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; taint; 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
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 attacker 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, however, 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: \( a = \{ \text{ret, par}_a, \text{par}_b \} \) in which ret means the return
data, \( \text{par}_b \) means the arguments passed in, and \( \text{par}_a \) means
the arguments passed out.

We invented an algorithm called \( TF(\text{twice filter}) \) 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 \( a.\text{par}_b \) or
\( a.\text{ret} \) propagates to \( b.\text{par}_a \), 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
\( G = \{ \text{in}, \text{out}, A, E \} \) in which \( \text{in} \) is the entry point, \( \text{out} \) 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: \( GS = \{ G_1, G_2, \ldots, G_n \} \).

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 \( a_i = \{ \text{ret}, \text{par}_r, \text{par}_l \} \), if
\( \text{par}_r \) 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 \( G_i = \{ A_1, A_2, \ldots, A_n \} \).

According to the definition of TCG, for a software system
\( S \) , if all \( A_i \in G_i \) 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 \( A_i \) 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,
declared as \( p : D \rightarrow D \) , in which \( p \in P \) , binary
relationship a represent that two security domains has
taint information flow between them. Its complement
\( P' \) 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 represents state transition, defined as: \( \text{do} : S \times \text{Call} \rightarrow S \). \( \text{output} \) is the return value of certain system call, defined as: \( \text{out} : S \times \text{Call} \rightarrow O \). \( \text{process} \) means the state produced after an execution of a sequence of system call, defined as: \( \text{process} : S \times A^* \rightarrow S \). Let \( r \) as an empty call, \( o \) is connection operation, the following recursive property is satisfied:

\[
\begin{align*}
\text{process}(s, r) &= s \\
\text{process}(s, o \circ A) &= \text{process}(\text{do}(s, a), A)
\end{align*}
\]

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

\[
\forall n \in \text{Taint} : n \in \text{write}(d_i) \land n \in \text{read}(d_j) \Leftrightarrow d_i \neq d_j
\]

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:

\[
R = \{ N, V, \text{read}, \text{write}, \text{observe} \}, \text{ in which } N \text{ is the set of identity for each memory unit, } V \text{ is the corresponding value set, } \text{read and write are two basic operations, observe is observation.}
\]

**Definition 4:** Function \( \text{origin} : \text{Call}^* \times D \rightarrow P(D) \), gets all security domains in taint call sequence:

\[
\begin{align*}
\text{origin}(r, u) &= u \\
\text{origin}(a \circ A, u) &= \text{origin}(A, u) \cup \text{dom}(a) \\
\text{if } &\exists v \in \text{origin}(A, u) \land \text{dom}(a) \text{ a } u \\
\text{origin}(a \circ A, u) &= \text{origin}(A, u) \text{ else}
\end{align*}
\]

**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: \( s[\approx u]r \), which represents that the value of state \( s \) and \( t \) observed in security domain \( u \) are the same. Formally defined as:

\[
\forall n \in \text{observe}(u) : s(n) = t(n)
\]

**Definition 6:** Function \( \text{filter} : \text{Call}^* \times D \rightarrow \text{Call}^* \) represents the gain of sub-sequence that after filtering all system calls that has taint inference with specific security domain:

\[
\begin{align*}
\text{filter}(r, v) &= r \\
\text{filter}(a \circ A, u) &= a \circ \text{filter}(A, u) \text{ if } \text{dom}(a) \in \text{origin}(A, u) \\
\text{filter}(a \circ A, u) &= \text{filter}(A, u) \text{ else}
\end{align*}
\]

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:

\[
\begin{align*}
\text{out}(\text{process}(s_0, A), a) &= \text{out}(\text{process}(s_0, \text{filter}(A, \text{dom}(a))), a)
\end{align*}
\]

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:

\[
s[\approx u]r \Rightarrow \text{out}(s, a) = \text{out}(t, a)
\]

2). Consistence of state equivalence:

\[
s[\approx u]r \land s[\approx \text{dom}(a)]r \Rightarrow \text{do}(s, a)[\approx u]\text{do}(t, a)
\]

3). Non-interference of taint data propagation:

\[
\forall n \in \text{observe}(u) : s(n) \neq \text{do}(s, a)(n) \Rightarrow \text{dom}(a) \text{ a } u \\
\text{dom}(a) \text{ a } u \Rightarrow \forall n \in \text{observe}(u) : s(n) \neq \text{do}(s, a)(n)
\]

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

\[
\begin{align*}
\text{process}(s_0, A)[\approx \text{dom}(a)] &\Rightarrow (a) \\
\text{process}(s_0, \text{filter}(A, \text{dom}(a))) &\Rightarrow (a)
\end{align*}
\]

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

\[
\text{process}(s_0, A)[\approx u]\text{process}(s_0, \text{filter}(A, u)) (b)
\]

We applied mathematical induction on the length of \( A \) to prove (b). When \( A = r \), (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).

\[
\begin{align*}
s[\approx u]r &\Rightarrow \text{process}(s, A)[\approx u]\text{process}(t, \text{filter}(A, u)) (c) \\
s[\approx u]r &\Rightarrow \text{process}(s, a \circ A)[\approx u]\text{process}(t, \text{filter}(a \circ A, u)) (d)
\end{align*}
\]

We can also applied mathematical induction on the length of \( A \) and reduction to absurdity approach on the assumptions of expression \( \text{dom}(a) \in \text{origin}(i \circ A, u) \) and
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 also will give a more reasonable explanation under structured environment.

**B. Explanation Under Structured Environment**

Let function \( \text{content} : S \times N \to V \) 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:

\[
s \equiv u \iff \forall n \in \text{observe}(a) : \text{content}(s, n) = \text{content}(t, n)
\]

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

\[
\forall a \in \text{Call} : u \in \text{shift}(a, \text{dom}(a)) \iff \text{ring}(\text{dom}(a)) \lor \text{ring}(a) \land (\text{dom}(a) \neq u) \in P
\]

Function \( \text{ring}(u) \) 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:

\[
\forall n \in \text{observe}(\text{dom}(a)) : \text{content}(s, n) = \text{content}(t, n) \Rightarrow
\]

\[
\forall n \in \text{observe}(\text{dom}(a)) :
\]

\[
\text{content}(\text{do}(s, a), n) = \text{content}(\text{do}(t, a), n)
\]

**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:

\[
\forall n \in \text{observe}(u) :
\]

\[
\text{content}(s, n) = \text{content}(t, n) \land \text{ring}(\text{dom}(a)) \Rightarrow \forall n \in \text{observe}(u) : \text{content}(\text{do}(s, a), n) = \text{content}(\text{do}(t, a), n)
\]

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

\[
\forall n \in \text{observe}(u) : \text{content}(\text{do}(s, a), n) \neq \text{content}(s, n) \Rightarrow
\]

\[
n \in \text{write}(\text{dom}(a)) \land u \in \text{shift}(a, \text{dom}(a))
\]

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](image)

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. context of system call database stores all critical system info such as register info, system call info, security domain info. Rule decision engine will decide whether the targeted program is trusted by theorem 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: ZDI1125-21).

REFERENCES