Building Dynamic Remote Attestation Framework

Wenjuan Xu
UNC Charlotte
wxu2@uncc.edu

Gail-Joon Ahn
Arizona State University
gahn@asu.edu

Hongxin Hu
Arizona State University
hxhu@asu.edu

Xinwen Zhang
Computer Science Lab
Samsung Info. Sys. America
xinwen.z@samsung.com

Jean-Pierre Seifert
Deutsche Telekom Lab and
Technical University of Berlin
jean−pierre.seifert@telekom.de

ABSTRACT

Remote attestation is an important mechanism to provide the trustworthiness proof of a computing system by verifying its integrity. In this poster, we propose a dynamic remote attestation framework for efficiently attesting a target system based on an information flow-based integrity model. With this model, the high integrity processes of a system are first verified through measurements and these processes are then protected from accesses initiated by low integrity processes. Also, our framework verifies the latest changes in a system's new state, instead of considering the entire system information. In addition, we adopt a graph-based method to represent integrity violations with a ranked violation graph, which supports intuitive reasoning of attestation results. We also describe our experiments and performance evaluation.

1. INTRODUCTION

In distributed computing environments, it is crucial to measure whether remote parties run buggy, malicious application codes or are improperly configured by rogue software. Normally, we analyze the integrity of remote systems to determine their trustworthiness. Trusted Computing Group (TCG) [1] introduces a hardware-based approach called trusted platform module (TPM) which can securely store and provide integrity measurements of systems to a remote party. In addition, remote attestation mechanisms have been proposed to facilitate such capabilities of TPM at application level. For instance, Integrity Measurement Architecture (IMA) [5] is an implementation of TCG approach to provide verifiable evidence with respect to the current runtime state of a measured system.

Typical attestation mechanisms are designed based on the following steps. First, an attestation requester (*attester*) sends a challenge to a target system (*attestee*), which responds with the evidence of integrity of its hardware and software components. Second, the attester derives runtime properties of the attestee, and determines the trustworthiness of the attestee. By adopting such an intuitive approach, we can help reduce potential risks that may be caused by the tampered remote systems. However, these existing approaches still need to cope with the challenges for attesting a platform where its *system state* frequently changes due to system-centric events such as updating security policies or installing new software packages. In addition, the efficienct attestation mechanism should be especially considered for dealing with such a dynamic nature of systems since the frequency of system changes and the volume of system state information would be tremendously increased due to the recent technological innovation of distributed computing. Consequently, it is necessary to have an effective way for presenting the attestation results and accommodating such results while resolving any identified security violations.

In this poster, we propose a framework for dynamic remote attestation to address the aforementioned issues. Our framework is based on system integrity property with a *domain-based isolation* principle. The high integrity processes of a system are first verified through measurements and these processes are then protected from accesses initiated by low integrity processes. Having this principle in place, our framework allows us to verify whether certain applications in the attestee satisfy integrity requirements as part of system attestation. Also, our framework mainly verifies the latest changes in a system's *new state*, instead of considering the entire system information. Through these two tactics, we believe our framework can efficiently attest the target system. Also, we adopt a graph-based analysis methodology [7] for analyzing policy violations, which would help cognitively identify suspicious information flows in an attestee. Besides, our ranking scheme is utilized for prioritizing the policy violations. We also share our experiment results to demonstrate the feasibility and practicality of our approach comparing to static attestation approaches.

2. DOMAIN-BASED ISOLATION

In a general-purpose computing system, an application's integrity not only depends on the integrity level of codes and data, but also relies on interactions between applications concurrently running on the platform. Considering this requirement, we develop a domain-based isolation model for integrity evaluation. In this model, we mainly focus on identifying and protecting high integrity processes (*system TCB* and *domain TCB*).

System TCB is a concept that is the same as the concept of traditional TCB [2]. Reference monitor-based approach is proposed to identify system TCB through the identification of subjects functioning as the reference monitor in a system [3]. In practice, other than system TCB protection, an application or user-space service is required to achieve integrity assurance as well by controlling information flow among running processes. Hence, we introduce a concept called *domain* TCB. Let *d* be an information domain functioning as a certain application or service through a set of related subjects and objects. We denote domain TCB of an information domain *d* by TCB(d). TCB(d) is composed of a set of subjects and objects in information domain *d* which have the same level of security sensitivity or similar protection requirements. TCB(d) can be initially identified through keywords and direct information flows. With the identifications of system TCB and TCB(d), for an information domain *d*, all other subjects in a system are categorized as NON-TCB. To protect the integrity of system TCB and TCB(d), we develop our domain-based isolation principle which is similar to those in Clark-Wilson [6]. Domain-based isolation is satisfied for an information domain d if (i) information flows are

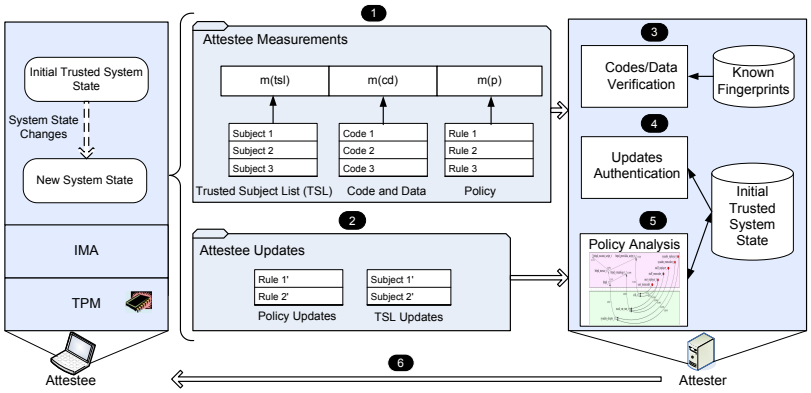


Figure 1: Dynamic remote attestation framework

from TCB(d); or (ii) information flows are from system TCB to either TCB(d) or TCB(d) protected resources; or (iii) information flows are from NON-TCB to either TCB(d) or TCB(d) protected resources via filter(s). Filters can be processes or interfaces that normally are distinct input information channels and are created by a particular operation such as `open()`, `accept()`, or other calls that enable data input. For example, `su` process allows a low integrity process (e.g., staff) to be a high integrity process (e.g., root) by executing `passwd` process, thus `passwd` can be regarded as a filter for processes run by root privilege.

3. DYNAMIC REMOTE ATTESTATION FRAMEWORK

Our framework includes several attestation processes, in which the attester verifies the different *system state* of the attestee based on domain-based isolation as shown in Figure 1. Here, the system state T_i at time i is defined based on different information flow-related conditions in an attestee, including a trusted subject list TSL_i (high integrity subjects belonging to system TCB and TCB(d)), a set of codes and data for loading these subjects CD_i, and a security policy $Policy_i$. The steps for verifying if system state T_i in moment i satisfies domain-based isolation can be seen as follows.

Measurements on Attestee (Step 1) In order to provide information for the attester to verify integrity property, an attestee measures state T_i and generates the measurement list M_i including trusted subject list (TSL_i) measurement, codes and data (CD_i) measurements, and policy ($Policy_i$) measurement. These measurements are generated by comparing the hash values of corresponding items and added to the measurement list.

Attestee Updates (Step 2) The attestee only sends the updated state information (typically TSL and/or policy) to the attester. Here, we only explain how to identify policy updates. Normally, the attestee can change its policy by modifying several policy rules, or adding or removing a set of policy rules after installing or uninstalling software. To have a complete list of policy updates, we require the new policy to be compared with the old policy. Then the changed policy with corresponding rule marks are added into the updates and sent to the attester.

Codes and Data Verification (Step 3) From the received measurements M_i, the attester retrieves the hash value of CD_i for trusted subject list TSL_i, and checks if it has corresponding known good fingerprints.

Authenticating Updates (Step 4) To prove that the received updated information is from the attestee, we authenticate the updated information by verifying that all measurements and integrity reporting subjects in the attestee are not altered by any system update. That is, the measurement components belong to TSL_i, and its codes and data should be measured. Also, the information flow to these components must be restricted within TSL_i subjects. Note that these components can be updated but after any updates of these components, the system should be fully measured and attested from boot time.

Policy Analysis (Step 5) To verify if there is integrity violation, we analyze policy updates using a graph-based analysis method. In this method, a policy file is first visualized into a graph. Then this policy graph is analyzed against our domain-based isolation principle and a policy violation graph is generated from this analysis step. On the attester side, this graph-based policy analysis tool continuously runs for performance consideration. Upon receiving the updated information from the attestee, the attester analyzes these updates based on the previous violation graph.

Attestation Result Feedback (Step 6) The attester also sends a snapshot of policy violation graph to the attestee for assisting the reconfiguration of its system configuration. Moreover, with this policy violation graph, the attester prioritizes the violations with ranking [4] and the trustworthiness of the attestee.

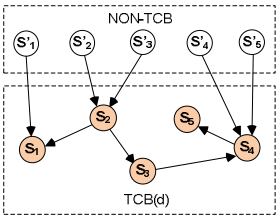
4. POLICY VIOLATION ANALYSIS

In addition to the boolean-based response of existing attestation solutions, we adopt a graph-based policy analysis mechanism, where a policy violation graph can be constructed for identifying all policy violations on the attester side and a ranking scheme is adopted to evaluate how severe the discovered policy violations are.

Figure 2 (a) shows an example of policy violation graph which examines information flows between NON-TCB and TCB(d). Five direct violation paths are identified in this graph: <S_1', S_1>, <S_2', S_2>, <S_3', S_2>, <S_4', S_4>, and <S_5', S_4>, crossing all the boundaries between NON-TCB and TCB(d). Also, eight indirect violation paths exist. For example, <S_2', S_5> is a four-hop violation path passing through other three TCB(d) subjects S_2, S_3, and S_4.

In order to explore more features of policy violation graphs and facilitate efficient policy violation detection and resolution, we introduce a scheme for ranking policy violation graphs. There are two major steps to rank a policy violation graph as follows:

Ranking Subjects in TCB(d) TCB(d) subjects in the policy violation graph are ranked based on dependency relationships among them. The rank of a TCB(d) subject shows reachable probability of



(a) A policy violation graph

Subject	SubjectRank
S_1	0.2799
S_2	0.3999
S_3	0.0799
S_4	0.4319
S_5	0.3456

(b) SubjectRank in TCB(d)

Path	PathRank
$\langle S'_1, S_1 \rangle$	0.2799
$\langle S'_2, S_2 \rangle$	0.8104
$\langle S'_3, S_2 \rangle$	0.8104
$\langle S'_4, S_4 \rangle$	0.6048
$\langle S'_5, S_4 \rangle$	0.6048

(c) PathRank between NON-TCB and TCB(d)

Figure 2: Example policy violation graph and rank. The SubjectRank and PathRank indicate the severity of violating paths.

information flows from NON-TCB subjects. Our notation of *SubjectRank* (SR) in a policy violation graph is a criterion that indicates the likelihood of information flows, which may come to a TCB(d) subject from NON-TCB subjects through direct or indirect violation paths. Figure 2 (b) illustrates how our ranking scheme can be applied to the policy violation graph shown in Figure 2 (a).

Ranking Direct Violation Path Direct violation paths in the policy violation graph are evaluated based on the ranks of TCB(d) subjects to indicate severity of these paths which allow low integrity information to reach TCB(d) subjects. We further introduce *PathRank* (PR) as the rank of a direct violation path. Direct violation paths are regarded as the entrances of low integrity data to TCB(d) in policy violation graph. Therefore, the ranks of direct violation paths give a guide for system administrator to adopt suitable defense countermeasures for resolving identified violations. Figure 2 (c) shows the result using the ranking scheme to calculate the *PathRank* of the example policy violation graph.

5. IMPLEMENTATION AND EVALUATION

Based on our framework, for the implementation of measurement verification, we start with a legitimate attestee system and make measurements of the system for the later verification. To present system state information and attestation result via self explanatory graphical user interface, the policy analysis module is developed as GUI application. Several graph-based policy analysis tools are available from the literatures. We leverage our previous work [7] for this purpose, which has the capability of visual queries. Our attestee platform is a Lenovo ThinkPad X61 with Intel Core 2Duo Processor L7500 1.6GHz, 2 GB RAM, Atmel TPMv1.2, and is installed with Fedora Core 6. We enable SELinux with the default policy based on the Fedora distribution.

To exam the scalability and efficiency of our framework, we evaluate its performance. Our performance analysis is based on the system policy changes. We attest the performance on attestee side and attester side separately (shown in Table 1 and Table 2). Based on our result, the increase of policy size requires more time for attestation on the attestee and attester side, and vice versa. In addition, we also compare the overhead of our approach with a static attestation. In the static approach, an attestee sends all system state information to an attester, and the attester verifies all information step by step. The results show that our dynamic approach can dramatically reduce the overhead compared to the static approach.

6. CONCLUSION

We have presented a dynamic remote attestation framework for efficiently attesting a target system. Our framework is based on an information flow-based domain isolation model to utilize the integrity requirements and identify integrity violations of a system.

Table 1: Attestation Performance For Dynamic Method (in seconds)

Change	**Dynamic**		
Size	attestee	attester	overhead
No change	0.23	0	0.23
-0.002MB (Reduction)	0.122	0.94	1.06
-0.019MB (Reduction)	0.09	0.91	1.00
-0.024MB (Reduction)	0.06	0.90	0.96
0.012MB (Addition)	0.38	0.96	1.34
0.026MB (Addition)	0.60	1.07	1.67

Table 2: Attestation Performance For Static Method (in seconds)

Change	**Static**		
Size	attestee	attester	overhead
No change	14.76	90.13	104.89
-0.002MB (Reduction)	14.76	90.11	104.87
-0.019MB (Reduction)	14.74	89.97	104.34
-0.024MB (Reduction)	14.74	89.89	104.23
0.012MB (Addition)	14.77	90.19	104.96
0.026MB (Addition)	14.78	90.33	105.11

We also adopted a graph-based methodology to represent integrity violations in an intuitive way with the ranking scheme. In addition, our results showed that our dynamic approach can dramatically reduce the overhead compared to the static approach.

7. REFERENCES

[1] Trusted Computing Group. https://www.trustedcomputinggroup.org/home.

[2] *Trusted Computer System Evaluation Criteria*. United States Government Department of Defense (DOD), Profile Books, 1985.

[3] A. P. Anderson. Computer security technology planning study. *Technical Report ESD-TR-73-51*, II, 1972.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[5] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[6] R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.

[7] W. Xu, M. Shehab, and G. Ahn. Visualization based policy analysis: case study in SELinux. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 165–174. ACM New York, NY, USA, 2008.