands. The policy enforcement components and the behavioral commands are essentially program codes. The problem is how to guarantee the correct execution of program codes. We employ the isolated memory of trusted virtual machine to support the correct execution of program codes [2, 25]. Immediately before a program is executed, the program's state is recorded. Then the program runs with the protection of isolated memory. It is possible to check whether the program codes are correctly executed by attesting the state of program codes immediately before they are executed.

In our scheme, the target's behaviors and the policy change behaviors are all recorded for verification. The record of a target behavior contains the states of the system before and after the behavior's enforcement, as well as the state of the behavior's commands. The record of a policy change behavior contains the policy before and after the behavior's execution. In order to attest whether a remote target satisfies these two compliance requirements introduced above, the process of verification in remote attestation consists of two parts: verification on expected behavioral model and verification on behavioral commands' execution.

3.3 Model-Driven Remote Attestation Architecture

The architecture of our model-driven remote attestation scheme is shown in Figure 1. The architecture of our scheme follows the basic framework of TCG attestation which consists of two parties: challenger $\mathcal{CH}$ and attester platform $\mathcal{H}_r$. The attester platform is supposed to be equipped with a TPM which functions as the trust root of the attester platform. The attester platform is also required to install an attestation service $\mathcal{AS}$ and a policy enforcement component $\mathcal{PE}$. The $\mathcal{PE}$ enforces specific security policies on $\mathcal{H}_r$. The $\mathcal{AS}$ runs as a service in operating system to monitor and record all related subjects, objects and their behaviors. The communication between the challenger $\mathcal{CH}$ and attesting platform $\mathcal{H}_r$ should be protected by a secure channel.

We use trusted virtual machine (TVM) to monitor the attesting target's behaviors and related semantic information like configuration state. TVM based approach has following advantages: the TVM can provide strong process isolation via virtual memory management; the executed codes, including operating system, are completely monitored by TVM. It is also possible to replace TVM with Secure Kernel (SK) [25] in our scheme. SK also provides the strong process isolation at runtime.

Trust Chain The TPM is the trust root of remote attestation. The $\mathcal{AS}$ employs features of TPM to guarantee the trustworthiness of the remote attestation process. When the attesting platform is being booted, an authenticated boot process is employed to record all the states of the platform from hardware to operating system. For application programs, attestation service monitors target programs and records their states immediately before they are executed. These measurements are protected by the secure storage of TPM. At the time of attestation, a challenger may verify the trust chain from TPM to $\mathcal{AS}$ by evaluating these measurements. $\mathcal{AS}$ monitors and records $\mathcal{PE}$'s execution, and these records are used to prove whether the policy enforcement mechanism functions correctly.

Attestation Service The attestation service can be implemented as an operating system service to monitor the behaviors of attesting target and security policy enforcement and changes. Attestation service dynamically records all related subjects' states before and after the enforcement of behaviors. The behavioral information for $\mathcal{AS}$ to record includes: state change behaviors in the system, states of system subjects (programs, data objects, etc.) before and after the enforcement of behaviors, states of policy enforcement components $\mathcal{PE}$ immediately before the execution, security policies.

Attestation Procedure A typical attestation process is carried out as following steps: With the support of TPM and trusted virtual machine , $\mathcal{AS}$ monitors and records the states and behaviors of running programs and their runtime environment in remote system, including the policy enforcement components $\mathcal{PE}$; When $\mathcal{CH}$ sends a challenge message to $\mathcal{AS}$, $\mathcal{AS}$ returns measurements back to $\mathcal{CH}$ and $\mathcal{CH}$ verifies the measurements to attest whether the remote system behaves as expected. In verification phase, challenger first verifies the initial subjects states in the system before these behaviors, then verifies the correct execution of these behaviors. If no failed verification happens, the enforced behavior compliance is satisfied. Then challenger has to verify the enforced behavior model is trustworthy. As we

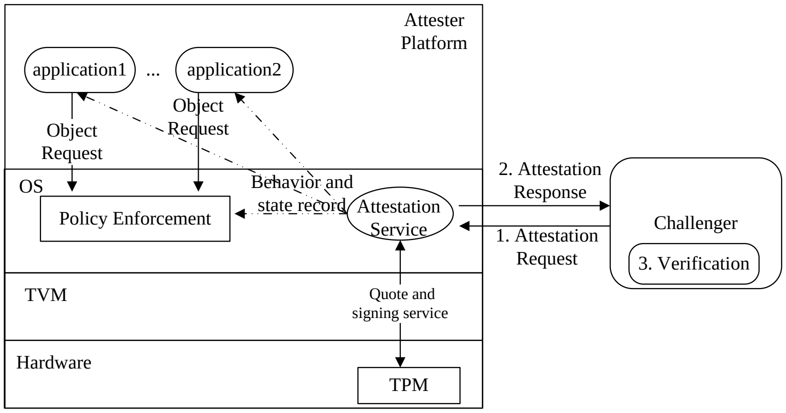

Figure 1. Model-Driven Remote Attestation Architecture

have introduced in Section 3.2, there are two ways to check the expected behavior compliance. The first way is to verify the enforced policy by comparing it with standard policy implementations which stand for the expected behavior models. If one policy instance is matched with the enforced policy, the enforced behavior compliance is satisfied. The second approach is to extract behavior models from the enforced policies and comparing the extracted models with the expected behavior model. If these extracted models comply with the expected behavior model, then the expected behavior compliance is satisfied.

3.4 Implementation: Attestation Service in Linux

We are now building our attestation service with Linux Security Module (LSM) [10] in Linux with 2.6.24 kernel. The LSM interfaces provide a broad set of hooks for enforcing system access control policy for the kernel. For example, SElinux [11] is a typical Mandatory Access Control implementation of LSM, which supports dynamic policies. Our attestation service dynamically monitors the system behaviors, and intercepts the information of system calls handled by these hooks, including program execution, file operations, kernel modules, as well as inter-process communication. By analyzing these operations' initiators and targets, attestation agent monitors the dynamic behaviors of processes and records their states. We implement the process of measuring a specific objects as a separated routine which is called by these inspected points in selected LSM hooks. The measurement service calls are inserted into hook functions at critical points to dynamically monitor the running of processes and carry out measurements for specific targets. This measurement service measures the target objects and securely stores the results as well as dynamic dependencies via TPM encryption function.

In LSM, the policy decision-making logic is encapsulated within a single component known as the security server with a general security interface. Therefore our attestation service specifically monitors the security server to attest the enforced behavior compliance.

4 Related Work

Most of the existing hardware-based attestation mechanisms employ TPM as the trust root. Terra [2] uses a Trusted Virtual Machine Monitor (TVMM) to partition a tamper resistant hardware platform into multiple virtual machines (VM) that are isolated from each other. With the protection of trusted hardware, TVMM provides both open-box VM and closed-box VM. TVMM can identify the contents of the closed box to remote parties, guaranteeing the trustworthiness of the content. Sailer et al. [3] introduced an integrity measurement architecture (IMA) which employs a loading time integrity measurement mechanism based on TPM to measure dynamic executable content from the BIOS all the way up to the application layer. Haldar et al. [4] introduced a semantic attestation mechanism based on a trusted virtual machine (TVM). The TVM based semantic attestation mechanism enables the remote attestation of high-level program properties. Shi et al. [17] proposed BIND which is a fine-grained attestation mechanism. BIND provides evaluation interfaces to attest the concerned pieces of code in an application. Jaeger et al. [5] introduced the Policy-Reduced Integrity Measurement Architecture (PRIMA) based on the information flow integrity in mandatory access control (MAC) policy. Li et al. [12] presented a system behavior-based attestation model which determines the trust state of target platform from its system trustworthiness related behaviors. Poritz et al. [15], Sadeghi et al. [6] and Chen et al. [16] separately introduced the property-based attestation which employs TPM to measure and verify the evidences of security properties without revealing the exact configurations of a target platform.

5 Conclusion

In this paper, we introduced the model–driven remote attestation to prove whether a remote system behaves as expected. Our model-driven remote attestation scheme is mainly carried out in two steps. First, being supported by trusted virtual machine on the remote system, the attestation service dynamically monitors and records system behaviors including system state changes and security policy enforcements. Second, at the time of attestation, a challenger verifies two trustworthy requirements to attest whether the remote system behaves as expected: expected behavior compliance and enforced behavior compliance. The expected behavior compliance is verified by extracting behavior model from enforced policies or generating policy instances for expected behavior model; the enforced behavior compliance is checked by verifying the runtime records of system behaviors. By attesting the behavioral aspect of remote systems, our model–driven remote attestation scheme brings us a step closer to the final goal of remote attestation in TCG specifications: to attest that a remote system behaves as expected.

Acknowledgements This work is partly supported by the High-Tech Research and Development Program of China under Grant No. 2007AA010301 and partly supported by the Office of Research, Singapore Management University.

Liang Gu is a Ph.D student at the Peking University and is currently on attachment to the School of Information Systems, Singapore Management University.

References

[1] Trusted Computing Group, "Trusted computing specifications." `http://www.trustedcomputinggroup.org/specs/`.

[2] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A Virtual Machine-Based Platform for Trusted Computing," in *SOSP 2003*, (Bolton Landing, New York, USA), October, 2003.

[3] R. Sailer, X. Zhang, T. Jaeger, and L. v. Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," in *Proceedings of the 13th USENIX Security Symposium*, (San Diego, CA, USA), August, 2004.

[4] V. Haldar, D. Chandra, and M. Franz, "Semantic Remote Attestation–A Virtual Machine directed approach to Trusted Computing," in *the Third virtual Machine Research and Technology Symposium (VM '04). USENIX.*, 2004.

[5] T. Jaeger, R. Sailer, and U. Shankar, "PRIMA: policy-reduced integrity measurement architecture," in *SACMAT '06 : Proceedings of the eleventh ACM symposium on Access control models and technologies*, (New York, NY, USA), pp. 19–28, ACM Press, 2006.

[6] A.-R. Sadeghi and C. Stble, "Property-based attestation for computing platforms: caring about properties, not mechanisms," *New security paradigms*, 2004.

[7] J. M. McCune, A. Perrig, A. Seshadri, and L. van Doorn, "Turtles all the way down: Research challenges in user-based attestation," in *Proceedings of the Workshop on Hot Topics in Security (HotSec)*, Aug. 2007.

[8] RSBAC–Rule Set Based Access Control . http://www.rsbac.org/.

[9] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau, "The Flask Security Architecture: System support for diverse security policies," in *Proceedings of the eighth USENIX Security Symposium (Security '99), August 23–26, 1999, Washington, DC, USA* (USENIX, ed.), (pub-USENIX:adr), USENIX, 1999.

[10] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux Security Modules: General security support for the Linux kernel," in *Proceedings of the 11th USENIX Security Symposium*, USENIX, Aug. 2002.

[11] S. Smalley, C. Vance, and W. Salamon, "Implementing SELinux as a Linux security module," Report #01-043, NAI Labs, Dec. 2001. Revised May 2002.

[12] L. Xiao-Yong, S. Chang-Xiang, and Z. Xiao-Dong, "An efficient attestation for trustworthiness of computing platform," in *Proceedings of the 2006 International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP'06)*, 2006.

[13] AMD, "AMD64 Virtualization Codenamed "Pacifica" Technology–Secure Virtual Machine Architecture Reference Manual," Tech. Rep. Publication Number 33047, Revision 3.01, AMD, May 2005.

[14] I. Corporation, "LaGrande technology preliminary architecture specification," Tech. Rep. Document Number: 315168 002, Intel Corporation, Sept. 2006.

[15] J. A. Poritz, "Trust[ed| in] computing, signed code and the heat death of the internet," in *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, (New York, NY, USA), pp. 1855–1859, ACM Press, 2006.

[16] L. Chen, R. Landfermann, H. Loehr, M. Rohe, A.-R. Sadeghi, and C. Stüble, "A Protocol for Property-Based Attestation," in *Proceedings of the 1st ACM Workshop on Scalable Trusted Computing (STC'06)*, ACM Press, 2006.

[17] E. Shi, A. Perrig, and L. V. Doorn, "BIND: A Fine-Grained Attestation Service for Secure Distributed Systems," in *2005 IEEE Symposium on Security and Privacy*, 2005.

[18] H. Zhang and F. Wang, "A Behavior-Based Remote Trust Attestation Model," *Wuhan University Journal of Natural Sciences*, vol. 11, 2006.

[19] P. Loscocco and S. Smalley, "Integrating flexible support for security policies into the Linux operating system," tech. rep., U.S. National Security Agency (NSA), Feb. 2001.

[20] S. Smalley, C. Vance, and W. Salamon, "Implementing SELinux as a Linux security module," Report #01-043, NAI Labs, Dec. 2001. Revised May 2002.

[21] J. A. Goguen and J. Meseguer, "Security policies and security models," in *Proc. IEEE Symposium on Security and Privacy*, pp. 11–20, 1982.

[22] 'OASIS' eXtensible Access Control Markup Language (XACML) . `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`.

[23] OMG, "Meta object facility (mof) 2.0 query/view/transformation specification." http://www.omg.org/docs/ptc/07-07-07.pdf, 2007.

[24] B. Agreiter, "Model-driven configuration of os-level mandatory access control: research abstract," in *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, (New York, NY, USA), pp. 995–998, ACM, 2008.

[25] "AMD platform for trustworthy computing," in *Microsoft WinHEC 2003*, pp. 78–89, `http://www.microsoft.com/whdc/winhec/papers03.mspx`.