Trusted Software Constitution Model Based on Trust Engine

Junfeng Tian
College of Mathematics and Computer Science of HeBei University, Baoding, China
Email: tjf@hbu.edu.cn

Ye Zhu and Jianlei Feng
Shijiazhuang Posts and Telecommunications Technical College, Shijiazhuang, China
College of Mathematics and Computer Science of HeBei University, Baoding, China
Email: zhuye626@163.com, snuboy_2008@126.com

***Abstract*—The guaranty of trustiness wasn't considered enough in traditional software development methods, the software developed in that methods lack effective measures for ensuring its trustiness. Combining agent technique with the support of trusted computing provided by TPM, a trusted software constitution model based on Trust Engine (TSCMTE) is demonstrated in this paper, Trust Engine extends the "chain of trust" of TCG into application, and cooperates with TPM to perform integrity measurement for software entity to ensure the static trustiness, then through verifying whether the dynamic behavior of software satisfies the trustiness constraints at runtime, Trust Engine guarantees the dynamic trustiness of software behavior. For the purpose of improving the accuracy of trustiness constraints, a strategy of determining the weights of characteristic attributes based on information entropy is proposed. Simulation experiments illustrate that the trustiness of software developed by the TSCMTE is improved effectively without performance degradation.**

***Index Terms*—Trusted Software Constitution, Trust Engine, Trust Control and Evaluation, Trust View, Software Behavior Trace**

I. Introduction

The increase in software size and the complexity of external environment have resulted in the increasingly descending of software quality. Once software failures and malfunctions occur, especially when software is attacked maliciously, it will bring tremendous loss to people's work and life. Trusted software will not result in malfunction or failure largely even if caused by malicious attacks, or system errors [1]. How to ensure the trustiness of software will be an inexorable trend of software's development and application.

Trusted software constitution technology is an important guarantee of software's correct execution, which ensures that software is always works in the intended way and goes towards the intended direction [2]. Based upon the theory and technology of traditional software, this paper presents a trusted software constitution model based on Trust Engine (TSCMTE). The remainder of this paper is organized as follows: Section 2 covers the previous related works on the research of software's trustiness. Section 3 introduces the framework of TSCMTE, and Section 4 presents the software behavior and trustiness evaluation in detail. Section 5 states the simulation experiments and results. Finally, Section 6 draws conclusion and outlines future extension of this work.

II. Related Works

In recent years, the trustiness of software has drawn more and more people's attentions and large quantities of research achievements have mounted in the field of the software's trustiness [3].

Formal methods [4] ensure the software's trustiness in a rigorous way. Cleaning Software Engineering [5] places the process of software's development in the control of statistic value, which can be used to develop software with high reliability certification. Aspect-oriented programming methods (AOP) [6] can be used to separate the monitoring and controlling for developing software with trustiness, which records the process of software's execution to guarantee trustiness. Qu Yanwen et al. [7] described the trustiness of software by Software Behavior, which is defined by the expectations of software's correct execution, and the trustiness can be classified into several classes. Lin Huimin et al. [8] carried out the formal research on software with high trustiness, through converting the problem that "whether the software has the intended characteristics" into a mathematical problem that "whether the software behavior S satisfies software character F", to ensure that the behavior of software is always consistent with the intended. Wang Huaimin et al. [9] proposed the trustworthiness classification specification of software, mainly included the definitions of the trustiness classes, measurements of trusted evidence and so on. Liu Jing [10] discussed how to integrate the UTP and UML together to form a unified modeling system, which not only makes the MDA technology to be used to constitute trusted software, but also adopts the formal specification of software and techniques of model checking in the software's

doi:10.4304/jsw.7.12.2771-2778

development process to ensure software's trustiness fundamentally.

Through a brief review of the above researches on software's trustiness, it can be concluded that the achievement on software's trustiness have initiated us into the causes of untrusted software, and some countermeasures have been adopted. However, in the field of software's trustiness, the researches on the constitution of trusted software continue to be scarce. To bridge this gap, this paper demonstrates a trusted software constitution model based on Trust Engine (TSCMTE), and aims to:

1. show the framework of TSCMTE by combining agent technique with the support of trusted computing provided by TPM.
2. introduce software behavior and trustiness evaluation in detail, including the guaranty for the static integrity of software with Trust Engine, the definition, representation and extraction of Software Intended Behavior Trace.
3. propose a strategy of determining the weights of characteristic attributes based on information entropy to improve the accuracy of constraints.

III. The Trusted Software Constitution Model Based on Trust Engine

The basic idea of TSCMTE is not only guaranteeing the static integrity of software, but also constraining the dynamic behavior in the process of software execution effectively. The trustiness of software is mainly manifested in the trustiness of static integrity and dynamic behavior of software. On the basis of ensuring the static trustiness of software, the TSCMTE is driven by software dynamic behavior, through monitoring the dynamic behavior in the process of software execution, it can be verified whether the dynamic behavior is always consistent with the intended behavior, then the dynamic behavior of software will be adjusted and controlled possibly to ensure the dynamic trustiness of software.

A. Framework of TSCMTE

The software based on traditional theories faces two typical security threats: first, the static integrity of software is broken probably, suffering from virus, which caused dynamic behavior to be changed; second, software is illegally injected or interrupted by other processes in the process of execution, such as buffer overflow attack, which can change the dynamic behavior of software possibly without breaking the static integrity.

Therefore, in order to identify the above-mentioned security threats and ensure software's trustiness, combining agent technique with the support of trusted computing provided by TPM, the framework of TSCMTE is proposed, as shown in Fig. 1.

The reasons why the TSCMTE is proposed are listed as follows:

1. Trusted computer based on TPM is of temporary trustiness in the initial phase [11]. When software is running, the trustiness of dynamic behavior of software can't be guaranteed if the static integrity has been broken.
2. It is effective for ensuring the dynamic trustiness to monitor and constrain the dynamic behavior of software. Inject the capability of monitoring in an appropriate way, and then the entity agent [12] can autonomously monitor the dynamic behavior and extract the related information with its context.

The TSCMTE is made up of Application Software, Trust Engine, TPM and Operation System.

1. Application Software: the source of software's dynamic behavior.
2. Trust Engine: extends the "chain of trust" of TCG into application, and cooperates with TPM to perform integrity measurement for software entity to ensure the static trustiness, and based on agent technique, through monitoring and extracting the dynamic behavior of software, it can verify whether the dynamic behavior satisfies the trustiness constraints, then adjust and control the software behavior to ensure the dynamic trustiness. It is composed of Trust Monitor, Context, Trust Control and Evaluation, Trust View and Trusted Communication Agent Interface.

(1) Trust Monitor: it can monitor the process of software execution to extract related information.
(2) Context: it refers to the necessary conditions for the operation and interaction of software, which includes the time, environmental factors and other related information. Whether the behavior is trusted or not is closely related to the specific context.

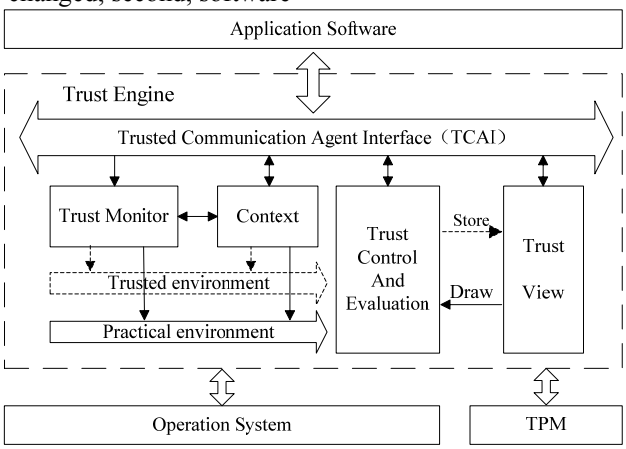

Figure 1. Framework of TSCMTE.

(3) Trust Control and Evaluation: it's responsible for trustiness evaluation, and then it can adjust the dynamic behavior and take appropriate measures to control anomalous behavior according to the result of evaluation.
(4) Trust View: it's defined to represent the characteristics of software behavior
(5) Trusted Communication Agent Interface (TCAI): it's a security bus actually which is of trustiness, privacy and integrity [13] and responsible for communication with other modules. That is, it can guarantee the security of identity, the transmission of message invisible to other processes and can't be modified unauthorizedly, etc.

Trust Engine as the core of TSCMTE interacts with other modules to ensure the trustiness of software.

3. TPM: trusted computer based on TPM can be of temporary trustiness in the initial phase.

The TSCMTE focuses on the logical framework and the design of Trust Engine.

B. Trust Engine

For the openness of operational environment, there is the security threat to the static integrity of software inevitably. Combining "chain of trust" of TCG with the support of trusted computing provided by TPM [14], the TCG sets up a root of trust in computer system and builds a chain of trust, which starts from root of trust to hardware platform, and operating system. Trust Engine in TSCMTE provides effective measurements to constrain the static integrity of Software Module, and then passes trust. Therefore, trust can be extended into the whole computer system. The chain of trust of TCG with Trust Engine is shown in Fig. 2.

When Application Software is loading, Trust Engine interacts with TPM firstly to check the static integrity, and then trust can be extended from the root of trust into Application Software. After that, the dynamic behavior of Application Software in the process of execution can be monitored and extracted to ensure the dynamic trustiness.

C. Trust Control and Evaluation

Application Software is the source of Software Behavior. The dynamic behavior of software monitored by Trust Monitor needs to be verified the consistency with the intended behavior. Therefore, the intended behavior as a benchmark of evaluation is the essential prerequisite of dynamic trustiness constraints. The Context extracted during software execution can guarantee objectivity and credibility of software behavior.

The Trust Control and Evaluation proposed is made up of Trusted Network Interface, Event Channel, Evaluation and Trust Control. Its framework is shown as in Fig. 3:

1. Trusted Network Interface: the software based on network interacts with other entities through the Trusted Network Interface.

2. Event Channel: receive the related information which extracted through monitoring the process of software execution.

3. Evaluation: verify whether the dynamic behavior satisfies the trustiness constraints.

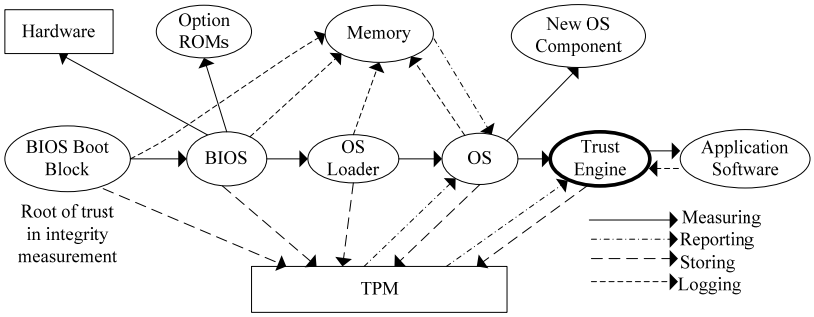

Figure 2. Chain of trust of TCG with Trust Engine.

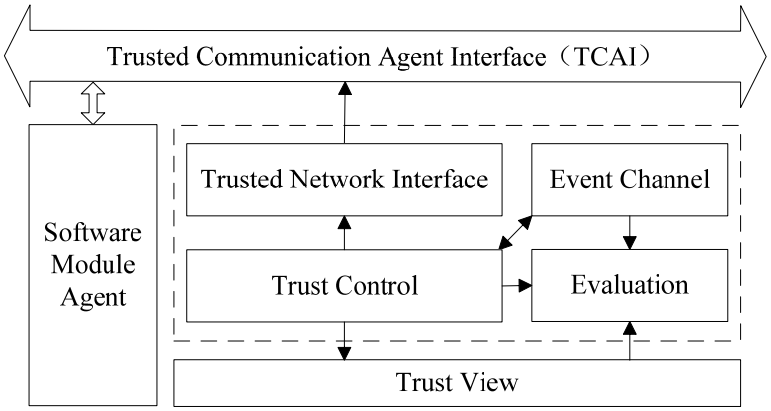

Figure 3. Framework of Trust Control and Evaluation.

4. Trust Control: according to the evaluation result, adjust dynamic behavior and take appropriate measures to control anomalous behavior.

IV. Software Behavior and Trustiness Evaluation

A. Related Definitions

The definition of trustiness is given by TCG according to the behavior of the entity: An entity can be trusted if it always behaves in the expected manner for the intended purpose [15]. Therefore, as an entity, the trustiness of software depends on whether its behavior is trusted or not. The related definitions are listed as below:

Definition 1. Software Behavior: Software behavior refers to any changes, influences or any operations made to the other independent entities when the software works as an independent entity [7], that is, software is able to perform its function by consuming computer resources.

Definition 2. Trustiness of Software Behavior: If the dynamic behavior in the process of software execution is always consistent with the intended behavior, it can be considered as trusted.

Definition 3. Software Intended Behavior Trace: it is the representation of intended behavior of software, which is composed of Software Intended Operation Trace and Software Intended Function Trace.

Definition 4. Software Intended Operation Trace: represents the intended routes on which some significant positions are selected orderly as monitor points. It can be denoted by ordered vectors.

Definition 5. Software Intended Function Trace: describes the intended functions performed on monitor points. It is constituted by a series of functions with related information and also denoted by ordered vectors.

B. Software Intended Operation Trace

Definition 6. Check Point: It's a significant point which was set up as a monitor on the route of software execution. It contains two types: Ordinary Check Point and Branch Check Point. Ordinary Check Point records function with its related information, and Branch Check Point records transfer condition and other related information.

The ordered vector of Check Points can guarantee the dynamic trustiness of software operation trace from the aspect of intended routes. However, where to set up the Check Points on the route of software execution is a significant problem that should be solved. It is to consider that: 1. from the routes coverage of software execution, setting up as more Check Points as possible will improve the accuracy of the constraints of software's dynamic operation trace; 2. from the efficiency of software execution, setting up more Check Points means extracting and storing more related information, which will reduce the efficiency of software execution. Therefore, how to make balance between accuracy and efficiency should be analyzed. In addition, the granularity of setting up Check Points determines the degree of software's dynamic trustiness. According to the rules as follows, Check Points can be set up on the routes of software execution:

Rule 1: Set up Ordinary Check Points at significant system calls. In order to perform certain function, most software needs to interact with kernel through system calls. System call sequences can reflect software behavior to a certain degree [16]. Therefore, it can evaluate the trustiness of software's dynamic behavior.

Rule 2: Set up Branch Check Points at each conditional branch, and set up Ordinary Check Points at the body of each branch separately. Due to the non-determinism caused by branches, software is easy to be attacked at each conditional branch and executes the unexpected branch path which is difficult to be detected. Therefore, it is necessary to set up Branch Check Points for ensuring the trustiness of dynamic behavior.

Rule 3: Set up Ordinary Check Points in the end of basic function. The results of software execution can evaluate whether the independent Software Module performed intended operation.

C. Software Intended Function Trace

Definition 7. Scene: It's a vector of n-tuples which records the background and function during software execution. Ordinary Check Point contains certain function with function name, function arguments, CPU load, memory usage, result of software execution and so on; Branch Check Point concludes CPU load, memory usage, branch data, transfer condition and so on.

Definition 8. Time Interval: It is an interval that software consumes between adjacent Check Points in the process of execution, which can ensure the dynamic trustiness of software behavior between adjacent Check Points.

The vectors which records Scene and Time Interval can guarantee the dynamic trustiness of Software Operation Trace from the aspect of intended function. However, for different Check Points, the same attribute, due to the difference of values, contributes to ensuring the dynamic trustiness of Software Intended Function Trace differently.

Information entropy is the measure of uncertainty of a random variable [17]. For the purpose of improving accuracy of the constraints of dynamic trustiness, a strategy of determining the weights of characteristic attributes based on information entropy is proposed.

Suppose that there is a set of n-samples, denoted by $E=\{e_1, e_2, \ldots\ldots, e_n\}$, which is obtained at certain Ordinary Check Point during software execution. Each sample e_j is represented by the vector of characteristic attributes $e_j=<Function, Argument, CPU, Memory, Result, TimeInterval>$. So, the matrix $E=\{ e(i,j) \}$, $1\le i\le n$, $1\le j\le 6$ is the source of determining the weights of characteristic attributes based on information entropy. The strategy mainly contains 5 steps as follows:

1. The probability p_{ij} when the value of attribute j is $e(i,j)$:

$$p_{ij} = \frac{\left|e(i,j)\right|}{\sum_{i=1}^{n}\left|e(i,j)\right|} \quad i = 1,2,\cdots,n \tag{1}$$

Where $|e(i,j)|$ is the frequency when the value of attribute j is $e(i,j)$, so that $\sum_{i=1}^{n}|e(i,j)|=n$

2. The information entropy of attribute j is:

$$e_j=-k\sum_{i=1}^{n}p_{ij}\ln p_{ij} \quad k=\begin{cases}1 & n_a=1\\ \frac{1}{\ln n_a} & n_a\geq 2\end{cases} \tag{2}$$

Where n_a is the count of different values of attribute j, $\ln n_a$ is the max value of information entropy, so $0\leq e_j\leq 1$

3. Let

$$g_j=1-e_j \tag{3}$$

4. The sum of information entropy of $E=\{ e(i,j) \}$ is:

$$E'=\sum_{j=1}^{6}e_j \tag{4}$$

5. The weight of attribute j after normalized is:

$$\omega_j=\frac{g_j}{6-E'} \quad j=1,2,\cdots,6 \quad 0\leq\omega_j\leq 1,\sum_{j=1}^{6}\omega_j=1 \tag{5}$$

However, the strategy has its limitation which didn't consider the relations among characteristic attributes, and an intensive study will be made in the future.

D. Extraction of Software Intended Behavior

Extracting the intended behavior of software is crucial to the generation of Trust View. The methods of extraction generally contains: static extraction and dynamic extraction. Static extraction doesn't need to execute software. It is able to obtain the control flow of software by analyzing the source code, but can't extract background information [18]. Dynamic extraction can obtain the execution model through monitoring the dynamic behavior. The model is incomplete because its generation relays on the input and operation [19], but it contains the context, execution time and related information of software execution.

The TSCMTE makes full use of static extraction and dynamic extraction. Firstly, static extraction was done to select some significant points as Check Points on the intended routes of software, and then the intended operation trace of software was gotten. Secondly, when the software is in the process of execution, dynamic extraction was done to extract the Scene, Time Interval with other related information by weaving the sensors on the position of Check Points. Specifically, sensor is essentially a program for extracting function with related information, which is triggered automatically. Then the intended function trace during software execution was obtained.

Both the intended operation trace and intended function trace mutually complete each other, and then constitute the software intended behavior trace accurately. The generation process of software intended behavior trace is shown in Fig. 4.

E. Representation of Software Intended Behavior

The existing representations of dynamic behavior of software, such as Petri nets [20], automata theory [21], didn't take the relationships between the behavior and trustiness of software into account. Qu Yanwen [7] proposed the behavior tree, which is a representation of behavior trace of software. Actually, the behavior tree is an effective representation of Software Intended Behavior.

The TSCMTE adopts Trust View to represent Software Intended Behavior which can be described as *TrustView(V, E, T, v_0, V_e)*:

V is the set of check points, can be expressed as *V(Id, Type, Scene)*, where *Id* is the unique identity; *Type* describes the type of check points: *0* represents ordinary check points, and *1* represents branch check points; Scene characterizes the function and related information of current check point. If *Type=0*, it can be denoted by *Scene<Function, Argument, CPU, Memory, Result>*; if *Type=1*, it can be denoted by *Scene<CPU, Memory, BranchData, TransferCondition>*.

E is a set of edges connecting check points associated with transitions. It is the subset of the 2-dimensional space $V\times V$. Its element e_i is a directed edge and be described as $e_i=<v_i, v_j>$.

T is the set of time intervals. For any edge $e_i=<v_i, v_j>\in E$, the weight of e_i represents the transferred interval between v_i and v_j.

v_0 is the initial check point, $v_0\in V$.

V_e is the set of the final check points, $v_e\in V$.

F. Trustiness Evaluation

Based on the research on software behavior, a strategy of trustiness evaluation is proposed, the process is given in Fig. 5.

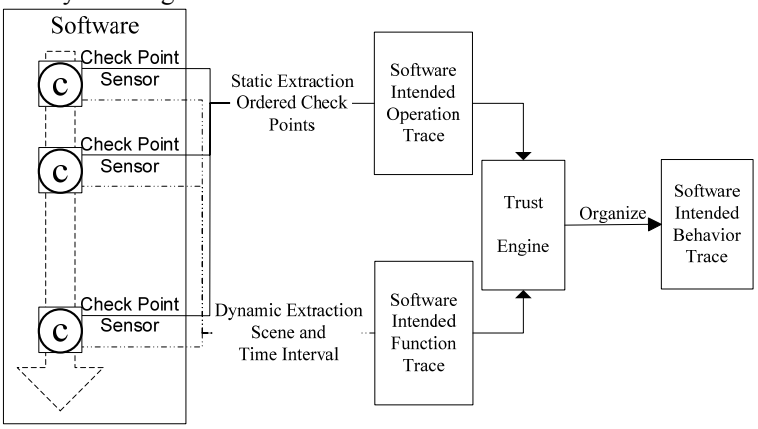

Figure 4. Generation process.

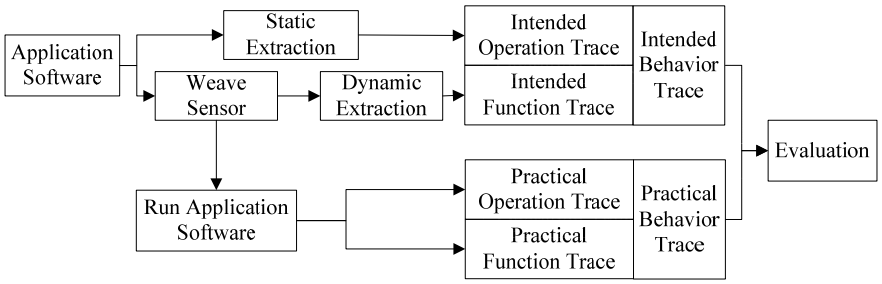

Figure 5. Trustiness Evaluation of TSCMTE.

The evaluation of software intended operation trace is to verify whether the practical identification of check point consistent with the intended. However, for the evaluation of software intended function trace, we adopt the above-mentioned strategy of determining the weights of characteristic attributes based on information entropy. After get the weights of all attributes, we sum up all sample pattern with different weighting factor to perform evaluation. The values of every attribute are described by abstract value range [22]. If the value of j belongs to the intended range, then $c_j=1$, else $c_j=0$. The value of evaluation C is:

$$C = \sum_{j=1}^{6} \omega_j c_j \tag{6}$$

A trusted threshold T is defined. If $C<T$, it indicates the software has been attacked, else the software is trusted. The value of T depends on special condition.

V. Simulation Experiments and Results

This section discusses the simulation experiments undertaken to evaluate the trustiness and performance of the TSCMTE. Trustiness is generally evaluated in two aspects: the effectiveness and accuracy. All evaluations obtained were run on a Core(TM)2 E8400 trusted computer based on TPM with 2GB of ram for Linux 2.6.12.

The TSCMTE proposed has carried out trustiness constraints of software's static integrity and dynamic behavior completely. The integrity measurement provided by TPM can guarantee the static integrity of software, but the dynamic constraints can't ensure the trustiness of software behavior accurately. The goal of our model is to keep high accuracy of trustiness evaluation.

Accuracy is generally evaluated in two aspects: True Positive Rate (TPR) and False Positive Rate (FPR). True Positive represents an effective detection, this is, accepting an event when it is actually trusted; False Positive represents a wrong detection, that is, rejecting an event when it is actually trusted. The TPR should be as high as possible, and the FPR should be as low as possible.

This paper carried out the experiments for the purpose of accuracy evaluation without weights (equal weights) and with weights respectively, the results reflected in Fig. 6 and Fig. 7 that show the TPR and FPR based on software as below: wu-ftpd, linuxconf, gzip and ps after attacked by existing exploits.

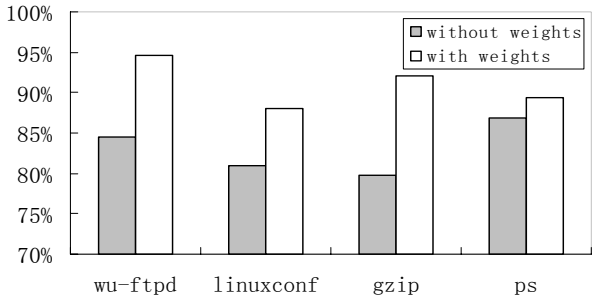

Figure 6. True Positive Rate of TSCMTE.

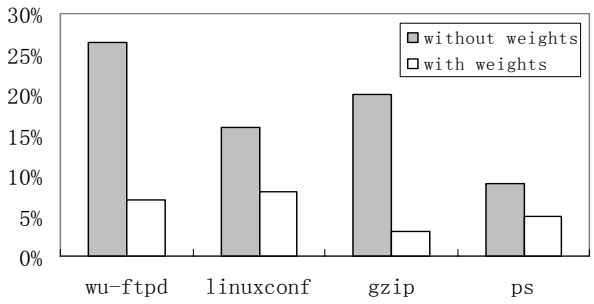

Figure 7. False Positive Rate of TSCMTE.

From the above experiments it can be found out that the TSCMTE without weights (equal weights) performs high True Positive Rate, but also has high False Positive Rate. This is due to the fact that the features of dynamic behavior are deficient, that is, the characteristic attributes extracted are so limited that can't distinguish the trusted behavior from anomalous behavior.

Take an Ordinary Check Point in gzip for example, this paper adopted the strategy introduced in section 4.3.2 to carry out experiments, the weights of characteristic attributes and detection results are shown in Table Ⅰ. It can be observed that, the TSCMTE with weights not only improves the True Positive Rate, but also greatly reduce the False Positive Rate.

TABLE I.

WEIGHTS OF CHARACTERISTIC ATTRIBUTES AND DETECTION RESULTS

Characteristic Attributes		Function	Argument	CPU	Memory	Result	TimeInterval	Detection Results
Without weights	Weights	0.166	0.166	0.166	0.166	0.166	0.166	——
	Attack	×	×	√	√	√	√	False Negative(0.33)
	Normal	√	×	√	×	√	×	False Positive(0.498)
With weights	Weights	0.37	0.23	0.13	0.05	0.2	0.02	——
	Attack	×	×	√	√	√	√	Attack(0.6)
	Normal	√	×	√	×	√	×	Normal(0.3)

Trusted threshold T is 0.4：√ represents consistence with the intended; × represents deviation from the intended.

The performance of the TSCMTE mainly comes from verifying whether the dynamic behavior trace extracted during software execution is in accordance with the intended. From the investigation, it is found that regular software has a considerably lower system call density. Take gzip for example, when the model was enabled, the CPU load increased about 15% and the memory usage grew by about 10%. Therefore, the TSCMTE has an acceptable performance. Meanwhile, it also greatly improves the trustiness of software.

In addition, this model has been adopted in the information management system of Department of Science and Technology of Hebei Province, China, the effective trustiness and acceptable performance have been proved completely.

VI. CONCLUSION

In this paper, a trusted software constitution model based on Trust Engine (TSCMTE) is proposed. Software Intended Behavior Trace was introduced to describe the intended behavior of software, which consists of Intended Operation Trace and Intended Function Trace. The former is the intended routes which can be denoted by vectors of ordered Check Points; the latter describes the intended functions, which is constituted by a series of functions with related information. Time Interval can ensure the trustiness of software behavior between adjacent Check Points. For the purpose of ensuring software's trustiness, the TSCMTE carries out constraints of software's static integrity and dynamic behavior completely. Furthermore, in order to improve the accuracy of constraints, a strategy of determining the weights of characteristic attributes based on information entropy is proposed. The simulation experiments and practical application show that the trustiness of software developed by the TSCMTE is improved greatly without performance degradation. However, the TSCMTE has its own limitations, an intensive research will be made on the trustiness evaluation and the temporal and spatial correlations among characteristic attributes for determining weights in the future.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China (Grant No.60873203), the Foundation of Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education (Grant No.AISTC2009_03), Hebei National Funds for Distinguished Young Scientists (Grant No.F2010000317), the National Science Foundation of Hebei Province (Grant No.F2010000319).

REFERENCES

[1] Chen Huowang, Wang Ji, Dong Wei, "High Confidence Software Engineering Technologies," *Acta Electronica Sinica*, vol. 31, no. 12A, pp. 1933–1937, 2003.

[2] Li Renjie, Zhang Zhuxi, Jiang Haiyan, Wang Huaimin, "Research and implementation of trusted software constitution based on monitoring," *Application Research of Computers*, vol. 26, no. 12, pp. 4585–4588, Dec. 2009.

[3] Mei Hong, Liu Xizhe, "Software techniques evolved by the Internet: Current situation and future trend," *Chinese Sci Bull*, vol. 55, no. 13, pp. 1214–1220, 2010.

[4] Edmund M Clarke, Jeannette M Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys*, vol. 28, pp. 626–643, 1996.

[5] S. J. Prowell, C. J. Trammell, R. C. Linger, J. H. Poore, *Cleanroom Software Engineering: Technology and Process*, Boston: Addison-Wesley Professional, 1999.

[6] VO Safonov, *Using aspect-oriented programming for trustworthy software development*, Wiley Interscience, Jun. 2008.

[7] Qu Yanwen, *Software Behavior*, Beijing: Electronic Industry Press, Oct. 2004.

[8] Liu Huimin, Zhang Wenhui, "Model Checking: Theories, Techniques and Applications," *Chinese Journal of Electronics*, vol. 30, no. 12A, pp. 1907–1912, Dec. 2002.

[9] Wang Huaimin, Liu Xudong, Xie Bing, *Software Trustworthiness Classification Specification*, TRUSTIE-STC V2.0, May 2009.

[10] Liu Jing, He Jifeng, Miao Huaikou, "A strategy for model construction and integration in MDA," *Journal of Software*, vol. 17, no. 6, pp. 1141–1422, Jun. 2006.

[11] Li Xiaoyong, Shen Changxiang, "Research to a dynamic application transitive trust model," *J.Huazhong Univ. of Sci. & Tech (Nature Science Edition)*, vol. 33, pp. 310–312, Dec. 2005.

[12] Liu Dayou, Yang Kun, Chen Jianzhong, "Agents: Present Status and Trends," *Journal of Software*, vol. 11, no. 3, pp. 315–321, Nov. 2000.
[13] David Challener, Kent Yoder, Ryan Catherman, *A Practical Guide to Trusted Computing*, Upper Saddle River, NJ: IBM Press, 2009.
[14] Shen Changxiang, Zhang Huanguo, Wang Huaimin, "Research on trusted computing and its development," *SCIENCE CHINA Information Sciences*, vol. 53, pp. 405–433, Mar. 2010.
[15] Trusted Computing Group, *TCG Specification Architecture Overview*, Https://www.trustedcomputinggroup.org/groups /TCG_1_ 0_Architecture_Overview.pdf.
[16] Yao Lihong, Zi Xiaochao, Huang Hao, Mao Bing, Xie Li, "Research of System Call Based Intrusion Detection," *Acta Electronica Sinica*, vol. 31, no. 8, pp. 1134–1137, Aug. 2003.
[17] Cover T M. and Thomas J A, *Elements of Information Theory*, New York: Wiley, 1991.
[18] M. Christodorescu, S. Jha, "Static analysis of executables to detect malicious patterns," *In Proceedings of the 12th USENIX Security Symposium (Security'03)*, pp. 169–186, Aug. 2003.
[19] L. Wendehals, "Improving Design Pattern Instance Recognition by Dynamic Analysis," *In Proc. of the ICSE 2003 Workshop on Dynamic Analysis (WODA)*, pp. 29–32, May 2003.
[20] Luo Junzhou, Shen Jun, Gu Guanqun, "From Petri Nets to Formal Description Techniques and Protocol Engineering," *Journal of Software*, vol. 11, no. 5, pp. 606–615, Nov. 2000.
[21] S Helke, F Kammiller, "Representing hierarchical automata in interactive theorem provers," *Proc of the 14th International Conference on Theorem Proving in Higher Order Logics*, London: Springer-Verlag, pp. 233–248, 2001.
[22] Xiao Qing, Gong Yunzhan, Yang Zhaohong, Jin Dahai, Wang Yawen, "Path Sensitive Static Defect Detecting Method," *Journal of Software*, vol. 21, no. 2, pp. 209–217, Feb. 2010.

Junfeng Tian, born in Baoding, China, 1965, received Ph.D degree of computer science from University of Science and Technology of China in 2004.

He is a professor of Computer Science at Hebei University. In the past few years, he has published many technical papers in refereed journals and conference proceedings. His research interests include network security, trust computing and distributed computing.

Ye Zhu, born in Shijiazhuang, China, 1985, received Master degree of computer science from Hebei University in 2011.

He works in Shijiazhuang Posts and Telecommunications Technical College. His research interests include network security, trust computing.

Jianlei Feng, born in Cangzhou, China, 1984, received Master degree of computer science from Hebei University in 2011.

He studied in the College of Mathematics and Computer Science of HeBei University His research interests include network security, trust computing.