Dual-mode Integrity Measurement System Based on Virtualized TEE

Duan Jialiang\textsuperscript{1*}, Cai Guoming\textsuperscript{1}, Xu Kaiyong\textsuperscript{1}

\textsuperscript{1}PLA Strategic Support Force Information Engineering University, Zhengzhou, Henan, 450001, China

\* Corresponding author’s e-mail: Julian.k4558@gmail.com

Abstract. The Trusted Execution Environment (TEE) is mainly used to protect sensitive information and ensure information security. TEE has been used in the system structure of various mobile devices. With the development of TEE, attack methods against TEE gradually appear. The virtualized TEE architecture proposed by existing research can solve some security problems caused by TEE architecture, but there are still some deficiencies. To enhance the TEE architecture, a virtualized TEE-based kernel integrity measurement scheme is proposed, and identity-based and neural network-based integrity measurements are used in different scenarios. The experimental results show that the designed dual-mode integrity measurement system can effectively detect attacks against different kernel objects and has little impact on system performance, thus achieving security enhancements for virtualized TEE.

1. Introduction
In the past decade, due to the rapid development of the Internet, the use of most electronic products has changed dramatically, such as smart phones, laptops and tablets. Hundreds of millions of electronic devices manage and monitor public resources such as transportation system, industrial system and power system, and also record all kinds of privacy data, such as personal health, online consumption, social networking and other online business. Therefore, the safety of electronic equipment is very important. Among them, the security of mobile intelligent terminal is the top priority in the field of electronic equipment security, so how to design effective mobile terminal security protection has become the focus of research.

Mobile device security mainly includes system security, communication security, component security, data security and so on. The operating system of modern mobile devices is becoming more and more powerful, which can install and run third-party software and provide various services such as data storage and processing. As a result, the operating system is often the target of vulnerabilities and malware attacks, and is facing huge security threats [1-6].

In 2010, the global platform organization GP (Global Platform) proposed the concept of Trusted Execution Environment (TEE), which is a security zone where all kinds of mobile smart devices are located on processors. The security, confidentiality and integrity of code and data can be guaranteed by loading into this environment. TEE also provides an isolated execution environment, enabling the system to achieve the goal of application and data security assurance. However, with the widespread use of TEE on mobile and embedded devices, and the deployment of more and more TCBs, attacks against TEE have gradually emerged. The current attacks mainly focus on the system security of TEE. In order to solve the system security problems, this paper mainly studies the TEE architecture. Li proposed a virtualized TEE solution, which solves the security problem that the existing TEE has too
high privileges[7]. Although it can ensure that TEE will not affect other systems and users after being attacked, but can not control the overall security condition of the device, the attacker can still operate the damaged system, leak sensitive data or continue to occupy hardware resources of the device.

This paper takes TEEv as an example, deeply studies and compares the implementation of TEE related technology and its security mechanism, and summarizes the advantages and disadvantages of different TEE technologies. security enhancement scheme for TEEv virtualization architecture is proposed, which verifies the integrity of TOS and ROS in the architecture, and dynamically measures the operating system in running state. Finally, for different application scenarios, two measurement modes are proposed that can be switched at system startup to achieve different security levels in different scenarios.

2. Overview of TEE

2.1. TEE Architecture

The trusted execution environment first appeared in OMTP[8], and proposed a two-system solution: to provide an additional isolated secure operating system within a smart terminal, and at the beginning of the design, this system was run on isolated hardware to specifically process sensitive information to ensure the security of information. Then, in July 2010, the Global Platform International Standards Organization announced a complete set of TEE system standards, which will run the common world on the same set of hardware devices as the rest of the world. The structure is shown in Figure 1.

![TEE Architecture Diagram](image)

The TEE architecture is mainly composed of the following components [9]: REE for Common Execution Environment, CA for Client Application, Functional API, Client API, Trusted Execution Environment, Trusted Application TA, TEE Internal API, and TEE Trusted Functional API.

2.2. Existing TEE Attacks

Since the introduction of TEE and related technologies, it can be roughly summarized into three stages. Phase 1: TEE is used to start a secure operating system and to detect the effectiveness of Rich OS. Phase 2: TEE supports different highly sensitive permissions and functions, such as encryption operations, biometric authentication, mobile payments, digital rights management, and so on. Phase 3: TEE can support TA dynamic installation, and for developers to more easily develop CA and TA, GP proposed a set of APIs for TEE and REE communication. Moreover, ARM and some TEE vendors have created OtrP (Open Trust Protocol) to combine TA management with security architecture [10].

Currently TEE is in the third stage, TEE systems and related technologies have been greatly developed, TA can be installed in Trust OS as a normal application like CA. At present, most TEE
systems have only one available TEE kernel, so all TAs can only choose to consider the TEE kernel trusted.

As a security architecture based on isolation idea on smart mobile devices, TEE CA cannot directly access any resources on the TEE side under normal conditions, and all hardware resources including CPU are isolated from the REE side when the device is in a secure state. However, this isolation mechanism does not use two sets of physical devices, such as CPU isolation through time division and memory allocation through different memory addresses and memory snap-in MMUs. Moreover, due to the incomplete TEE system standards proposed by GP organization, there are still security risks at the architecture level.

Therefore, TEE has developed so far, there are also many ways to affect its security, mainly divided into hardware and software attacks. Hardware attack modes include hardware bypass attack, hardware fault injection attack, bus detection attack, physical storage export attack, etc. Compared with hardware attack, software attack does not need to touch mobile devices, so the attack is more convenient. The main ways are software bypass attack, fault attack, command exception attack, firmware rollback attack, code injection attack and so on. The vulnerabilities resulting from these attacks are divided into three basic categories.

1. TEE vulnerabilities can lead to sensitive data leaks or arbitrary code execution. For example, CVE-2017-0518/0519, CVE-2016-2431/2432, and so on.
2. TEE vulnerability will further affect the security of other TAs after one TA in Trust OS is attacked. For example, CVE-2016-0825, CVE-2015-6639/6647.
3. TEE vulnerabilities lead to the use of related functions within TEE to give REE higher privileges and even direct control of hardware to leak or manipulate sensitive information. For example, CVE-2016-8762/8763/8764, etc.

In order to protect TEE from attack, you need to enhance the TEE's own security or use a more secure structure. This paper mainly studies the TEE solution based on virtualization structure, analyses its advantages and disadvantages, and makes security enhancements and improvements for the TEE virtualization structure.

3. TEE based on virtualization architecture

3.1. TEE System Threat Analysis

The Attack Tree Model is a model proposed by Bruce Schneier in 1999 to identify the interdependence between the attacking behavior and the steps using a tree structure [12], which is used to establish a model to describe the security state of the system. The target an attacker wants to reach is identified by the root node in the model, while the sub-target to reach the total target is represented by the sub-node, and the attacker's attack method is represented by the leaf node of the tree structure. This modeling and analysis method has the advantage of clear structure, exposing the essence of attack means, and the data representation of attack tree is flexible and reusable [13].

The relationship between nodes of an attack tree is divided into three types: or, and, and sequential and. At the same time, leaf nodes are instantiated by different specific events in different environments, as shown in Figure 2.

There are three main sources of security issues mentioned in Section 2.2 of this paper: large TEE attack surface, isolation between the world, and memory protection mechanism. All three can cause TEE security problems. An attack tree model is mapped from the analysis of the three, as shown in Figure 3. The nodes in the tree are divided into target nodes (shadow nodes), state nodes (dashed nodes), and event nodes (solid nodes). The specific meaning of each node is shown in Table 1.
As shown in Figure 3, an attacker needs to achieve Target A, which raises security issues for TEE, by raising a series of architectural issues (BCDs). For example, in order for an attacker to cause a C problem, an attacker needs to agree that TA can map physical memory attack REE (E) or sensitive information that can be leaked to the general world (F) through debugging channels. Attackers need to generate malicious TA (a) to load into TEE or by destroying normal TA (b), which requires TEE privilege upgrade (e) to gain the privileged identity of TEE system (f), such as kernel privilege, by taking advantage of existing TA vulnerability (d), and then upgrade the privilege to get TEE kernel privileges.

After analyzing the threat sequence, we can get the threat sequence expression of TA mapping physical memory attack REE threat final target (E), which is composed of atomic events. In order to reach the E threat, we need to construct \{defbc\} and \{a\} threat sequences.

After model analysis, it can be seen that the unsafe of TEE architecture is mainly due to the high privileges of TEE, which can control all devices and systems completely, while TEE can not avoid increasing its functionality, causing TCB to be too large and then expanding its attack surface. Therefore, the introduction of virtualized TEE will solve a series of security problems caused by high TEE privileges.

### 3.2. TEEv System

To solve the many security problems TEE faces, LI proposed TEEv, a virtualization-based TEE solution (hereinafter referred to as TEEv) [7].

#### 3.2.1. TEEv System Overview

TEEv mainly solves the vulnerability of TEE from the virtualization structure. The main idea is to introduce a TEE-visor mini-manager to replace the monitor in TrustZone technology and to allow multiple isolated TEE instances (hereinafter referred to as vTEE) to run simultaneously.
In TEEv, TEE-visor has the maximum privileges and is considered the TCB of the whole system, assuming that its boot process is protected by hardware and that it will not be compromised by other attacks after boot.

In the management of vTEE, the idea of zero trust is implemented. vTEE does not trust any other vTEE but only some of the functionality of the system vTEE provided by the mobile device manufacturer. In each vTEE, vTEE does not trust any TA, and TA does not trust each other.

Since multiple vTEEs are enabled in a single device, these vTEEs are isolated from each other and from the REE at the same time. Therefore, TEE-visor was introduced as a small TEE manager, running at the highest privilege level logically, responsible for all resource management, memory mapping, and the allocation of all peripherals. TEE-visor also manages the installation, update and deletion of third-party vTEEs.

3.3. Security issues with TEEv systems
Virtualization-based TEEv solutions reduce TEE privileges by introducing a mini-manager and multiple isolated vTEEs. At the same time, TEEv allows different manufacturers to join their own TEE designs, making the TA ecosystem more dynamic and open. On the security side, TEEv makes one TEE or TA under the control of an attacker and does not affect the security of other TAs or the entire device.

Although the TEEv architecture reduces the permissions of each TEE and isolates them from each other, the existing TEE vulnerabilities still have the following shortcomings: 1. TEEv only solves the problem that TA or TEE will affect other TEEs or the whole device after it is attacked, users may still use damaged TA or TEE, and sensitive data within their permissions is still not protected. 2. Although TEE-visor has the highest privilege level to control all hardware resources, it is still unable to know whether all vTEEs are running safely. As a result, it is unable to handle the attack, which may cause damaged vTEEs to continuously request hardware resources and make the device unusable.

Therefore, this paper presents a security enhancement scheme to solve the problems of the above TEEv architecture.

4. Security enhancements for TEEv
There are still some drawbacks in the TEEv architecture mentioned above, which can be summarized as follows: TEEv monitors cannot monitor the security status of all TEEs or REEs running in the device. To solve this problem, it is necessary to let the monitor know the running status of all systems. Therefore, the enhancement scheme proposed in this paper is to make the system detect attack behavior faster and control the running state of the system by adding a dynamic measurement system.

4.1. System Structure
Static Integrity Measurement refers to the measurement of a pre-set goal at a specific time to obtain the characteristic information of the target, and to compare the measurement result with the initial set standard value to determine whether the integrity of the target has been compromised, so it is also called Integrity Measurement [14]. Dynamic integrity measurements refer to real-time integrity measurements of running operating systems to enable more timely detection of attacks and to maintain system security. Based on the analysis of the security issues of the virtualized TEE system, the design of the kernel integrity measurement structure is carried out on the basis of the virtualized TEE structure. The overall design idea of the system is shown in Figure 4.
As can be seen from Figure 4, the system design in this paper can be divided into two main measurement methods according to different usage environments when the system is running normally: the measurement method for private environment and the measurement method for general environment. Private environments are scenarios where the system is relatively fixed and dynamic installation of new trusted applications or security drivers is not allowed. A common environment is a scenario in which a system allows dynamic installation or trusted applications, or updates.

The difference between the different measurement methods is that because of the stability of the system in the private environment and the fixed application and system information, the measurement method for the private environment uses the Benchmark-based measurement method, which calculates and saves the benchmark value of the measurement object for the initial state of the system, and then completes the measurement operation by comparing the calculated identity value with the benchmark value. The general environment-oriented measurement method treats the memory content of the measurement object as the feature value, then trains the feature value as the input layer of the neuron, and finally trains the trained model. Calculate the security status of the system and complete the measurement operation. Switching between the two environments mainly involves selecting the measurement mode that needs to be enabled before the system restarts, and then using different measurement modules for measurement. Key implementations and measurement processes for both environments are described later.

However, in any scenario, the design focus of the measurement method in the integrity measurement system is mainly three aspects:

1. the measurement object. Integrity measurement objects can be divided into the following seven objects in three stages of design thought: kernel code, loadable kernel module, read-only data segment, initial segment, exception table, system call table, data invariant.

2. Position of measurement. The position of the measurement is set by the measurement object within the Kernel Load Module tool function, at the beginning of the data segment, initial segment, and data invariant memory, and at the entry address of the exception and system call tables. Different types of measurement operations can be performed on the measurement objects.

3. Position of measurement system in TEEv. In the TEEv virtualization structure, TEE-visor is given the highest privilege level, so the measurement system should be running inside TEE-visor as resident software. Because TEE-visor has a complete memory view of the mobile device, the measurement system can get the running status of the measurement object through the memory information of different TEEs and REEs.

4.1.1. Private Environment Critical Implementation. In dedicated mode, the main method is to compare the identifier value of the measurement object with the initial benchmark value of the system. If it is not equal, the integrity of the system is considered to be compromised.
In order to preserve the initial benchmark values of the system, the most important thing is to encrypt the identity values. In virtualized TEE, encryption is required through system-TEE and stored in the REE system security store. The baseline encryption process is shown in Figure 5.

FEK is the file encryption key, which is generated by the key manager through random number generation. TSK is the application key required by TA in each TEE environment and is mainly generated by SSK (Secure Storage Key) identifier (UUID) in TA. The TSK is generated to encrypt and decrypt file encryption keys.

The original encrypted data is the security file stored in REE security storage after encrypting the base value. When using it, the encryption process needs to be reverse decrypted to get the original base value and the identity value obtained by the measurement module.

4.1.2. Universal Environment Critical Implementation. In a general environment, the measurement method is mainly to migrate and learn the trained neural network model, and add the measurement object to the model for calculation at runtime to obtain the security state of the system to achieve the integrity measurement of the system kernel.

In this paper, we need to judge the system integrity by the eigenvalues of the in-memory data, so we choose the fully connected network to calculate for the application of the neural network in pattern recognition. A neural network model is established as shown in Figure 6.

\[ X_i \] is the eigenvalue of the selected kernel measure object in memory data, while the number of hidden layer ganglion points is calculated by classical algorithm [15]. Output layer \( Z_1 \) and \( Z_2 \) represent the probability of system security and unsafe, respectively.

The effect of each memory feature on system security is bi-categorized, and in order to speed up the convergence of the model in training, the activation function from the input layer to the implicit layer selects ReLU. The implied layer to the output layer requires the output security probability, so the
Softmax activation function is used. The training set mainly uses common rootkits to test multi-object attacks and build attack datasets against the measured objects.

4.2. Process of measurement

In the design of this system, because TEE-visor is the highest privilege level and is in the security state by default, the key part of designing software-based kernel measurement system should be in TEE-visor, which uses its high privilege and high confidence to measure the kernel integrity of vTEE and REE. There are three phases to achieve the goal of measuring the integrity of the operating system core.

1. Safe boot and start. In the TEEv architecture, TEE-visor exists as a TCB, assuming that it is started with hardware protection, and TEEv will not be damaged afterwards. Therefore, the security boot and boot of TEE-visor are not considered. The subsequent addition of vTEEs requires integrity verification through the measurement system to ensure that they are not tampered with. That is, the identity values of the original files provided by the vTEE vendor are encrypted and compared, and if they do not match, the vTEE is rejected.

2. Measurements when the kernel loads modules.

Because security bootstrapping establishes the integrity trust chain from the hardware to the highest privilege manager for TEEv, it ensures that the basic kernel has good integrity. Therefore, when a kernel loads a module, it is only responsible for measuring the system integrity when a loadable kernel module loads. At this point, the measurement process should be divided into measure operations when the kernel module is acquired and loaded.

Module acquisition is mainly through the need for TEE vendors to provide module information that needs to be loaded when a new TEE needs to be dynamically installed. If the module information for the corresponding identity does not exist, the module is marked as untrusted and as the focus of monitoring, and its identity is saved as a baseline value.

3. System Runtime Monitoring. After the security boot starts TEE-visor and vTEE, the system goes into normal operation. At this time, it is necessary to monitor the running status of vTEE and REE during the running process, collect the measurement characteristics of operating system running status information, and timely discover and report the behavior that damages system integrity according to the system measurement policy, and take corresponding protective measures. Runtime system monitoring and measurement targets operating system critical resources, including: kernel code (basic kernel and loadable kernel modules), system call tables. Interrupt vector table, Kernel function pointer, etc.

After analysing the integrity of the operating system kernel [16], the main reasons why the integrity of the operating system is destroyed during operation are that the objects related to the operating system kernel are destroyed and the behavior of the operating system is changed. The factors that can affect the integrity of the operating system are divided into two main categories: data integrity and control flow integrity:

(1) Key resources that affect data integrity: the service behavior of the operating system depends on related data objects, and the effect of service functions is partially reflected in these related data objects. These data become key data, including process control blocks, page tables, file index nodes, etc.

(2) Key resources that affect the integrity of the control flow: the behavior of the operating system is determined by the code being executed. The execution path and sequence of the code must be correct and complete. Unlawful code cannot be executed first, and then the code cannot be executed in an illegal order. Key resources that affect the integrity of control flow include: a) Kernel code: including basic and loadable kernel module code. b) System Call Table and Interrupt Vector Table: Switching from user space to kernel space is mainly done by system interrupt and system call, so the entry of code in kernel space is also defined in the Interrupt Vector Table and System Call Table. c) Kernel function pointer: There are a large number of function pointers in subsystems such as the network of the kernel and virtual file system, which define the interface between the kernel and each module. The three key resources mentioned above are the key factors that mainly affect the control flow. Therefore, in order to ensure the integrity of the control flow, the above critical resources need to be protected from attack or tampering.
The overall system measurement process is shown in Figure 7.

![Diagram of system measurement process]

5. Experimental results

This paper implements the development and testing of dynamic integrity measurement system for TEEv based on OKA40I-C development board of quad-core processor A40i Cortex-A7. The TEE system uses OP-TEE, which has been widely used by open source. The REE system uses Android 4.1.2. The system baseline values are obtained and saved in the version 3.8.7.4 SQLite database after the system is installed.

5.1. Functional testing

By loading malicious programs, adding malicious and hidden kernel modules, modifying existing kernel modules and adding hidden processes, the integrity measurement system detects the results as shown in Table 2. In addition to the data invariants, the detection accuracy of the main attack test objects is more than 90 percent. It can be concluded that the integrity measurement system has a good detection capability for kernel attacks.

The average detection accuracy of the two modes is detected using two modes, and the detection accuracy of the two modes is similar in different use environments. This shows that the two measurement methods can ensure their functionality in different measurement environments.

| Attack target              | Total number of attacks | Dual mode average detection accuracy |
|----------------------------|-------------------------|-------------------------------------|
| Kernel code snippet        | 60                      | 95%                                 |
| Read only data segment     | 35                      | 91.42%                              |
| System call table          | 50                      | 98%                                 |
5.2. Performance testing
In order to study the impact of integrity measurement system on system performance, a time function is added to load test module and test application to get the start time, which is compared with the time before and after the experiment. The experimental results are shown in Table 3.

| Test objectives          | Measurement time of private mode | Measurement time of general mode |
|--------------------------|----------------------------------|----------------------------------|
| Kernel code snippet      | 0.04                             | 0.06                             |
| Read only data segment   | 0.12                             | 0.18                             |
| System call table        | 0.03                             | 0.03                             |
| System exception table   | 0.03                             | 0.03                             |
| Loadable module          | 0.09                             | 0.15                             |
| System process           | 0.06                             | 0.08                             |

From the data in the table, it can be concluded that in private mode, the average measurement time is less than 0.1s except for read-only data segments, which has little impact on the overall system operation, and can further reduce the performance impact by reducing the number and frequency of measurement points. In the general mode, since the neural network model is used for calculation, it takes a little longer, but is less than 0.2s, and the performance impact is still low.

6. Conclusion
In this article, we do in-depth research on TEE and vTEE, summarize the existing problems and deficiencies that vTEE solves, introduce a dynamic integrity measurement system to monitor the running state of vTEE in real time, and propose a dual-mode measurement method for different usage environments, so that its security can still be guaranteed in different usage environments. And the measurement system has little influence on the overall device performance, but does not affect the normal use. However, the extraction of each measurement content in the system can be further improved to enhance its accuracy and measurement speed, and its security and performance can be enhanced later by refining the measurement content. And the model of the neural network can still get better model training result by enlarging the training of the dataset.

References
[1] Scott King. (2017) Mobile Device Threat Data–Q1 2017. https://blog.zimperium.com/mobile-device-threat-data-q1-2017.
[2] CVE. (2008) VMware buffer overflows in VIX API let local users execute arbitrary code. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2100.
[3] MR Wigan .(2015) Android Stagefright vulnerability threatens all devices – and fixing it isn't that easy[J]. Network Security, 2015(8).
[4] Sylvie Liu, Security Program Manager, MSRC Community Programs. (2019) Announcing 2019 MSRC Most Valuable Security Researchers https://msrc-blog.microsoft.com/2019/08/07/announcing-2019-msrc-most-valuable-security-researchers/
[5] Secunia. (2019) Xen multiple vulnerability report. http://secunia.com/advisories/44502/ .
[6] R. Wojtczuk and J.(2018) Rutkowska. Xen Owning trilogy. Black Hat conference . http://www.blackhat.com/.
[7] Li, W. , Xia, Y. , Lu, L. , Chen, H. , & Zang, B. . (2019). TEEv: virtualizing trusted execution environments on mobile platforms. the 15th ACM SIGPLAN/SIGOPS International Conference. ACM.
[8] OMTP. (2009). ADVANCED TRUSTED ENVIRONMENT:OMTP TR1. https://www.gsma.com/newsroom/wp-content/uploads/2012/03/omtpadvancedtrustedenvironmentomtptr1v11.pdf

[9] GlobalPlatform Device Technology. (2010) The trusted execution environment. https://wenku.baidu.com/view/ad503b975ef7ba0d4b733b39.html.

[10] ARM. (2016). Connected devices need e-commerce standard security say cyber security experts. https://goo.gl/1ePiQC.

[11] Zheng XY, Li W, Meng D. (2016) Analysis and Research on TrustZone technology. Acta computa Sinica, 39 (09): 1912-1928.

[12] SCHNEIER B. (1999) Attack trees: modeling security threats. Dr. Dobb's Journal of Software Tools, 24 (12): 21-29.

[13] Xiang G, Cao Y. (2003) Research on attack tree generation algorithm based on attack classification. Journal of Beijing University of technology, 2003 (03): 340-344.

[14] Liu Z, Feng D. (2010) Dynamic integrity measurement architecture based on Trusted Computing. Acta electronica Sinica, 32 (04): 875-879.

[15] Xia K W, LI C B, SHEN J Y. (2005) An Optimization Algorithm on the Number of Hidden Layer Nodes in Feed-forward Neural Network. [J]. Computer Science, 2005(10): 143-145.

[16] Robert L, Luo F, Chen L, Tang H, Zhang B. (2006). Linux kernel design and implementation. China Machine Press