TEM: A Novel Measurement Method for Java Applications on demand in Cloud Computing

Haihe Ba, Songzhu Mei, Jiangchun Ren, Zhiying Wang
College of Computer, National University of Defense Technology, Changsha China
E-mail: bahaihe@hotmail.com

***Abstract*—While Cloud computing brings much convenience to companies; it also produces many a threat to information security. TEM is a Java-based architecture that gives strong safety guarantees to high-assurance Java applications in cloud computing. TEM advances the state of the art in Java Virtual Machine in both the design of its own fine-grained integrity measurement on demand and in the ability to ensure recoverability in the face of malicious attacks. In this paper, we introduce TEM, as a novel fine-grained measurement method for Java application, which takes advantages of the structural feature of Java class file. It not only satisfies users' necessary trust from the trusted platform module, or TPM, but also has less impact on running performance in cloud environment.**

Keywords-TEM, Cloud computing, fine-grained, TPM

I. INTRODUCTION

Cloud computing has generally emerged as one of the most influential technologies in both the industry and academia about information technology. As a novel computing resource organizing methods, cloud computing can offer scalable, flexible and unlimited application services to enterprises. As a result, many designate their application services to the cloud, however, they are not able to be aware whether the services is tampered to be untrusted [12].

As a pure object-oriented programming language, Java is widely used in many fields, such as embedded systems, mobile devices, servers and so on. Also, it is more and more popular in cloud computing. It is largely due to its own byte code structure, independent of hardware platform.

Nowadays, with the rapid development of Java, many a technology has been proposed to fulfill the need for security in Java Virtual Machine [4]. One of them is trusted computing which attracts wider attention. Trusted computing can give Java applications enough protections from the low-level hardware. Whereas, their chief focus is always on the integrity of entire application files that is too coarse-grained to meet realistic secure demand [1][2].

In this paper, we propose TEM to support the trust of Java applications guarantee in cloud computing. TEM has three significant advantages:

- Fine-grained measurement on demand. TEM makes utilization of the feature of Java class file to measure and verify every class to provide strong security for Java applications. And it takes advantages of Java Class Loader to have Java applications to measure on demand.
- High availability. TEM employs recovery and/or backup mechanism to support Java application availability with the lower cost, which is just to redirect the loading path.
- Trusted-degree management. According to the loading behavior of every class in some Java application, TEM makes use of Trusted-Degree Calculator component to evaluate trust degree of the Java application. Then, it gives users the trust value to support to make next action on the application.

The rest of this paper is organized as follows. Section II introduces the background about cloud computing, Java class file format, Java virtual machine and trusted computing. Then we provide the main motivation of TEM in Section III. Section IV is the design and implementation of TEM. Finally, Section V and Section VI cover the related work and conclusion of this paper.

II. BACKGROUND

A. Cloud Computing

National Institute of Standards and Technology, or NIST, has proposed the cloud model is composed of three service models [6].

- Infrastructure as a Service, or IaaS. It virtualizes IT infrastructure to elastic resource pool, where consumers is able to deploy and run any software from operating systems to the top applications.
- Platform as a Service, or PaaS. It provides some programming languages and tools to create and deploy users' own applications.
- Software as a Service, or SaaS. It gives consumers the providers' applications running in the cloud to use.

Consumers could use any of above models to deploy their applications according to the functionally and security concerns.

B. Java Class File Format

Java application is usually organized by a set of classes, part of different packages. Every class is compiled into a specific class file format by the compiler; and the Java interpreter executes these files at the runtime. The Java class file is an architecture-neutral byte format, with the feature of "compile once, run everywhere". It provides binary formats independent of the underlying platform to allow the Java applications to run on any hardware platforms and any operating systems [3] as long as it has Java virtual machine on.

Java application is typically packaged as JAR or WAR file, which is similar to the popular ZIP file format to aggregate many class files and other resources into one file. They not only are used to compress and release, but also allow Java applications at the runtime to deploy a set of classes and their associated resources.

C. *Java Virtual Machine*

Java Virtual Machine, or JVM is the core of the Java technology system. It is also the foundation stone for Java applications to achieve the feature of platform independence. JVM has defined a specific platform-independent instruction set to encapsulate its specific platform language features. The Java class file is parsed into a platform-independent instruction stream and further translated to the specific underlying machine instructions inside the Java Virtual Machine.

The Java Class file is loaded by class loaders. It supports dynamic class loading [5] in the Java virtual machine. There are three loaders that is, boot class loader, extension class loader and application class loader together to complete the loading process of the Java application in the typical JVM. It is called a parent delegation model: if you receive the request to load a given class, this first thing you do is to delegate it to your own parent class loader. Every loader does in the same way until the boot class loader. Only when the parent could not load the given class to the memory, does it try to operate itself independently.

D. *Trusted Computing*

In October 1999, Intel, Microsoft, IBM, Compaq and HP corporations have co-founded Trusted Computing Platform Alliance, or TCPA. And in March 2003, it was renamed Trusted Platform Group, or TCG.

TCG has proposed a trusted measurement chain model: BIOS Boot Block → BIOS → OS Loader → OS. Along this chain, the higher levels measure and verify the lower to ensure the security of computing system and give the trust guarantee to computing platform [7]. The root of trust in this chain comes from a hardware component on the motherboard of a platform called Trusted Platform Module, or TPM, with its trusted software stack, or TSS [8].

The object of measurement in the chain is trust, but it is hard to be directly measured. Instead, we usually select program integrity as the object, that is, the hash value of applications produced by some well-studied hash function. Then we check whether it is modified. It is often generated by extended calculation: New PCR_i = Hash (Old PCR_i || New Value). In this calculation, new values are linked to the original to build a new hash value as new measurement value, which can improve storage utilization effectively and provide consistency about computing system's behavior [11].

The remote attestation is used to attest the trust state of an entity to a remote entity, which is referred as integrity reporting and applied in a mountain of applications by different companies.

III. MOTIVATION OF TEM

A. *Threat Model*

With the rapid development of the cloud computing, applications' design and deploy pattern has been gradually shifting to higher application platform. It is the main reason that Java is very popular in enterprise information system as well. Nevertheless, it has been suffering from more and more malicious attacks.

TEM does not address application-level bugs, and will not stop an application that deliberately divulges secret data. TEM can detect a tampered application according to its integrity digest table. Also, TEM has an ability to find out dummy clients no matter what a bit it just modifies.

More subtle availability attacks are possible, such as deleting volatile data, which will be detected in a timely manner, but may lead to the loading failure and fail to provide continuous services. To solve above problems, we make TEM support the Java applications' availability by using backup and recovery technique.

B. *Inheritance Relations of Java Class Loader*

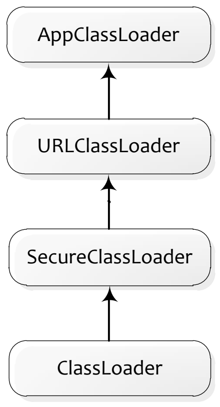

Figure 1. Inheritance relations of Java Class Loader

From the view of inheritances between Java Class Loader, we see Java Class Loader consists of four diverse class loaders as the Figure 1 is shown. From the top, it is AppClassLoader, an enter to load classes. Then we call the next URLClassLoader that finds and locates the resource in which the loading class is. SecureClassLoader has an implement of Java policies. What is at the lowest level is ClassLoader, which makes parent delegation model come true.

We have URLClassLoader to be reconstructed because of the important role it has. First, we need to know the loading class's location and its byte stream for the purpose of measurement and verification. Second, we expect not to have extra influence on Java Class loader's process as far as possible. Third, we hope that it can be suitable for not only common Java applications, but also Java web application containers, such as tomcat, etc.

C. *TEM Role*

As Figure 2 is pointed out, TEM has a significant role in trust chain. From the view of application developers, TEM gives strong security guarantees to the top Java application. As for the underlying hardware developer, it expends the trust chain to the application level. Also, we uses the JNI technique to encapsulate trusted software stack, or TSS, which is across the operating system to make TEM more trusted by using the hardware chip's API. It is made into trusted Java API to invoked by TEM to realize the basic mechanism, such as measurement, verification, encryption, attestation and so on.

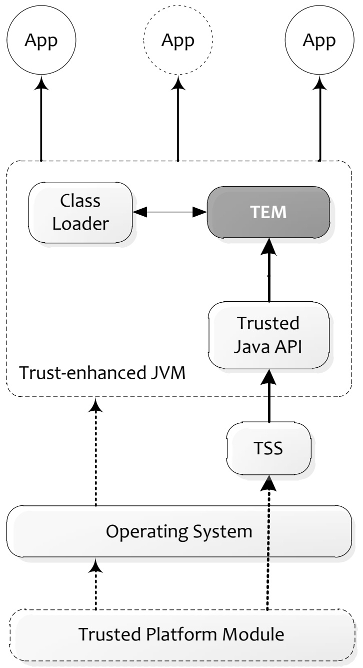

Figure 2. The role of TEM in the trust chain.

In the Figure 2, we see that both the TEM and Java Class Loader have worked on cooperation. Without the help of TEM, the Java Class Loader could not start out as usual. With respect to TEM, it has some main functions:

- Integrity Measurement. It invoke the trusted Java API to measure and/or verify the integrity of the loading class on demand.
- Recovery Mechanism. When the application passes before the loading time, it is bound to be backed up in a security position. The TEM would redirect the resource's path to security position once the loading class is proved to be invalid at the loading time.
- Trusted Audit. TEM records a series of operations in a logger as TEM and Java Class Loader are on.
- Trust-Degree Evaluation. TEM evaluate the loading behaviors to generate diverse trusted-degrees, which provides reference criteria to take next action.

IV. Design and Implementation

The most significant shortcomings of traditional integrity measurement is the coarse grain manner, which is far from the need of high-assure process events. And traditional ways always ignores the availability of the application, that is , it could not perfectly support that service has been running without incurring downtime even though it fails to prove itself own integrity. In addition, it is out of so performance that is hardly to take log in account.

We present TEM which is a novel measurement method in Java Virtual Machine to solve above problems and have as less impact on system performance as possible. It expends the trust chain to the isolated Java applications, which even TPM does not.

A. *TEM Architecture*

The architecture of TEM are shown in Figure 3. A TEM at least has eight core components. The components include:

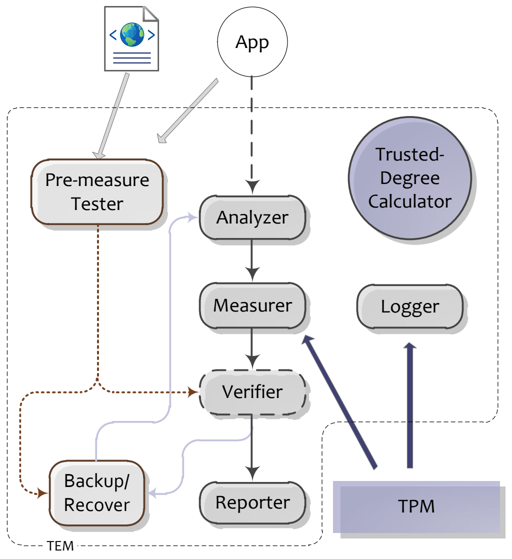

Figure 3. TEM architecture.

- Pre-measure Tester. This components parses an application as well as its own digest table to guarantee it against tampered by malicious software and prove its source to be trusted. If some passes, Pre-measure Tester would store the digest table into a secure storage media. At the meantime, it drives Backup/Recover to safely copy an application into a trusted position.
- Analyzer. It is responsible to interact with Java Class Loader and/or Backup/Recover component at the loading time. It is to get byte streams from them.
- Measurer. This component invokes the underlying TPM's functional interface to enhance itself security. It makes use of JNI, which Java provide to interact with C or C++ to improve the Java execution environment performance. We employ JNI to encapsulate TSS to produce trusted Java API to support Measurer and Logger.
- Verifier. By using the digest value of the loading class produced by Measurer and the reference value from the digest table, it verifies whether the loading class is trusted. If the result is not integrity at the first time, it enforces Backup/Recover component to redirect the resource's path to re-load this class.

Once it fails to recover the classes, it would stop current load process and feed failure information back to Reporter.

- Backup/Recover. This component has two core functions: one is that backups some application proved trusted by Pre-measure Tester; the other is that supports recovery mechanism for the loading class which is verified to be untrusted for the first time.
- Reporter. This component accepts Verifier about measurement results and generate an integrity report with respond to the users' attest request.
- Logger. In company with underlying TPM, it records certain application's loading activities securely, and is able to respond others' attest requests including users and remote party and so on.
- Trusted-Degree Calculator. In the traditional fashion, trusted computing looks upon the relationship of trust on an all-or-none problem. Nevertheless, in the real world, is a dynamic, on-changing relationship. This component is used to calculate the application's trusted-degree in a dynamic way, based on the behavior about loading and recovery.

We offer trusted Java API to Measurer for two aspects of reasons: for one thing, it can have as less influence as on execution performance; for another thing, it has power to achieve more security in comparison to directly invoking the API about security from Java.

B. *Trusted-Degree Estimate*

Trust is a hypothesis about a future behavior, inferred with the current state of certain application. As a result, we take into account the loading result to calculate the trusted-degree. The loading result may include one of the situations: fail to recover (failure), recover successfully (recovery), load successfully without recovery (success).

For estimating the trust-degree of some application, we build a three-level model. As depicted in Figure 4, the root node of the model stands for the entire trusted-degree T of an application. The trusted-degree $T \in [0,1]$, where 1 means the application can be trusted absolutely while 0 means the application is untrusted [10]. The children-nodes of the root node represent the jar file beliefs that belongs to the application. The leaf nodes of this model is fine-grained class files, which belongs to certain jar file.

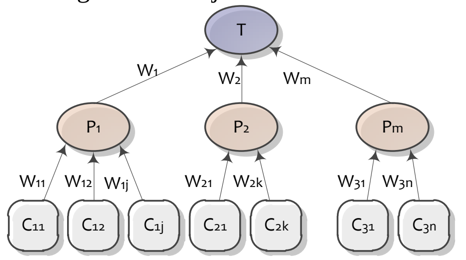

Figure 4. Trusted degree model

The trusted-degree value of the application is given by these formulas:

$$T_{app} = \sum_{1}^{m} W_i \times T_{P_i}$$

$$T_{P_i} = \sum_{1}^{n} W_{ij} \times T_{C_{ij}}$$

W_i is the weight of each jar file's belief;

W_{ij} is the weight of each class file's belief.

When estimating the trusted-value of a class file, we use this formulas as the basic mathematic tool.

$$T_{C_{ij}} = \begin{cases} 0 & failure \\ 0.5 & recovery \\ 0 & success \end{cases}$$

C. *Test Results*

This test is based on the Java Virtual Machine, which is OpenJDK version 6, running on an Intel Core^{TM} i3 2.20GHz machine with 2GB of RAM, and Debian 6, with TPM Specification Version 1.2. We did some functional and performance tests to determine the achievement of TEM.

The functional test's details are shown in Table I and Table II. We use the test suit to do three groups of experiments. From the table, we see that JVM with TEM can effectively identify and locate the Jar package and its own classes which are modified.

TABLE I. TEST SUITS FOR TEM

Test Suite	**Usage**
Tomcat	Application container
Netbeans	Integrity development environment
CXF	Java Web Service
Hibernate	Data persistency framework
Struts	MVC framework
Spring	Inverse of control framework

TABLE II. FUNCTIONAL TEST

description	**Times**	**Accuracy**	**Availability**
Jar package is modified.	1000	100%	100%
One class is missing.	1000	100%	100%
A bit in some class is modified	1000	100%	100%

We use Java Grande benchmarks to test performance impact which our TEM brings to JVM at the loading time. For this, two different kinds of experiments is designed. First of all, we make TEM unable to gain the loading time. In this situation, we run the Serial benchmark of Java Grand's Section 3 benchmarks. On the other hand, we make JVM along with TEM in the same condition. The test results are given out by Table III. In this table, we see that it has a significant increment in the loading time when JVM works with TEM compared with original JVM. Yet, it only occurs to this loss at the loading time and it has little influence on the running performance. it is acceptable just because it is at the seconds level, which users hardly sense.

TABLE III. PERFORMANCE COMPARISON AT THE LOADING TIME

benchmark	original JVM (s)	JVM with TEM (s)
Euler	0.01868651	1.82454909
MolDyn	0.02656407	1.83402396
MonteCarlo	0.02268057	3.76024453
RayTracer	0.02087416	3.09892023
Search	0.02184764	3.26334720

Also, we use SPECjvm2008 to determine the impact that our TEM brings to Java applications at the running time, which takes the applications' loading time into consideration. We design two opposite experiments to do this test. First, original JVM is used to get operations every minute of every benchmark in SPECjvm2008. Second, we make JVM with TEM to achieve operations every minute of all benchmarks of SPECjvm2008. Figure 5 show the details about these results.

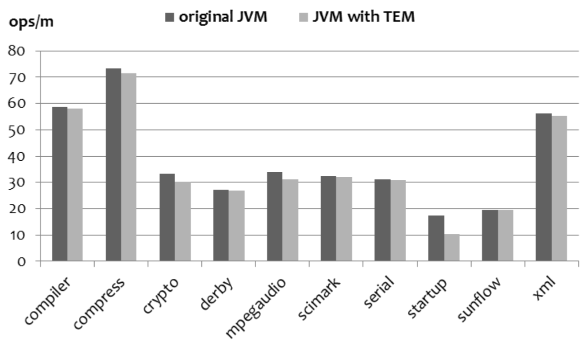

Figure 5. This is performance comparison for the running applications between original JVM and JVM with TEM. This takes the loading time and the runtime of Java applications altogether into account.

From Figure 5, we find that JVM with TEM could strengthen trusted protection to top Java applications with the accepted loss. Even, the TEM hardly affect the running performance of the Java applications. In comparison to the performance overheads, the extra loading time along with TEM is too little to having an effect on the running applications' performance.

V. RELATED WORK

A large amount of research has been done on trusted computing, mostly on the improvement of trusted computing itself or Java applications.

There is some prior work that aims to make the Java applications more trusted. In Ref. 5, Liang and his partners introduce dynamic class loading in JVM, with many advantages, such as type-safe linkage which guarantees type safety [5]. Though this support for Java is powerful, it does not have a mechanism to measure and verify Java class integrity with more formidable hash algorithm; for instance, SHA-256, SHA-384 and SHA-512. In Ref. 9, Shi et al. propose transitive trust in Java Virtual Machines, which extends trust chain to JVM and enhances Java application's trust by class loaders [9]. However, it concentrates more on control flow of class loading in JVM and does not make full use of measurement technology.

VI. CONCLUSION

Java Virtual Machine ensures that class is loaded on demand and class is not loaded by the same class loader again. According to the features of the Java class file format, we design TEM to realize the measurement mechanism on demand to support Java applications integrity guarantee in cloud computing. In other words, they would not be measured once again. To improve the top applications' availability in cloud computing , we provide trusted backup mechanism for recovery at the loading-time. With the secure log, it make the system audit at the convenience.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China (NSFC) under grant No.60903204.

REFERENCES

[1] Shen changxiang, Zhang Huanguo, Feng Dengguo, Cao Zhenfu, Huang Jiwu. Survey of Information Security, Science in Chian Series F, 2007, 50 (3): 273-298.

[2] Shen Changxian, Zhang Huanguo, Wang Huaimin,Wang Ji, et al. Researches on Trusted Computing and Its Developments[J]. Science China: Information Science, 2010, 53 (3):405-433.

[3] B. Venners. Inside the Java 2 Virtual Machine, Second Edition. Jan. 2002.

[4] L. Gong, G. Ellison, M. Dageforde. Inside Java^{TM} 2 Platform Security: Architecture, API Design and Implementation, Second Edition, Jan 2003.

[5] S. Liang, G. Bracha. Dynamic Class Loading in the Java Virtual Machine. Proceeding of the ACM Conference on Object Oriented Programming System, Languages and Applications, Vancouver, British Columbia, pp.36-44, October 1998.

[6] P. Mell and T. Grance, The NIST Definition of Cloud Computing version 15, http://csrc.nist.gov/groups/SNS/ cloud-computing, National Institute of Standards and Technology. Gaithersburg, MD, 2009.

[7] Trusted Computing Group. TPM Specification Version 1.2. http:// www.trusted-computinggroup.org/resource/tpm_main_specification. Trusted Computing Group, 2007.

[8] D. Challener, K. Yoder, R. Catherman, D. Safford, L.V. Doorn. A Practical Guide to Trusted Computing. 2007.

[9] Y. Shi, Z. Han, C.X. Shen. The Transitive Trust in Java Virtual Machines. Proceedings of the Eighth International Conference on Machine Learning and Cybernetics, pp.12-15, July 2009.

[10] Jigar Patel, W. T. Luke Teacy, Nicholas R. Jennings, and Michael Luck. A Probabilistic Trust Model for Handling Inaccurate Repution Source. Trust Management, Third International Conference, iTrust 2005, Paris, France, 193-209, May 23-26, 2005, Proceedings.

[11] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture [C]. In Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA, pages:223-238.

[12] Robinson, N., Valeri, L., Cave, J., Starkey, T., Graux, H., Creese, S., Hopkins, P.: TheCloud: Understanding the Security, Privacy and Trust Challenges. RAND Corporation, California , 2011.