Software Trusty Analysis Based on Taint Trace with Non-interference

Chen Shu
School of computer science
HuaZhong normal university
WuHan China
Chenshu181@aliyun.com

Chen Ming Kai
Information Station Department
HuBei Military Area
WuHan China

Ye jun Min
School of computer science
HuaZhong normal university
WuHan China

Zhang Fan
Communication School
HangZhou DianZi University
HangZhou 3 China

Abstract—we proposed a novel approach for software trust analysis, in order to solve the problems in auto testing that, the test process is lack of guidance, further more, with high performance cost. The main idea of this approach is based on taint data tracing by taint tracing started from outside of software environment. By extracting the raw behaviors such as API invocation that may cause un-trust consequence, the scope of codes that being tested will be narrowed. These may-un-trust behaviors then form into a taint dependency behaviors model, and will be proved that whether these behaviors can be trusted in certain environment. The proving mechanism will be done by a so called non-interference information flow model. The behavior model will be evolved by refinement, and the context of each test iteration will be taken advantaged of for guidance. Besides, non-interference theories will also be applied in proving. We also briefly discussed about the implementation of this approach, and necessary theorems for this approach are also proved.

Keywords-component; software trust ;software behavior; tiant; non-interference;behavior model

I. INTRODUCTION

Measurement of software trusty is always a hot spot in information security research area. Traditional approach based on static analysis applied the comparison of harsh code of software to determine whether the software has been illegally modified, however, not able to discover its self-owned vulnerability, nor able to discover the malicious behaviors during run-time. For example, the attacker may damage the system or gain unauthorized permissions by the using of buffer leakage or formatted strings. To solve these problems, dynamic analysis approaches are proposed. By analyzing dynamic behaviors of software, malicious actions are targeted in runtime. Even TCG (Trusted Computing Group) has proposed the concept of trusted computing, and invented the definition of trust in dynamic perspective, and focus more on the predictability and controllability of software behaviors. TCG kept the guideline that "An entity of software can be considered as trust if it achieves its expected result by behaviors or act in an expected way" [1, 2].

As far as we have concerned, dynamic analysis of software is only at its beginning stage, lots of unsolved problems need further research. For example, we are lack of efficient method to extract and analysis runtime behavior; we are also lack of a unified trusted proving system [2]. According to the definition of TCG, a unified trust proving model contains two parts at least: the first is a reasonable behavior model to simulate the runtime behaviors of software. The second is proving and deducing method based on behavior model. According to this theory, we proposed a model which is based on taint data and noninterference theory to dynamically analysis the behavior of software. The main contribution contains the following:

1). A analysis and tracing approach of software behavior is proposed based on data tainting. Suspicious data is tainted and established the dependency relationship among that suspicious system invocation which produce or consume the taint data, thus dramatically decrease the analysis complexity, as well as the impact of code obscure technology.

2). Proposed a trust deductive method based on behavior model.

3). Proposed a unified model for trust analysis.

The paper is organized as follows: section 2 introduce the related works, section 3 discuss about the behavior analysis method based on taint tracing, section 4 states about the trust deduction based on behavior model, section 5 briefly introduce the implementation. Conclusion will be showed in the final section.

II. RELATED WORKS

Taint analysis was first proposed by Dawn Song [3, 4], who tainted suspicious un-trusted input data to help trace and analysis suspicious malicious behaviors. Taint data is those data that may cause un-trusted facts. The source that taint data come from is called taint source, including

© 2014. The authors - Published by Atlantis Press

keyboard input, internet socket, etc. System invocation is the interface between application software and operation systems, almost all functions of application software are implemented by system invocations. Kevin and Wang [5, 6] claim that most malicious behaviors caused by malicious code are resulted from redirection of normal execution to certain malicious instructions, which was previous designed by attracter and injected in certain memory space, such as buffer leakage. By using the trace of taint data, certain malicious behaviors can be sketched and reoccur, in order to represent how certain system calls are illegally used to achieve certain unauthorized goals. For more, to establish more strict and specific security strategies. Thus, the monitoring on critical system calls, or in other speaking, monitoring the proliferation of taint data can be used to improve the accurate of analysis malicious behaviors [7]. Song and Ming [3,4] applied taint data to characterize the critical information flow, Kevin and Wang [5, 6] used taint data to eliminate the code obscure, how ever, these researches are only limited in special circumstances and lack of a theoretical support.

In the perspective view of system running, the return of system call and the argument transition can be recognized as information flow, and the non-interference theory proposed by rushby [8,9,10,11] is a typical model based on information flow. Non-interference model is widely used in explaining the access and control strategies of security systems. Zhang [12] applied non-interference to represent the trust issue of information proliferation.

As we have discussed above, our approach employed taint data to construct a dependency graph of critical system calls in order to represent the model of suspicious behaviors, and applied non-interference to analysis the trust issue on the model, thus, composing a unified model on trust analysis.

III. DYNAMIC ANALYSIS OF BEHAVIORS BASED ON TAINT DATA

The following section is based on the following symbolic agreement: Upper case 'A' is used to represent the sequence of system calls, while lower cases 'a' for individual system call. A system call is defined as a three group: $a = \{ret, par_i, par_o\}$ in which *ret* means the return data, par_i means the arguments passed in, and par_o means the arguments passed out.

We invented an algorithm called TF(*twice filter*) to establish the dependency graph based on taint data. The main idea of TF is described as following:

1). First filter: consider two system calls a and b, trace the taint source of system call a, if $a.par_o$ or $a.ret$ propagates to $b.par_i$, then record that b is data-dependent on a. Repeat the process until the taint data is no long propagate or the program terminates. The production of the first filter is a dependency graph $G = \{in, out, A, E\}$ in which *in* is the entry point, *out* is the exit point, A is the collection of all system calls in the graph, edge E represented as $A \times A$ describes the dependency relations. As multiple entries may exist, thus we may have more than one dependency graphs, we describe them as a set: $GS = \{G_1, G_2, , , G_n\}$.

2). Second filter: for each graph $G_i \in GS$ in GS, we start from the exit point to scan each calls reversely by depth first strategy. For each system call $a_i = \{ret, par_i, par_o\}$, if par_i is taint, then we put it into another collection S. If system call a has defined some memory space $l \in S$,then mark a as taint dependent and eliminate l from S, for more add the memory address that is used by a to S. Repeated the process until set S is empty or reach the entry point. Delete all nodes that were not marked as taint in G_i as well as the edges that related to those nodes. The purpose of this process is to find the nearest call before a or the call that modify taint data m when system call a is visiting data m.

According to TF, for any $G_i \in GS$, as it has an entry and an exit point, G_i can be considered as a composition of several sequence of system calls from *in* to *out*. We then decompose G_i as the set of several sequence of system calls that may overlap as $G_i = \{A_{i1}, A_{i2}, , A_{in}\}$. According to the definition of TCG, for a software system S, if all $A_{ij} \in G_i$ in all $G_i \in GS$, if the execution status is met as expected, then S can be considered as trusted, thus the following work in section 4 will be the decision made on whether sequence A_{ij} is as expected.

IV. TRUST DEDUCTION BASED ON TAINT FLOW AND NON-INTERFERENCE

According to the definition of TCG, An entity of software can be considered as trust if it achieves its expected result by behaviors or act in an expected way. For call sequence A, if the result of A is the same as expected, then A can be considered as trust. We applied non-interference theory, and treat illegal jump with taint data as interference. If the sequence A violates the non-interference strategy, then we say that A has malicious behaviors with taint data. We also assume that our program is running under a structured machine, which will be introduced soon.

A. Trust Deduction with Non-interference

We first give a few definitions as following.

Definition 1: A structured machine is defined as an eight group: $M = \{S, Call, O, D, P, do, out, process\}$ in which *Call* is the collection of system call, O is the set of output, D is a set of security domains in which the process that contains system calls belongs to. For all $a \in Call$, we have $dom(a) \in D$, P is the set of security strategies, defined as $p : D \text{ a } D$, in which $p \in P$, binary relationship a represent that two security domains has taint information flow between them. Its complement $\not{a}$ means no taint information has interfered between them; N is the name space of machine M, V is the set of values accord with N, S is a set of states, each $s \in S$ represents an ordered pair between N and V,

do is single step execution, which represent state transition, defined as: $do : S \times Call \to S$, *output* is the return value of certain system call, defined as: $out : S \times Call \to O$, $process$ means the state produced after an execution of a sequence of system call, defined as: $process : S \times A^* \to S$. Let τ as an empty call, o is connection operation, the following recursive property is satisfied：

$$\begin{cases} process(s, \tau) = s \\ process(s, a \circ A) = process(do(s, a), A) \end{cases}$$

Definition 2: Taint information flow. Assume n is a memory space area on which behaviors in a certain security domain $d_1 \in D$ have the write authority, other behaviors in domain $d_2 \in D$ having only read authority. If n has been marked as taint, then taint flow exist between domain d_1 and d_2 under non-interference strategy. Formally defined as:

$$\exists n \in Taint : n \in write(d_1) \wedge n \in read(d_2) \Leftrightarrow d_1 \text{ a } d_2$$

Unauthorized behaviors caused by taint flow is also called unauthorized control flow, the system call that related to taint flow is called taint call sequence.

Definition 3: The runtime memory space for a structured machine M is defined as a five group: $R = \{N, V, read, write, observe\}$, in which N is the set of identity for each memory unit, V is the corresponding value set, *read* and *write* are two basic operations, *observe* is observation.

Definition 4: Function $origin : Call^* \times D \to P(D)$, gets all security domains in taint call sequence:

$$\begin{cases} origin(\tau, u) = u \\ origin(a \circ A, u) = origin(A, u) \cup dom(a) \\ \qquad\qquad if\ \exists v \in origin(A, u) \wedge dom(a) \text{ a } u \\ origin(a \circ A, u) = origin(A, u)\ else \end{cases}$$

Definition 5: If structured machine M has a multiple view which are observable, then for all security domains $u \in D$, an equivalence relation exist: $s[\approx u]t$, which represents that the value of state s and t observed in security domain u are the same. Formally defined as:

$$s[\approx u]t \Leftrightarrow \forall n \in observe(u) : s(n) = t(n)$$

Definition 6: Function $filter : Call^* \times D \to Call^*$ represents the gain of sub-sequence that after filtering all system calls that has taint inference with specific security domain:

$$\begin{cases} filter(\tau, v) = \tau \\ filter(a \circ A, u) = a \circ filter(A, u)\ if\ dom(a) \in origin(A, u) \\ filter(a \circ A, u) = filter(A, u)\ else \end{cases}$$

Taint sequence as well as its taint data dependency records the proliferation of taint data during critical system calls, these taint data may be capitalized by attackers. With the definition of trust proposed by TCG, we can concluded that if a system is running in an security environment, if the proliferation of taint data is the same as expected, the system is trusted.

Definition 7: Taint call sequence A is trusted against to non-interference strategy if the environment of A is screening between layers and no illegal taint control sequence exists:

$$out(process(s_0, A), a)$$
$$= out(process(s_0, filter(A, dom(a))), a)$$

Definition 7 provides a rule to decide whether a system is trusted. The left side of the expression represents the actual out put of sequence A; the right side represents the expected output under non-inference strategy. For the sake of understanding, we also extend definition 7 into single step representation.

Theorem 1: taint call sequence A is trusted if and only if it satisfied the following properties:

1). Single step screening:

$$s[\approx u]t \Rightarrow out(s, a) = out(t, a)$$

2). Consistence of state equivalence:

$$s[\approx u]t \wedge s[\approx dom(a)]t \Rightarrow do(s, a)[\approx u]do(t, a)$$

3). Non-interference of taint data propagation:

$$dom(a) \not\text{a} \ u \Rightarrow s[\approx u]do(s, a)$$
$$\forall n \in observe(u) : s(n) \neq do(s, a)(n) \Rightarrow dom(a) \text{ a } u$$
$$dom(a) \text{ a } u \Rightarrow \exists n \in observe(u) : s(n) \neq do(s, a)(n)$$

Proof：According to 1), the Necessary and sufficient condition that definition 7should meets is to prove (a):

$$\begin{matrix} process(s_0, A)[\approx dom(a)] \\ process(s_0, filter(A, dom(a))) \end{matrix} \quad (a)$$

Expression (a) can be induce to expression (b)：

$$process(s_0, A)[\approx u]process(s_0, filter(A, u)) \quad (b)$$

We applied mathematical induction on the length of A to prove (b). When $A = \tau$, (b) is inevitably satisfied, when the length of A is bigger then 0, our proving target is becoming to when (c) is satisfied then $a \circ A$ should also be satisfied, which referenced to as (d).

$$s[\approx u]t \Rightarrow process(s, A)[\approx u]process(t, filter(A, u)) \quad (c)$$

$$\begin{matrix} s[\approx u]t \Rightarrow \\ process(s, a \circ A)[\approx u]process(t, filter(a \circ A, u)) \end{matrix} \quad (d)$$

We can also applied mathematical induction on the length of A and reduction to absurdity approach on the assumptions of expression $dom(a) \in origin(i \circ A, u)$ and

$dom(a) \notin origin(i \circ A, u)$ respectively to prove (d). For the limit of space, we omit the proving process.

We have discussed and proved about the deduction of trust on taint call sequence; we will also give a more reasonable explanation under structured environment.

B. Explanation Under Structured Environment

Let function $content : S \times N \to V$ represent getting the value of memory n under state s, in which n means the state that could be observed in security domain u, including process state, register state. Thus, we modify the expression of state equivalence as following:

$$s[\approx u]t \Leftrightarrow \forall n \in observe(u) : content(s, n) = content(t, n)$$

Let function $shift : Call \times D \to P(D)$ represent getting the security domains that system call a can shift to under security domain u ⊤, and satisfy the following property:

$$\forall a \in Call : u \in shift(a, dom(a)) \Leftrightarrow$$

$$ring(dom(a))\ ]\quad ring(u) \wedge (dom(a)\ \text{a}\quad u) \in P$$

Function $ring(u)$ means the security level of domain u.

Definition 8: Consistency within a domain represent that the state remains consistent after a taint system call was executed in the same security domain. Formally represented as following containing expression:

$$\forall n \in observe(dom(a)) : content(s, n) = content(t, n) \Rightarrow$$

$$\forall n \in observe(dom(a)) :$$

$$content(do(s, a), n) = content(do(t, a), n)$$

Definition 9: One-way property between domains represents that when taint data is transmitted between domains, it should flow through ordered layer. Formally represented as following containing expression:

$$\forall n \in observe(u) :$$

$$content(s, n) = content(t, n) \wedge ring(dom(a))\ ]\quad ring(u)$$

$$\Rightarrow \forall n \in observe(u) : content(do(s, a), n)$$

$$= content(do(t, a), n)$$

Definition 10: Consistency between domains represent that data should be transmitted in an authorized way between domains, formally represented as following containing expression:

$$\forall n \in observe(u) : content(do(s, a), n) \neq content(s, n) \Rightarrow$$

$$n \in write(dom(a)) \wedge u \in shift(a, dom(a))$$

The following Theorem proved that in a structured environment, if any one of definition 8 to 10 is not satisfied on a taint call sequence, then the sequence is not trusted.

Theorem 2: Given a taint call sequence A, if A is running under a structured environment, but not all of 8,9,10 are satisfied, then A is not trusted.

The proving process on theorem 2 can be fulfilled by applying reduction to absurdity on definition 8, 9, 10 respectively, such that theorem 1 which deduced form the trust definition of TCG is not satisfied. We omit the proving detail because of the space limit.

V. IMPLEMENTATION

We have also implemented the prototype under Ubuntu 9.04-32bit, the structure of the system is showed in Fig .1

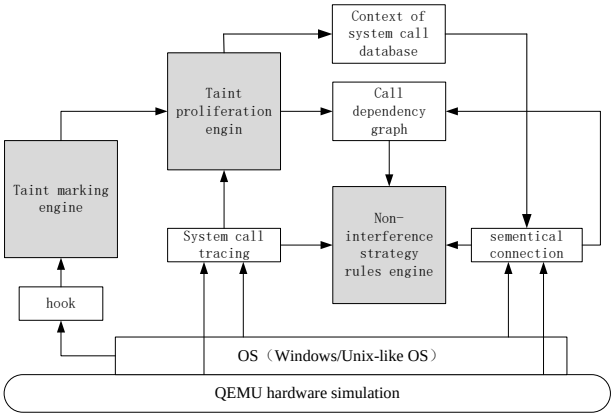

Figure 1. Structure of trust analysis based on taint and non-interference

The prototype is based on QEMU hardware simulation, in order to doing analyzing on varies of operation systems. The system is mainly divided into three parts: taint marking engine, taint proliferation engine and rule decision engine. Taint marking engine take the responsibility of tainting data that we are interested in (keyboard input, network adapter I/O) by hooking. Taint proliferation engine records all tracing of system calls which propagating the taint data in operation level, and generate call dependency graph by algorithm tf. contex of system call database stores all critical system info such as register info, system call info, secuity domain info. Rule decision engine will decide whether the targeted program is trusted by throrem 2.

VI. CONCLUSION AND FURTHER WORK

Dynamic software analysis posed a new direction for software trust measurement, however, the variations in software implementation, the ambiguity that imposed by malicious code, and the mutation speed the malicious code has, built difficulties in behavior analysis. To solve this problem, we propose an approach by using taint injection, trace the taint data flow and generate a system call dependency graph that related to taint proliferation. For more, we proposed theory lines to prove a system is trusted or not by applying non-interference theory on system call dependency graph and established a unified framework for trust analyzing.

How ever, it seems a challenge to describe and divide domain in some operation system such as windows, and lack of isolation mechanism [11]. Thus in our further work, more research about the more rational approach on

explanation of domain and environment will be included. For more, more optimization on the generation of dependency graph especially in complex system should also be taken into consideration.

ACKNOWLEDGMENT

This research was financially supported by the Scientific Key project of National Language Committee (Project ID: ZDI125-21).

REFERENCES

[1] David Challener, Ryan Catherman, A practical Guide to Trusted Computing [M], Machinery Industry Press, 2009.

[2] ShenChangxiang，ZhangHuanGuo, research and development in trust computing [J]. science china-information, 40(2)：139-16, 2010.

[3] James Newsome, Dawn Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software[C]. In Proceedings of the Network and Distributed Systems Security Symposium (NDSS), pp:56-73, Feb 2005.

[4] Min Gyung Kang, Stephen Mc Camant, Pongsin Poosankam, Dawn Song. DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation [C]. In Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS),2011.02, SanDiego, CA, pp: 67-81, 2011.

[5] Kevin Coogan, Gen Lu, Saumya Debray. Deobfuscation of Virtualization Obfuscated Software A Semantics-Based Approach [C]. CCS 2011, Chicago, Illinois, USA, pp: 17–21, 2011.

[6] WangRui, FengDengGuo, Semantics-Based Malware Behavior Signature Extraction and Detection Method [J], journal of software, 23(2):378-39, 2012.

[7] Li Y, Zuo ZH. An overview of object code obfuscation technologies. Journal of Computer Technology and Development [J], l7(41)：125-127, 2007.

[8] Ushby J, Noninterference, Transitivity, and Channel-Control Security Policies[R]. Computer Science Laboratory, SRI International，2005.

[9] Haigh J T，Yong W D. Extending the noninterference model of MLS for SAT [C]. IEEE Transaction On Software Engineering，2(13)：141-150, 1987.

[10] Goguen J A, Meseguer J. Security policies and security models [C] Proc. of the 1982 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, pp: 11-20, April 1982.

[11] ZhaoJia, ShenChangXiang, A noninterference-based trusted chain model [J]. journal of computer research and development, 45(6):974-980, 2008.

[12] ZhangFan, ChenShu, Noninterference model for integrity [J], journal on communications, 32(10):11-19, 2011.

[13] Fabrice Bellard. QEMU documentation [R]. 2011. http://wiki.qemu.org/Main_Page.