Journal of Systems Engineering and Electronics
Vol. 26, No. 2, April 2015, pp.367–380

Analysis of system trustworthiness based on information flow noninterference theory

Xiangying Kong[1,2,*], Yanhui Chen[2], and Yi Zhuang[1]

1. College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China;
2. Jiangsu Automation Research Institute, Lianyungang 222061, China

Abstract: The trustworthiness analysis and evaluation are the bases of the trust chain transfer. In this paper the formal method of trustworthiness analysis of a system based on the noninterference (NI) theory of the information flow is studied. Firstly, existing methods cannot analyze the impact of the system states on the trustworthiness of software during the process of trust chain transfer. To solve this problem, the impact of the system state on trustworthiness of software is investigated, the run-time mutual interference behavior of software entities is described and an interference model of the access control automaton of a system is established. Secondly, based on the intransitive noninterference (INI) theory, a formal analytic method of trustworthiness for trust chain transfer is proposed, providing a theoretical basis for the analysis of dynamic trustworthiness of software during the trust chain transfer process. Thirdly, a prototype system with dynamic trustworthiness on a platform with dual core architecture is constructed and a verification algorithm of the system trustworthiness is provided. Finally, the monitor hypothesis is extended to the dynamic monitor hypothesis, a theorem of static judgment rule of system trustworthiness is provided, which is useful to prove dynamic trustworthiness of a system at the beginning of system construction. Compared with previous work in this field, this research proposes not only a formal analytic method for the determination of system trustworthiness, but also a modeling method and an analysis algorithm that are feasible for practical implementation.

Keywords: trusted computing, trust chain, intransitive noninterference (INI), dynamic trustworthiness, access control.

DOI: 10.1109/JSEE.2015.00043

1. Introduction

Trusted computing proposed by the trusted computing group (TCG) is regarded as an effective method for solving the security problem of information systems, and the core idea is to use a hardware module called trusted platform module (TPM) which is independent of CPU control as the source of trust. A chain of trust is established on the basis of the trusted TPM, which means that all the software must be verified to be trustable before running [1]. Software is the soul of a modern computer system because the functionality of a modern computer system is implemented in the form of software programs. Trustworthiness of a computer system is based on trustworthiness of all its running software.

The concept of trustworthiness of a software program originates from social science [2]. Trust in society is defined as "a social behavior that prefers those who behave according to societal rules" [3]. The trust relationship between two parties may vary with time and situations, so trust relationships between groups or people are dynamic [4,5]. In [4], trust is characterized as a social relationship that is dynamically developed and spirally reinforced. In [6], trust is viewed as a quantity that gradually changes rather than a binary choice between trust and distrust. In fact, the trust relationship between people varies as time and situation change. For example, the trust relationship between group members varies according to the nature of their tasks and the progress of these tasks. Similar to that in social science, the trust relationship between software entities also possesses this dynamic nature. Trustworthiness of a software entity depends on both internal and external causes. The internal causes are dynamic changes of the software entity itself, and the external causes are the dynamic environment in which the software entity is executed, i.e. changes of the state of a computer system. However, the impact of the system state on trustworthiness of software is largely overlooked in most existing researches.

Currently, there are many research achievements in the measurement model of software trustworthiness. Patel and Beth use the probabilistic approach to model and measure trustworthiness of software entities running in an open network environment, but their approach emphasizes the "trust and reputation" of a software entity [7,8]. In order

Manuscript received April 25, 2014.
*Corresponding author.
This work was supported by the Natural Science Foundation of Jiangsu Province (BK2012237).

to reflect the fuzzy nature of the trust relationship, Tang and Mo employ the fuzzy set theory to model and assess the management of trustworthiness of software entities in an open network environment [9–11]. There are some other approaches to model software trustworthiness. In [12,13], trust models based on the process algebra and evidence theory were proposed, and in [14], a trust model based on software behaviorism was presented. All these models represent the authors' own understanding and definition of the concept of trust, but they do not reflect the nature of the run-time interaction between software entities. The TCG proposed a TPM-based binary attestation (TBA) [1]. IBM proposed integrity measurement architecture (IMA) based on the TBA, which is capable of doing an integrity measurement before loading modules into the system [15], but the integrity of modules is only one of many trustworthiness-related properties of a software entity. In [16], the limitations of trustworthiness measurement before TCG loading was analyzed and a conclusion was made that load-time trustworthiness does not imply run-time trustworthiness, the integrity and trustworthiness are not equivalent and static trustworthiness does not imply run-time trustworthiness.

To solve the problem of software run-time trustworthiness, some researchers take approaches based on the noninterference (NI) theory of information flow. In [17], the NI theory of information flows into the field of trusted computing. Their approach instantiates the abstract term entities in the TCG's definition of trustworthiness in processes, maps processes to the security domain of the NI theory, establishes a trust chain transfer model based on the NI theory, and provides a formal description and validation of this model, but the forms of interference are not analyzed in their papers, and the trust chain transfer function needs further studies. Base on their work, in [18,19], an abstraction was made that a running system is composed of processes/modules, actions and state outputs. They employed the intransitive noninterference (INI) theory and reference monitor assumptions (RMA) to present a formal definition, description and analysis of the conditions of a trusted running process/module. They also provided the judgment method of trust chain transfer of the trusted computing platform and proved it. A prototype of trusted computing using the virtual machine separation technique was also constructed.

There are two problems left [18,19]: (i) The prototype system is implemented using the virtual machine which uses a very coarse trustable unit. Trustworthiness within the virtual machine itself and how the trust chain transfers are not discussed. (ii) Any details about the implementation of the trustable communication channel are not given. The software dynamic trustworthiness based on process algebra is modelled, and a method of analyzing software dynamic trustworthiness is proposed [20]. The method is applied to analyze the interference relationship between information flow within a single process and information flow among multiple processes based on the INI theory. These results provide a novel idea for using the INI theory to analyze the software trustworthiness during load-time and run-time. However, researches mentioned above do not take into account the impact of system state on software dynamic trustworthiness, and system trustworthiness is not studied either.

NI of information flow was proposed first by Goguen and Meseguer [21], which is a method for analyzing system security from the perspective of information flow. After that, Rushby improved it by proposing the INI model of information flow [22]. The core idea of the INI model is that for a particular access operation, if the system state after a sequence of access actions are executed is identical to the state after all actions that do not interfere with a particular domain are removed from the sequence of actions with the *ipurge* function, then the access operation is considered to be secure, otherwise, there must be a potential access operation that could interfere with the execution of the access operation, which results in an unsafe operation. The INI model divides the system entity into several different security domains according to the predefined security levels. If actions of a security domain u have no effect on the outputs of another security domain v, then it can be considered that domain u does not interfere with domain v. The interference rules of an INI model are considered to be static, so the INI theory cannot describe changes of the interference relationship among software entities caused by changes of the system states and the INI theory cannot help to analyze the impact of the system state on the transfer of the trust chain.

To reflect the impact of the system state on interference rules, Leslie proposed a dynamic intransitive noninterference (DINI) model [23], which uses the automaton to model a system. An action will cause the state transition of the system, and generate some outputs. To reflect the dynamic characters of security rules of the system which changes with time, the DINI model associates rules with the state. That is to say, the system state is a dependent variable of the interference relationship. In different system states, the interference relationship between domain u and domain v is also different, so DINI is able to describe the interference relationship among security domains in different system states.

In this paper, the impact of the system state on trustworthiness of software is analyzed, the access control tech-

nique is employed to model the run-time states of software, a formal analysis method to evaluate trustworthiness of the software entity in the process of trust chain transfer and the definition of trustworthiness of trust chain transfer are proposed and a judgment rule of the system dynamic trustworthiness is provided. The main innovations are: (i) we discuss the impact of system state on software trustworthiness and establish an automaton model of the dynamic systems; (ii) we propose a formal analysis method for analyzing trustworthiness of software entities and systems, and give a judgment theorem; (iii) we extend the RMA to the dynamic reference monitor assumptions (DRMA) and give a judgment theorem of trustworthiness of a system.

The rest of this paper is organized as follows. In Section 2, the access control technique is used to model the run-time state of software systems. Section 3 first introduces the DINI model based on which a formal analytic model of run-time software trustworthiness is provided and a DINI-based system dynamic judgment theorem is proved. Section 4 uses examples to illustrate the process of analyzing the software dynamic trustworthiness with the DINI model, and presents a prototype system with dynamic trustworthiness. Section 5 is a conclusion of this paper with a summary of all work in the previous sections and gives a future research plan.

2. Modeling of the dynamic behavior of a system

In a modern computer system, a software program is stored in a data storage device in the form of a binary executable image before it is loaded into the system. To run the software program, the operating system (OS) will load the binary image into memory, allocate all necessary resources, create a corresponding process and start it, so a process is the form of a run-time software program. At the program loading stage, trust chain transfer is carried out with the software module as the basic unit of a transfer, so at this stage the software entity is composed of all the independently loadable modules. At the run-time stage when all the processes are scheduled to run by the OS scheduler, transfer of the system trust chain is carried out with the process as the basic unit of a transfer, so at this stage the process is the object of this study of dynamic trustworthiness of systems.

2.1 Impact of the system state on trustworthiness of processes

In [17–19], the assumption is made that the basic entity of trusted computing is the process. In [19], a trusted computing environment using the virtualization technology is constructed. The entity of the trust chain transfer is called a component, which is also a process from the perspective of the hypervisor. All these researches are based on the INI model, which assume that interferences among processes are invariant, and do not take into account the impacts of the system state on software trustworthiness.

In a practical system, trustworthiness of a software entity is closely related to the system state in which it is running. Consider the following two cases.

Case 1 Assume there are two users Hu and Li in a system and they belong to two different security levels (security domains). Normally information is only allowed to flow from Li to Hu rather than the reverse direction of information flow, but under some particular conditions, the system will temporarily upgrade the security level of Li, and allow information flow from Hu to Li. For example, Li obtains a temporary secret key, and is allowed to have access to a file that belongs to Hu.

Case 2 The INI system constructed with the virtual machine in [19] is shown in Fig. 1. The arrows represent the direction of information flow, i.e. the interference relationship. Due to the characteristics of software, updates are unavoidable. Assume virtual machine A with a high security level needs to be updated, and then the system has to do this via the update module of the HyperVisor. A new domain A' is introduced to minimize impacts of the updating process on the running system A. In the HyperVisor, another new domain H should be introduced, which creates the domain A', and copies information of A to A'. The interference rule is $A \sim\!> H$, $H \sim\!> A'$ ($A \sim\!> H$ means domain A is allowed to interfere with domain H, i.e. information is allowed to flow from domain A to domain H). When domain H is copying information to configure domain A', domain A can still be communicating with virtual machine C via a trusted channel B, as shown in Fig 2. Once A' is properly configured, A' will replace A to communicate with C via the trusted channel B and the system rule now is changed to $A \not\sim\!> H$ ($A \not\sim\!> H$ means domain A cannot interfere with domain H, i.e. information flow from domain A to domain H is not allowed), but domain H still keeps high level security information it obtains while it is configuring domain A', as shown in Fig. 3. Since the system needs domain H to update domain C, a new domain C' has to be created, which introduces rules $C \sim\!> H$ and $H \sim\!> C'$, and then an insecure channel is formed which allows information with a high security level to flow to domains with a low security level, as shown in Fig. 4. As a result, as the system state changes, trustworthiness of modules which are used to update the virtual ma-

chine in domain H also changes.

From the cases above, it can be seen that the interference relationship among processes changes with the system state, and trustworthiness of software at a particular time and in a particular system environment does not imply its trustworthiness when it is scheduled to run for the second time in a different system environment.

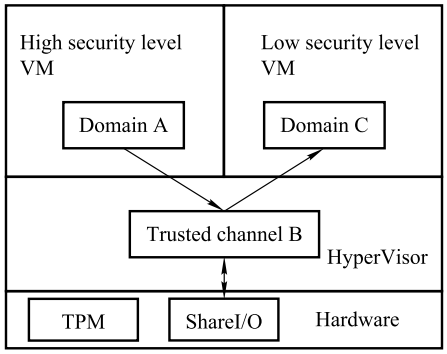


Fig. 1 Trusted computing system constructed in [19] based on virtual machine

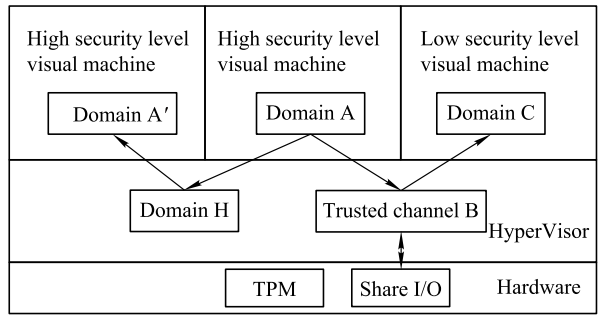


Fig. 2 An illustration of the process of updating domain A

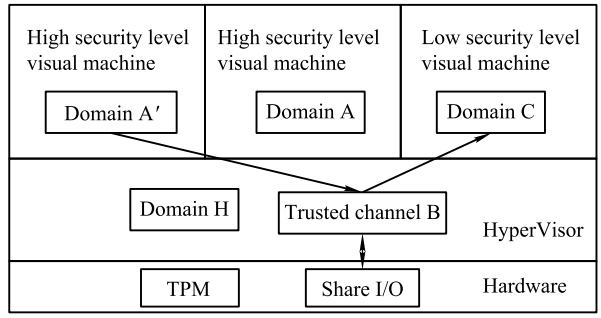


Fig. 3 Process after the upgrade

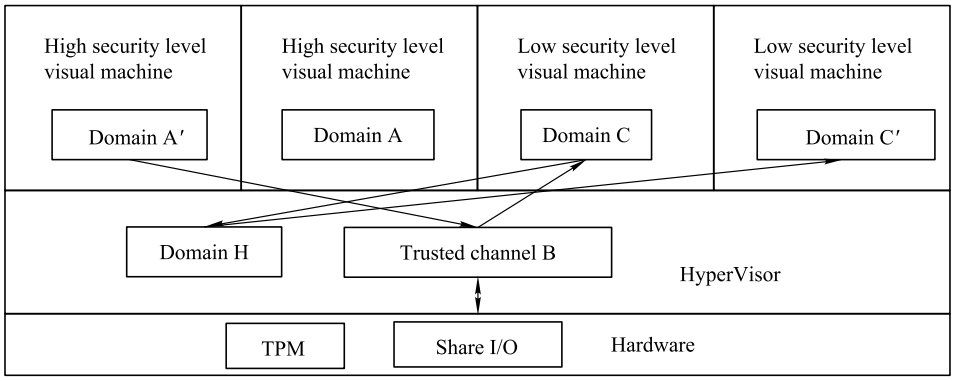


Fig. 4 An illustration of the process of updating domain C

2.2 Definition of trustworthiness of a software entity

Trustworthiness of a software entity R_A includes not only module-integrity trustworthiness, but also run-time dynamic trustworthiness. To make it simple, in this paper it is assumed that each software entity has only one corresponding process in the system. The following is the formal definition of trustworthiness of a software entity.

Definition 1 Trustworthiness of software entity R_A can be formally described as

$$Trusted(R_A) \Leftrightarrow (Load\_Trusted(M_A) \wedge Run\_Trusted(P_A)) \quad (1)$$

where M_A denotes the binary module before R_A is loaded into the system memory. $Load\_Trusted(M_A)$ means M_A is trusted at the module loading stage, P_A denotes the run-time process of R_A, and $Run\_Trusted(P_A)$ means P_A is trusted during the run-time.

The TCG specification gives a measurement method for integrity trustworthiness [1]. In this paper, trustworthiness of run-time processes is discussed.

In most computer systems, an independently loadable software module M_A is an object file such as a binary executable or a dynamic loadable library stored on a storage device like a disk. Such a software module can be loaded into the system memory through a system loading function (data exchange between the primary memory and the sec-

ondary memory could happen when processes are rescheduled and memory pages are swapped in and out), and will possess codes and data that are used to perform a particular functionality, which is the same for OS kernels. These software modules possess the independent text (code) segment and data segment, and exist in the form of one process or a group of processes during the run-time. Each process possesses a group of registers, stacks/heaps and other necessary software and hardware recourses, such as CPU, I/O ports, file/device descriptors (handles).

According to the definition of a trusted software entity by the TCG, in a trusted system, an entity is considered to be trusted if it behaves as the system expects. In other words, it does not violate the predefined conventions of the system and has no adverse effect on other entities in the same system. In computer systems, during the trust chain transfer process, a software entity is loaded into the system memory, and is allowed to use system resources such as CPU and I/O devices, which implies that the direct control of registers, memory and I/O is handed over to the software entity. This type of state changes in a computer system could become the basis for a vicious process to attack the system and cause critical information leakage. To be specific, the main ways a malicious process can take to adversely affect other entities are: (i) violating CPU usage conventions, and occupying the CPU completely; (ii) using the system memory illegally, obtaining information unauthorized for the malicious software entity and viciously altering the contents in the memory of storage devices; (iii) illegally accessing files and devices. CPU usage is determined by the OS kernel, so this kind of attack can be avoided with the improvement of the security of the OS kernel. Attacks through (ii) and (iii) essentially take advantage of the information flow in the system. When a process has the CPU time, it will execute a series of actions to obtain or revise the permissions, operations or information of other software entities. These actions and outputs will violate the predefined rules of the system, and have an adverse effect on other entities. From this point of view, software attacks and information leakage can be viewed as an unexpected information flow in the system, which can be analyzed with the NI theory of information flow.

Software and hardware resources such as system memories, files and devices can be viewed abstractly as independently accessible (observable and changeable) objects. Each object is given a name and an associated value. The name of each object is fixed, but the corresponding value changes with the system state. The rule of access to an object of a process in the system is determined by a predefined access control function. The access control function determines whether a given process has the right to observe (read) or change (write) a given object's value in a particular state. A process in the system is composed of a sequence of actions. Each action accesses the set of names according to the access control rules, and generates some outputs. Operation of the system is driven by actions, and execution of each action will drive the system into a new state. The system state is determined by all the name-value pairs in the system. The following is the definition of the dynamic behavior model of computer systems.

Definition 2 A computer system M is an octuple $\{N, V, A, S, s_0, O, D, F\}$.

N is the set of names, representing the set of all the alterable resources in the system, such as memory space, variables, files, I/O ports etc. Italic letter m is used to represent names in the set.

V is the set of values, which is all the possible values of all the names. Greek letter ν represents the values.

A is the set of actions. Each action can be viewed as "input", "command" or "instruction" that the automaton executes, and italic letter a and b are used to denote actions. Greek letters α and β denote a sequence of actions.

S is the set of states, which is denoted using italic letters s or t.

s_0 is the initial state, $s_0 \in S$.

O is the set of outputs, which is defined as all the values corresponding to the names in N in states S.

D is the set of processes. Each process has a corresponding security domain, and it is a one-to-one correspondence between a domain and a process. The rest of this paper makes no distinction between the security domain and its corresponding process. u and v are used to denote the security domain (process). Each process sends out operational instructions to the system to interact with other parts of the system, and the corresponding results can be observed.

F is the set of functions, which includes the following six functions:

$contents$: $S \times N \rightarrow V$, $contents(s, m)$ is the value corresponding to the name m in the state s.

$observer$: $D \times S \rightarrow P(N)$, where $P(N)$ is the power set of the set of names, $observer(u, s)$ is the set of names whose values can be observed by process u in the state s.

$alter$: $D \times S \rightarrow P(N)$, $alter(u, s)$ is the set of names whose values can be altered by process u in the state s.

dom: $A \rightarrow D$, a mapping from an action to a process. $dom(a)$ denotes the associated process of action a.

$step$: $S \times A \rightarrow S$, the single-step state transfer function $step(s, a) = t$ means when an action a takes place in the state s, the system will transfer from state s to state t.

$output$: $S \times A \to O$, the output function $output(s,a)$ is the returned result of an action a in the state s.

The first three functions determine the access matrix, which can be used to dynamically describe the access control rules of the system. Notice that the state is a dependent variable of the access matrix.

The last three functions are used in the DINI model.

Definition 3 In system M and state s, running a process u is composed of executing a sequence of actions $a_0^u, a_1^u, \ldots, a_n^u$ that belong to this process, where $0 \leqslant i \leqslant n$, $a_i^u \in A$, $dom(a_i^u) = u$. The system goes through a sequence of state transfers $s_0^u, s_1^u, \ldots, s_n^u$, where $s_i^u \in S$, $0 \leqslant i \leqslant n$, $s_0^u \in step(s, a_0^u)$, $s_j^u \in step(s_{j-1}^u, a_j^u)$, $1 \leqslant j \leqslant n$. $s_0^u, s_1^u, \ldots, s_n^u$ is called the state trace of process u produced when u begins to run in state s, which is denoted by τ_s^u. $a_0^u, a_1^u, \ldots, a_n^u$ is called the action trace, which is denoted by α_s^u.

Since programs might jump to several different branches according to different conditions, so in different states, the external input might be different, and the action sequence executed by the process is also different, resulting in different traces.

To make it simple, it is assumed that the length of the action trace α_s^u is n, I is the set of non-negative integers, $n \in I$, $i \in I$ and $0 \leqslant i < n$.

The functions $head$, $tail$ and one are defined as

$$head\text{: } A^* \times I \to A^*, head(\alpha_s^u, i) = a_0^u, a_1^u, \ldots, a_{i-1}^u \quad (2)$$

$$tail\text{: } A^* \times I \to A^*, tail(\alpha_s^u, i) = a_i^u, a_{i+1}^u, \ldots, a_{n-1}^u \quad (3)$$

$$one\text{: } A^* \times I \to A^*, one(\alpha_s^u, i) = a_i^u. \quad (4)$$

A^* denotes the set of action sequences.

The functions $head$, $tail$ and one can be used respectively to obtain the prefix, suffix of the given length and the action of the given position in an action trace.

3. Trustworthiness modeling of a process based on DINI theory

In [19], the DINI model proposed by Leslie inherits many basic thoughts of Rushby, and it views a computer system as an automaton. An action will cause the transition of the system state, and generate some outputs. The main improvement of the DINI model includes the revised definitions of the sources function, the *ipurge* function and local interference.

Functions run, do and $test$ defined in [18] are applied to the computer system model defined above in this paper.

For a system M, the system running function $S \times A^* \to S$ is an extension of the function step.

$$run(s, \Lambda) = s \quad (5)$$

$$run(s, a \circ \alpha) = run(step(s,a), \alpha) \quad (6)$$

Λ denotes the empty sequence, $\circ$ denotes concatenation and $a \circ \alpha$ means a is the action before the action sequence α.

Functions do and $test$ are defined as follows:

$$do\text{: } A^* \to S,$$

$$do(\alpha) = run(s_0, \alpha) \quad (7)$$

$$test\text{: } A^* \times A \to O,$$

$$test(\alpha, a) = output(do(\alpha), a). \quad (8)$$

To reflect the dynamics of rules, the system state is introduced into the interference relationship.

$$\sim\!>\text{: } \overset{s}{\sim}\!>:: D \times D \times S \to Bool$$

It denotes the interference relationship among security domains in state s. $u \overset{s}{\sim}\!> v$ means domain u interferences with domain v in state s, i.e. in state s, information is allowed to flow from u to v, and v is able to observe the result of the execution of actions of the process u. $\overset{s}{\not\sim}\!>$ means there is no interference relationship.

Similarly, the system state into $sources$ and $ipurge$ functions is introduced as a parameter to reflect system dynamics.

In order to differentiate the function sources from $ipurge$ in the INI model and reflect the dynamic nature of the system model, these two functions are renamed as $dsources$ and $dipurge$ respectively in this paper.

Definition 4 Function $dsources : A^* \times D \times S \to P(D)$ where $P(D)$ is the power set of D.

$$dsources(\Lambda, u, s) = \{u\} \quad (9)$$

$$dsources(a \circ \alpha s, u, s) =$$

$$\begin{cases} dsources(\alpha s, u, t) \cup \{dom(a)\}, & \text{if } \exists v : v \in \\ \qquad dsource(\alpha s, u, t) \wedge dom(a) \overset{s}{\sim}\!> v \\ dsources(\alpha s, u, t), \quad \text{otherwise} \end{cases} \quad (10)$$

where $t = step(s, a)$.

For a non-empty action sequence $a \circ \alpha s$ executed in a given state s, $dsources$ checks whether a can directly interfere with a domain in $dsources(\alpha s, u, step(s, a))$ under the rules of state s.

Defintion 5 Function $dipurge$: $A^* \times D \times S \to A^*$ is defined as

$$dipurge(\Lambda, u, s) = \Lambda \quad (11)$$

$$dipurge(a \circ \alpha s, u, s) =$$

$$\begin{cases} a \circ dipurge(\alpha s, u, t), & \text{if } dom(a) \in \\ \qquad dsources(a \circ \alpha s, u, s) \\ dipurge(\alpha s, u, t), & \text{otherwise} \end{cases} \quad (12)$$

where $t = step(s, a)$.

In short, the result of performing the function $dipurge(\alpha, u, s)$ is to remove those actions in the sequence α which have no inference effect on u, and leave actions in the sequence that directly or indirectly interfere with domain u.

Definition 6 The dynamic security property of a system. A system with dynamic rules is secure if an arbitrary action sequence α and action a satisfy

$$test(\alpha, a) = test(dipurge(\alpha, dom(a), s_0), a). \quad (13)$$

The security property of the system dynamic rules can be explained as follows: the system starts from the initial state s_0, executes a series of actions $\alpha \in A^*$, and produces a series of state transitions. Each state transition generates some outputs, and the system finally reaches the state $s = do(\alpha)$. In this state, action a of domain u (where $u = dom(a)$) is executed, an output $test(\alpha, a)$ is generated. Actions a and its output u are used in a test to obtain some information about action sequence α. If u is able to differentiate action sequence α from $dipurge(\alpha, u, do(\alpha))$ by their different outputs through this test, then in state $do(\alpha)$, some domains v ($v \overset{s}{\nsim} u$) which are supposed to have no interference relationship with domain u contain actions that interfere with u, which contradicts with the predefined rule $v \overset{s}{\nsim} u$, so the conclusion is that this system is insecure.

In system M, processes are mapped to security domains and the trust chain transfer of the system (module loading and process scheduling) is equivalent to performing corresponding actions belonging to different security domains, so a given trust chain transfer process can be viewed as the execution procedure of the corresponding action sequence. If the interference relationship between the mapped domains satisfies the conditions of DINI, then the information flow between processes will obey the predefined security rules and violations of security rules will not happen. That means scheduling and execution of the processes are performed according to the predefined ways and rules with no illegal mutual interference, and then the corresponding loading procedure is trustable.

The following is the definition of trustworthiness of process execution in state s.

Definition 7 A system M performs a sequence of actions $\alpha \in A^*$, and reaches state $s = run(s_0, \alpha)$. $u \in D$ is a process in the system M. α_s^u τ_s^u are the action trace and the state trace of process u in state s, respectively.

If for $\forall a_i^u$, $a_i^u = one(a_s^u, i)$ all satisfies (14), then it can be said that process u is trusted in state s.

$$test(\alpha \circ head(\alpha_s^u, i), a_i^u) = test(dipurge(\alpha \circ head(\alpha_s^u, i)u, s_0), a_i^u) \quad (14)$$

Next is the definition of trusted transfer of the system trust chain.

Definition 8 The initial state of system M is s_0. s_0 is the root of the trust chain. Processes $u_0, u_1, \ldots, u_n \in D$ are sequentially scheduled and executed in the system. State s_i is the system state at the beginning of the execution of process u_i. If on the scheduling list of the OS scheduler, the execution of each process u_i is trusted in its corresponding state s_i, then the trust chain transfer is trusted.

4. Decision theorem of system dynamic trustworthiness based on DINI

The process dynamic trustworthiness model and its analysis method have been provided and the decision theorem of computer system dynamic trustworthiness, which can decide trustworthiness of the designed system, will be proposed in the following part.

In order to give the decision theorem of system dynamic trustworthiness, definitions of output consistency, weak single step consistency and dynamic partial interference of the computer system M are given as follows.

Definition 9 View partition and output consistency. The system M is view partitioned. If every process $u \in D$ has an equivalent relationship $\overset{u}{\sim}$ on S, and those equivalent relationships are consistent, then it shall meet (15).

$$s \overset{dom(a)t}{\sim} \Rightarrow output(s, a) = output(t, a) \quad (15)$$

Output consistency means if the two states s and t in the system are the same for process u, then in these two states, the execution of process u and the output result generated by the system are also the same.

Definition 10 Weak single step consistency. M is weak single step consistent if it meets (16).

$$s \overset{u}{\sim} t \wedge s \overset{v}{\sim} t \Rightarrow \forall a, dom(a) = v, step(s, a) \overset{u}{\sim} step(t, a) \quad (16)$$

Weak single step consistency means for processes u and v, if the two states s and t in the system are different, then in these two states, the execution of one process is transparent to the other, i.e. they cannot notice the execution of each other.

Definition 11 Dynamic partial interference. For one rule, if M is a view-partitioned output consistency system, then it can be said that M is dynamic partial interfered, if it meets (17).

$$v \overset{s}{\nsim} u \Rightarrow \forall a, dom(a) = v, s \overset{u}{\sim} step(s, a) \quad (17)$$

Namely, if the process v does not interfere with the process u in the state s, then the activity effect of v in the state

s cannot be noticed by u. From the perspective of state s, u cannot differentiate the result of state s before it executes action a from the result of state s after it executes action a in process v.

Definition 12 Regulation interference. A system meets regulation interference. If the two states are equivalent from the perspective of process u, then under the regulations of state s and t, the aggregates of the process that interfere the process u are also the same. Then, (18) is valid.

$$s \overset{u}{\sim} t \Rightarrow (v \overset{s}{\sim} > u \Leftrightarrow v \overset{t}{\sim} > u) \tag{18}$$

Theorem 1 Expansion theorem of dynamic regulation. M is a view-partitioned dynamic regulation system. If it meets all the following conditions:

(i) output consistency;

(ii) weak single step consistency;

(iii) dynamic partial interference;

(iv) regulation interference,

then M is safe relative to the dynamic regulation, which means (12) is valid.

In order to apply the expansion theorem of the dynamic regulation to the model system just established, it is necessary to introduce the DRMA, which is different from Rushby's RMA. It introduces dynamic characters that reflect the operation of the system.

First, the definition of equivalent relationship $\overset{u}{\approx}$ of state aggregation S is given.

Definition 13 Equivalent relationship $\overset{u}{\approx}$ of state aggregation S. $s \overset{u}{\approx} t$ and meets (19).

$$\begin{gathered}(observer(u,s) = observer(u,t)) \wedge \\ (alter(u,s) = alter(u,t)) \wedge \\ (\forall n \in observer(u,s) : contents(s,m) = \\ contents(t,n)) \wedge (\forall v(\forall n \in observer(u,s) \wedge \\ n \in alter(v,s)) \Rightarrow (n \in alter(v,t)))\end{gathered} \tag{19}$$

In state s, if acts of the system match the explanation of functions $observer(u,s)$ and $alter(u,s)$, the corresponding query control regulation can be executed and is regarded as safe. It needs to meet the following three assumptions.

Definition 14 Dynamic reference monitor assumptions.

Assumption 1 In order to make the output of act a only rely on the value observed by domain $dom(a)$, (20) needs to be valid.

$$s \overset{dom(a)}{\sim} t \Rightarrow output(s,a) = output(t,a) \tag{20}$$

Assumption 2 If act a transforms the system from one state to another, all new values that change the object must only rely on the value that can be observed and inquired by $dom(a)$, namely it should meet (21).

$$\begin{gathered}(s \overset{dom(a)}{\approx} t) \wedge \\ (contents(step(s,a),m) \neq contents(s,m) \vee \\ contents(step(t,a),m) \neq contents(t,m)) \wedge \\ (contents(step(s,a),m) \neq contents(s,m) \vee \\ contents(step(t,a),m) \neq contents(t,m)) \wedge \\ (contents(step(s,a),m) \neq contents(s,m) \vee \\ contents(step(t,a),m) \neq contents(t,m)) \\ \Rightarrow contents(step(s,a),m) = contents(step(t,a),m)\end{gathered} \tag{21}$$

Assumption 3 In state s, if act a changes the value of object m, then $dom(a)$ should be able to alter and access m.

$$\begin{gathered}contents(step(s,a),m) \neq \\ contents(s,m) \Rightarrow \\ n \in alter(dom(a),s)\end{gathered} \tag{22}$$

In the above DRMA, the difference between DRMA and RMA is that Assumption 1 and Assumption 3 introduce the state restriction. This improvement directly reflects the information flow interference relationship between the system state change and the software entity, which means the trustworthiness restriction.

The following is the dynamic trustworthiness judgment theorem based on the DINI system.

Theorem 2 System dynamic trustworthiness judgment theorem. If a system meets the requirement of DRMA and (23), then the system is trustworthy.

$$\begin{gathered}n \in alter(u,s) \wedge \\ n \in observer(v,s) \Rightarrow u \overset{s}{\sim} > v\end{gathered} \tag{23}$$

Proof the system should meet four requirements of the dynamic regulation expansion theorem (Theorem 1).

(i) Requirement 1: output consistency.

Make the view-partitioned relationship $\overset{u}{\sim}$ of the dynamic intransitive expansion theorem equivalent to the corresponding relationship $\overset{u}{\approx}$ of the DRMA. Assumption 1 of DRMA will directly get the output consistency.

(ii) Requirement 2: weak single step consistency.

In order to prove the weak single step consistency, it's necessary to prove

$$\begin{gathered}s \overset{u}{\sim} t \wedge s \overset{dom(a)}{\sim} t \Rightarrow \\ step(s,a) \overset{u}{\sim} step(t,a).\end{gathered}$$

The above formula can be rewritten as

$$s \overset{u}{\sim} t \wedge s \overset{dom(a)}{\sim} t \Rightarrow$$

$$contents(step(s,a),m) =$$

$$contents(step(t,a),m)$$

$$n \in observer(u,s).$$

It can be divided into the following three cases:

Case 1

$$contents(step(s,a),m) \neq contents(s,m)$$

Assumption 2 of DRMA can directly get the conclusion based on $s \overset{dom(a)}{\sim} t$.

Case 2

$$contents(step(t,a),m) \neq contents(t,m)$$

Case 3

$$contents(step(s,a),m) = contents(s,m) \wedge$$

$$contents(step(t,a),m) = contents(t,m)$$

Since $s \overset{u}{\sim} t$, $contents(s,m) = contents(t,m)$, then the conclusion is valid.

(iii) Requirement 3: dynamic partial interference.

Prove

$$dom(a) \overset{s}{\nsim} u \Rightarrow s \overset{u}{\nsim} step(s,a).$$

Through the proof by contradiction, the definition of $\overset{u}{\sim}$ can be extended, namely

$$(\exists n \in observer(u,s) : contents(s,m) \neq$$

$$contents(step(s,a),m)) \Rightarrow dom(a) \overset{s}{\nsim} u$$

If $contents(step(s,a),m) \neq contents(s,m)$, through Assumption 3 of DRMA, it can be seen that $m \in alter(dom(a),s)$ and then

$$m \in alter(dom(a),s) \wedge m \in observer(u,s).$$

$dom(a) \overset{s}{\nsim} u$ can be extrapolated through the conditions given by the theorem.

(iv) Requirement 4: regulation interference, namely to attest

$$s \overset{u}{\sim} t \Rightarrow ((v \overset{s}{\nsim} u) \Leftrightarrow (v \overset{t}{\nsim} u)).$$

In consideration of the orderliness of s and t, it is necessary to prove only

$$s \overset{u}{\sim} t \Rightarrow ((v \overset{s}{\nsim} u) \Rightarrow (v \overset{t}{\nsim} u)).$$

where $v \overset{s}{\nsim} u$, namely,

$$\exists n \in alter(v,s) \wedge m \in observer(u,s).$$

From the definition of $\overset{u}{\sim}$ in Assumption 1 of DRMA

$$(m \in alter(v,s) \wedge m \in observer(u,s)) \Rightarrow$$

$$(m \in alter(v,t) \wedge m \in observer(u,t)),$$

it is known that $v \overset{t}{\nsim} u$.

And then, the theorem is proved. □

5. Example analysis and realization of prototype system

5.1 Example analysis

For case 1 mentioned above, the system is modeled as follows:

State aggregation: $S = Bool \times Bool$

User aggregate: $U = \{Hu, Li\}$

Command aggregate: $C = \{flip, slip\}$

Act aggregate: $A = U \times C$

Output aggregate: $O = Bool \times Bool \cup Bool$

System state: $s \in \{(h,l),(\overline{h},\overline{l}),(\overline{h},l),(h,\overline{l})\}$

Safe domain: $D = \{H, L\}$

Function: $dom = \{((Hu,c),H),((Li,c),L)\}$

Security policies:

$$\sim>= \{(L,H,(h,l)),(L,H,(\overline{h},l)),$$
$$(L,H,(h,\overline{l})),(H,L,(\overline{h},\overline{l}))\},$$
$$\nsim>= \{(H,L,(h,l)),(H,L,(\overline{h},l)),$$
$$(H,L,(h,\overline{l})),(L,H,(\overline{h},\overline{l}))\}.$$

Namely, in state $(\overline{h},\overline{l})$, the system allows information to flow from Hu to Li.

$step$ and $output$ functions:

$$step((h,l),(Li,flip)) = (\overline{h},\overline{l})$$

$$step((h,l),(Li,slip)) = (h,l)$$

$$step((h,l),(Hu,flip)) = (\overline{h},l)$$

$$step((h,l),(Hu,slip)) = (h,l)$$

$$output((h,l),(Hu,c)) = [h,l]$$

$$output((\overline{h},l),(Hu,c)) = [\overline{h},l]$$

$$output((h,\overline{l}),(Hu,c)) = [h,\overline{l}]$$

$$output((\overline{h},\overline{l}),(Hu,c)) = [\overline{h},\overline{l}]$$

$$output((h,l),(Li,c)) = [l]$$

$$output((\overline{h},l),(Li,c)) = [l]$$

$$output((h,\overline{l}),(Li,c)) = [\overline{l}]$$

$$output((\overline{h},\overline{l}),(Li,c)) = [\overline{h},\overline{l}].$$

For system act series

$$\alpha = (Hu, flip)(Li, flip).$$

The initial state of the system is $s_0 = (h, l)$. Pay attention that the initial state can be any trustworthy state of the system. According to Definition 8, check whether the transmission of the system trustworthiness chain is trustworthy or not.

(i) From initial state $s_0 = (h, l)$, the system executes α sequentially. System states are

$$step((h, l), (Hu, flip)) = (\overline{h}, l)$$

$$step((\overline{h}, l), \alpha) = (h, \overline{l}).$$

(ii) For two safe domains, calculate $dipurge(\alpha, L, (h, l))$, and $dipurge(\alpha, H, (h, l))$.

For safe domain L

$$dsource(\Lambda, L, (h, \overline{l})) = \{L\}$$

$$dsource((Li, flip), L, (\overline{h}, l)) = \{L\}.$$

Because $dom((Hu, flip)) = H$, and $H \overset{(h,l)}{\sim} > L$,

$$dsource(\alpha, L, (h, l)) =$$

$$dsource((Li, flip), L, (h, l)) = \{L\}.$$

Because $dom(Li, flip) = L$, and $dsource((Li, flip), L, (\overline{h}, l)) = \{L\}$,

$$dipurge((Li, flip), L, (\overline{h}, l)) =$$

$$(Li, flip)dipurge(\Lambda, L, (h, \overline{l})) = (Li, flip).$$

Because $dom(Hu, flip) = H$, and $dsources(\alpha, L, (h, l)) = \{L\}$ and $H \notin \{L\}$,

$$dipurge(\alpha, L, (h, l)) =$$

$$dipurge((Li, flip), L, (\overline{h}, l)) = (Li, flip).$$

Namely,

$$dipurge(\alpha, L, (h, l)) = (Li, flip).$$

For safe domain H,

$$dsource(\Lambda, H, (h, \overline{l})) = \{H\}.$$

Because $dom((Li, flip)) = L$, and $L \overset{(\overline{h},l)}{\sim} > H$,

$$dsource((Li, flip), H, (\overline{h}, l)) =$$

$$\{L\} \cup dsource(\Lambda, H, (h, \overline{l})) =$$

$$\{L\} \cup \{H\} = \{L, H\},$$

$$dsource(\alpha, H, (h, l)) = \{L, H\}.$$

Because $dom((Li, flip)) = L \in dsources((Li, flip), H, (\overline{h}, l)) = \{L, H\}$,

$$dipurge((Li, flip), H, (\overline{h}, l)) =$$

$$(Li, flip)dipurge(\Lambda, H, (h, \overline{l})) = (Li, flip).$$

Because $dom((Hu, flip)) = H \in dsources(\alpha, H, (h, l)) = \{L, H\}$,

$$dipurge(\alpha, H, (h, l)) =$$

$$(Hu, flip)dipurge((Li, flip), H, (\overline{h}, l)) =$$

$$(Hu, flip)(Li, flip).$$

Namely, $dipurge(\alpha, H, (h, l)) = \alpha$.

(iii)

$$do(\alpha) = run((h, l), \alpha) = (h, \overline{l}),$$

while

$$do(dipurge(\alpha, L, (h, l)) =$$

$$step((Li, flip), (h, l)) = (\overline{h}, \overline{l}).$$

(iv)

$$test(\alpha, (Li, slip)) =$$

$$output((h, \overline{l}), (Li, slip)) = [\overline{l}],$$

$$test(dipurge(\alpha, L, (h, l)), (Li, slip)) =$$

$$output((\overline{h}, \overline{l}), (Li, slip)) = [(\overline{h}, \overline{l}],$$

$$test(\alpha, (Li, slip)) \neq$$

$$test(dipurge(\alpha, L, (h, l)), (Li, slip)).$$

Therefore, according to Definition 8, for the given safe rule, the transmission of this trustworthiness chain is untrustworthy, because the act $(Hu, flip)$ in the safe domain H interferes with the safe domain L.

For Case 2, the model is:

State aggregate:

$$S = \{\phi, (s_A, s_C), (s_{\overline{A}}, s_{\overline{C}}), (s_A, s_{\overline{C}}), (s_{\overline{A}}, s_C)\}, s \in S.$$

ϕ means empty, s_A and s_C mean A and C have not been updated respectively, and $s_{\overline{A}}$, $s_{\overline{C}}$ mean A and C have been updated respectively. For simplifying the process, the updating here is regarded as an atom act which is completed instantly. Communication between A and C will not change the state.

Act aggregate: $A = \{a_A, a_C\}$, which means both virtual machines A and C are updated. To make it simple, communication between A and C is not considered.

Safe domain: $D = \{u_H, u_A, u_C\}$.

To make it simple, the domain in which the trustworthy channel is located is not considered.

Output aggregate: $O = \{\phi, \{o_A\}, \{o_C\}, \{o_A, o_C\}\}$ means the information left in HyperVisor by the updating act.

Function: $dom = \{(a_A, u_A), (a_C, u_C)\}$.

Security policies:

$$\sim>= \{(u_H, u_A, (s_A, s_B)),$$
$$(u_H, u_A, (s_{\overline{A}}, s_B)),$$
$$(u_H, u_A, (s_A, s_{\overline{B}})),$$
$$(u_H, u_A, (s_{\overline{A}}, s_{\overline{B}})),$$
$$(u_H, u_B, (s_A, s_B)),$$
$$(u_H, u_B, (s_A, s_{\overline{B}}))\}.$$
$$\sim>= \{(u_H, u_B, (s_{\overline{A}}, s_{\overline{B}})),$$
$$(u_H, u_B, (s_{\overline{A}}, s_B))\}.$$

This means that when the virtual machine A is updated, the system does not allow the information to flow from u_H to u_C.

Functions $step$ and $output$:

$$step((s_A, s_C), a_A) = (s_{\overline{A}}, s_C)$$
$$step((s_A, s_{\overline{C}}), a_A) = (s_{\overline{A}}, s_{\overline{C}})$$
$$step((s_{\overline{A}}, s_C), a_A) = (s_{\overline{A}}, s_C)$$
$$step((s_{\overline{A}}, s_{\overline{C}}), a_A) = (s_{\overline{A}}, s_{\overline{C}})$$
$$step((s_A, s_C), a_C) = (s_A, s_{\overline{C}})$$
$$step((s_{\overline{A}}, s_C), a_C) = (s_{\overline{A}}, s_{\overline{C}})$$
$$step((s_A, s_{\overline{C}}), a_B) = (s_A, s_{\overline{C}})$$
$$step((s_{\overline{A}}, s_{\overline{C}}), a_C) = (s_{\overline{A}}, s_{\overline{C}})$$
$$output((s_A, s_C), a_A) = \{o_A\}$$
$$output((s_A, s_{\overline{C}}), a_A) = \{o_A, o_C\}$$
$$output((s_{\overline{A}}, s_C), a_A) = \{o_A\}$$
$$output((s_{\overline{A}}, s_{\overline{C}}), a_A) = \{o_A, o_C\}$$
$$output((s_A, s_C), a_C) = \{o_C\}$$
$$output((s_A, s_{\overline{C}}), a_C) = \{o_C\}$$
$$output((s_{\overline{A}}, s_C), a_A) = \{o_A, o_C\}$$
$$output((s_{\overline{A}}, s_{\overline{C}}), a_C) = \{o_A, o_C\}.$$

For system act series $\alpha = a_A a_C$, the initial state of the system is $s_0 = \phi$. Note that the initial state can be any trustworthy state in the system. Through analysis, it can be found that the process is similar to circumstance 1. According to Definition 8, it can be judged that the system has information leakage. Therefore, the transmission of the trustworthiness chain corresponding to the act is untrustworthy.

5.2 Realization of the prototype system

This paper has provided an analysis method for processing dynamic trustworthiness. According to this method, a prototype system that supports the certification of system dynamic trustworthiness is designed and realized.

The prototype system adopts bi-kernel trustworthy architecture (bi-kernel TRA). It is composed of the super kernel and the user kernel. The super kernel is an unchangeable part. It can access all data in the user-kernel which is regarded as a process operation. In order to support the certification of system dynamic trustworthiness, the bi-kernel TRA is improved and the improved prototype is called the dynamic trustworthiness system based on bi-kernel (DT-biK). The structure is shown in Fig. 5.

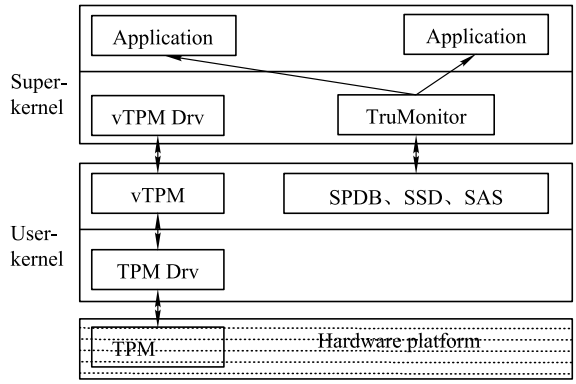


Fig. 5 Dynamic trustworthiness system based on bi-kernel

The lowest level of the DT-biK is the hardware level including TPM. The upper one of it is the super-kernel level, and the topmost level is the application system level user-kernel. There is a TruMonitor module embedded in the user-kernel. It can monitor and intercept the operations of all the application programs. The super-kernel level includes a security policies database of application systems and other data structures that need to be maintained. The user-kernel can access virtual TPM (vTPM) provided by the super-kernel level through the virtual TPM driver (vTPM Drv) and the vTPM of the super-kernel level can access the real TPM through vTPM Drv.

To make it simple, in the prototype system, the focus is only on acts that are related to the process and document access, concentrating on Case 1. The system needs to

maintain the following three data structures.

SPDB: A security policy database used to describe the dynamic interference relationship among processes;

SSD: A system state descriptor used to record the current state of the system;

SAS: A system act sequence used to record executed act series of the system.

According to Definition 1, the trustworthiness certification algorithm of $Trusted(R_A)$ of the software entity R_A can be divided into two phases. The first phase is to test $Load\_Trusted(M_A)$ and the second phase is to test $Run\_Trusted(P_A)$.

During the loading process of software entity, $Load\_Trusted(M_A)$ measures and tests the completeness of M_A by adopting the method given by reference [11], so as to ensure static trustworthiness of R_A.

$Run\_Trusted(P_A)$'s test opportunity happens when the current document executes the files. The TruMonitor intercepts the system's scheduling and tests the trustworthiness of the wait-to-be scheduling process $curProc$. The test algorithm is as follows.

Step 1 Record the act of $curProc$ as Act. P is the current system process line and p is the head of P. Make the current process in a trustworthy state.

Step 2 Calculate $O_1 := test(SAS, Act)$.

Step 3 If p is empty, leap to Step 8.

Step 4 Calculate $dipurge(SAS, P, SSD)$.

Step 5 Calculate $O_2 := test(dipurge(SAS, P, SSD), Act)$.

Step 6 If $O_1 <> O_2$, make the process in an untrustworthy state and leap to Step 8.

Step 7 $p := p$->next, leap to Step 3.

Step 8 If the process is in a trustworthy state, then $SAS := SAS \circ Act$, modify $SSD := step(SSD, Act)$.

Step 9 Return to the state of the process.

With the realization thought of RTLinux [24,25], the DT-biK on Loongson 2F is designed and realized. As Fig. 6 shows, the super-kernel is modified by adopting GDB-STUB. It is a super kernel. The user-kernel is modified on Linux core 2.6.18. Since the super-kernel is modified by GDB-STUB, it can be independent of the user-kernel code that is modified based on the standard Linux core. They can be independently stored physically and trustworthiness of the module increases.

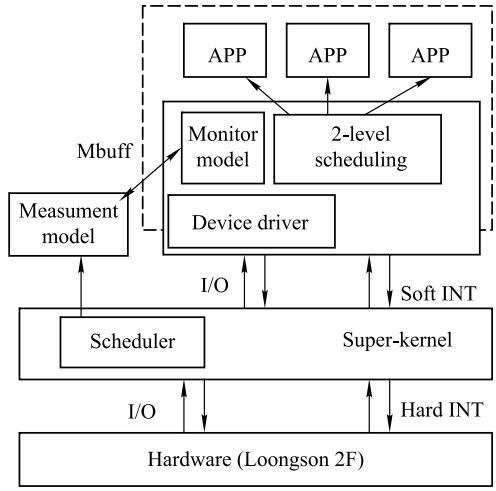


Fig. 6 Realization structure of DT-biK based on Loongson 2F platform

For the interruption management, the super-kernel uses the hardware interruption of Loongson IP2-IP7 and isolates the user-kernel from the interruption controller through soft interruption. This is realized by replacing the interruption switch function (cli and sti), interruption return function (ret_from_irq), interruption processing function and interruption vector table of Linux. Interruption processing of the user-kernel is replaced by soft interruption, which means using a group of variables to record the interruption switch and happening conditions. However, all hardware interruptions are intercepted by the super-kernel and whether it is necessary to conduct interruption groups to the user-kernel is decided by the interruption type and mark.

On task scheduling, the super-kernel redefines the scheduling function. The user-kernel is regarded as the low priority of the super-kernel, while the normal user process can still operate on the user-kernel and use all kinds of services provided by the user-kernel.

As the "real process" running in the super-kernel, the communication between the measurement module and the user-kernel still adopts the Mbuff communication mechanism of RTLinux.

In the prototype system of DT-biK, a simulation test for Case 1 is conducted. The simulation result shows that Algorithm 1 can identify dynamic trustworthiness of the current process operation.

6. Conclusions

Focusing on the analysis of software trustworthiness during the transmission of the trustworthiness chain, the paper analyzes the impact of the system state on process trustworthiness. It adopts access control to establish the software operation state mutual interference act model. The

access control matrix of the mutual interference act of the trustworthy entity caused by the memory, file/equipment handle can be directly transformed to codes so as to provide software analysis tools for system dynamic trustworthiness. It can be further used for collection/analysis of software operation state acts. Based on the DINI theory, the paper puts forward a system dynamic trustworthy form judgment theorem, which provides theoretic support for trustworthiness calculation. In addition, sufficient conditions of the judgment theorem in this paper are too strict for the entity trustworthiness judgment, and [26] gives a sufficient requisite condition which is used to check whether a system meets the INI model and provides the corresponding algorithm. The algorithm is divided into two phases. First of all, introduce an iP-observability attribute into the INI system and prove that the system will meet the INI only when it is regarded as iP-observability. And then, through the automaton transformation, a relationship between iP-observability and P-observability is tested. IP-observability is transformed to P-observability in the automaton so as to check whether the corresponding system meets INI [26] through the judgement of P-observability in the automaton. This provides a thought and basis about how to check whether a system meets DINI. The following plan of the research is to study the transformation method between DINI and the automaton and provide the requisite condition and the corresponding algorithm with which the system will meet the dynamic trustworthiness judgment theorem.

References

[1] TCG Group. TCG specification architecture overview. TCG Specification Revision. http://www.trustedcomputinggroup.org.

[2] H. G. Zhang, J. Luo, G. Jing, et al. Trusted computing research. *Wuhan University: Natural Science Edition,* 2006, 52(5): 513 – 518. (in Chinese)

[3] C. S. Dong. A sociological study of credibility and its typology. *Journal of Hebei Normal University (Philosophy and Social Science)*, 2004, 27(1): 40 – 43. (in Chinese)

[4] D. E. Zand. Trust and managerial problem solving. *Administrative Science Quarterly*, 1972, 17(2): 229 – 239.

[5] H. H. Brower, F. D. Schoorman, H. Tan. A model of relational leadership: the integration of trust and leader–member exchange. *Leadership Quarterly*, 2000, (11): 227 – 250.

[6] R. C. Mayer, F. D. Schoorman, J. H. Davis. An integrative model of organizational trust. *Academy of Management Review*, 1995, 20(3): 709 – 734.

[7] T. Beth, M. Borcherding, B. Klein. Valuation of thust in open network. *Proc. of the European Symposium on Research in Computer Security*, 1994: 3 – 18.

[8] J. Patel, T. W. T. Luke, N. R. Jennings, et al. A probabilistic trust model for handling inaccurate reputation sources. *Proc. of the Third International Conference on Trust Management*, 2005: 193 – 209.

[9] W. Tang, Z. Chen. Research of subjective trust management model based on the fuzzy set theory. *Journal of Software*, 2003, 14(9): 1401 – 1408. (in Chinese)

[10] W. Tang, J. B. Hu, Z. Chen. Research on a fuzzy logic-based subjective trust management model. *Journal of Computer Research and Development*, 2005, 42(10): 1654 – 1659. (in Chinese)

[11] J. Q. Mo, Z. W. Hu, X. L. Ye. Research of trust assessment method in trust computing based on fuzzy theory. *Journal of Computer Application*, 2013, 33(1): 142 – 145. (in Chinese)

[12] X. X. Wang, X. Y. Kong, X. B. Chen. A dynamic noninterference trust chain model based on security process algebra. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 2014, 12(1): 747 – 752.

[13] L. L. Yuan, G. S. Zeng, W. Wang. Trust evaluation model based on dempster-shafer evidence theory. *Journal of Wuhan University (Natural Science Edition)*, 2006, 52(5): 627 – 630. (in Chinese)

[14] Y. W. Qu. *Software Behavior*. Beijing: Electronic Industry Press, 2004. (in Chinese)

[15] R. Sailer, X. Zhang, T. Jaeger, et al. Design and implementation of a TCG-based integrigty measurement architecture. *Proc. of the 13th USENIX Security Symposium*, 2004: 223 – 238.

[16] T. Jaeger, R. Sailer, U. Shankar. PRIMA: Policy reduced integrity measurement architecture. *Proc. of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT2006)*, 2006: 134 – 143.

[17] J. Zhao, C. X. Shen, J. Q. Liu, et al. A noninterference-based trusted chain model. *Journal of Computer Research and Development*, 2008, 45(6): 974 – 980. (in Chinese)

[18] X. Zhang, Y. L. Chen, C. X. Shen. Non-interference trusted model based on processes. *Journal on Communications*, 2009, 30(3): 6 – 11. (in Chinese)

[19] X. Zhang, Q. Huang, C. X. Shen. A formal method based on noninterference for analyzing trust chain of trusted computing platform. *Chinese Journal of Computers*, 2010, 33(1): 74 – 81. (in Chinese)

[20] F. Zhang, M. Jiang, H. G. Wu, et al. Approach for trust analysis of software dynamic behavior based on noninterference. *Computer Science*, 2012, 39(1): 101 – 103. (in Chinese)

[21] J. Goguen and J. Meseguer. Security policies and security models. *Proc. of IEEE Symposium on Security and Privacy*, 1982: 11 – 20.

[22] J. Rushby. *Noninterference, transitivity, and channel – control security policies*. Menlo Park Stanford Research Institute, 1992.

[23] L. Rebekah. Dynamic intransitive noninterference. *Proc. of the IEEE International Symposium on Secure Software Engineering*, 2006, 65 – 74.

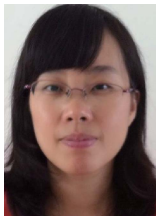
[24] X. Y. Kong, X. B. Chen, Y. Zhuang. A dynamic trustworthiness attestation method based on dual kernel architecture. *International Journal of Hybrid Information Technology*, 2013, 16(5): 237 – 248.

[25] C. E. All. A real-time Linux system for autonomous navigation and flight at titude control of an uninhabit edaerial vehicle. *Proc. of the 20th Conference on Digital Avionics Systems*, 2001: 1A11 – 1A19.

[26] N. Ben Hadj Alouane, S. Lafrance, F. Lin, et al. Characterizing intransitive noninterference for 3-domain security policies with observability. *IEEE Trans. on Automatic Control*, 2005, 50(6): 920 – 925.

Biographies

Xiangying Kong was born in 1972. He received his B.S. degree in Xi'an Jiaotong University in 1995. He is currently a Ph.D. candiadate of Nanjing University of Aeronautics and Astronautics and a professor of Jiangsu Automation Research Institute. He presided over the development of multi-type electronic equipment and national defense pre-research project. His research interests include software engineering, information security and real-time operating system.
E-mail: kongxy716@aliyun.com



Yanhui Chen was born in 1973. She received her B.S. degree from Department of Computer Science, Southeast University. She is currently a senior engineer of Jiangsu Automation Research Institute. She presided over the development of multi-type automatic control system and C3I system. Her research interests include software engineering, information security and system engineering.
E-mail: chenyh@jari.cn

Yi Zhuang graduated from Department of Computer Science, Nanjing University of Aeronautics and Astronautics in 1981. She is now a Ph.D. supervisor in Nanjing University of Aeronautics and Astronautics. She has published over 50 papers in core journals or academic conferences at home and abroad. Besides, she has presided more than thirty state or provincial projects, or scientific and technological cooperation projects. Her research interests include network and distributed computing, and information security.
E-mail: zhuangyi@nuaa.edu.cn