软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(8):2268−2286 [doi: 10.13328/j.cnki.jos.005768]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

软件实时可信度量:一种无干扰行为可信性分析方法*

张 帆[1,2], 徐明迪[3], 赵涵捷[1,4,5], 张 聪[1], 刘小丽[6,7], 胡方宁[2]

[1](武汉轻工大学 数学与计算机学院,湖北 武汉 430023)

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

[3](武汉数字工程研究所,湖北 武汉 430205)

[4](东华大学 电机系,台湾 花莲 08153719)

[5](宜兰大学 资讯工程系,台湾 宜兰 02415271)

[6](暨南大学 信息科学技术学院,广东 广州 510632)

[7](暨南大学 网络空间安全学院,广东 广州 510632)

通讯作者: 徐明迪, E-mail: mingdixu@163.com

摘 要: 可信度量作为可信计算"度量、存储、报告"三大核心功能的基础,到目前为止仍未有有效的数学理论以及运行时(runtime)度量方法.其困难在于 3 点:一是如何建立涵盖不同主流"可信"定义的通用数学模型;二是如何依托数学模型构建运行时可信度量理论;三是如何将上述模型和理论映射到真实信息系统以形成可实践的实时度量方法.提出了一种基于无干扰的软件实时可信度量方法.首先,利用无干扰模型解释了各类主流的可信定义,表明无干扰模型可以作为可信计算通用数学模型的一个选择.其次,基于无干扰模型提出了一种软件实时可信度量理论,其基本思想是将系统调用视作原子动作,将软件真实行为α看做系统调用的序列,并基于α中所有系统调用所属安全域之间的无干扰关系计算软件理论上的预期行为β,得到α和β之后,利用无干扰等式判定两者之间是否存在偏差,从而实现对软件可信性的实时度量.最后,给出了实时可信度量算法,算法的时间复杂性为 $O(1)$.原型实验结果表明了所提出的方法的有效性.

关键词: 可信度量;无干扰;行为可信;可信计算;软件安全

中图法分类号: TP311

中文引用格式: 张帆,徐明迪,赵涵捷,张聪,刘小丽,胡方宁.软件实时可信度量:一种无干扰行为可信性分析方法.软件学报, 2019,30(8):2268−2286. http://www.jos.org.cn/1000-9825/5768.htm

英文引用格式: Zhang F, Xu MD, Chao HC, Zhang C, Liu XL, Hu FN. Real-time trust measurement of software: Behavior trust analysis approach based on noninterference. Ruan Jian Xue Bao/Journal of Software, 2019,30(8):2268−2286 (in Chinese). http://www.jos.org.cn/1000-9825/5768.htm

Real-time Trust Measurement of Software: Behavior Trust Analysis Approach Based on Noninterference

ZHANG Fan[1,2], XU Ming-Di[3], CHAO Han-Chieh[1,4,5], ZHANG Cong[1], LIU Xiao-Li[6,7], HU Fang-Ning[2]

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

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

[3](Wuhan Digital and Engineering Insitute, Wuhan 430205, China)

* 基金项目: 国家自然科学基金(61502438); 湖北省自然科学基金(2015CFA061)

Foundation item: National Natural Science Foundation of China (61502438); Natural Science Foundation of Hubei Province (2015 CFA061)

本文由"面向自主安全可控的可信计算"专题特约编辑张焕国教授推荐.

收稿时间: 2018-06-04; 修改时间: 2018-09-21; 采用时间: 2018-12-13; jos 在线出版时间: 2019-03-28

CNKI 网络优先出版: 2019-03-29 09:47:20, http://kns.cnki.net/kcms/detail/11.2560.TP.20190329.0947.018.html

[4](Department log Electrical Engineering, Dong Hwa University, Hwalian 08153719, China)
[5](Department of Computer Science and Information Engineering, Ilan University, Ilan 02415271, China)
[6](College of Information Science and Technology, Jinan University, Guangzhou 510632, China)
[7](College of Cyber Security, Jinan University, Guangzhou 510632, China)

Abstract: Trust measurement, which is the basis of "measurement, storage, and reporting" of trusted computing, is still lack of mathematical theory and has few applications in a real-time environment thus far. The difficulty lies in three points. One is how to establish a general mathematical model that can cover different mainstream definitions of "trust"; the second is how to build a runtime trust measurement theory based on the established mathematical model; and the third is how to map the above the model and theory to real information systems, and therefore form a practical real-time measurement method. To address the above issues, a runtime software trust measurement approach is proposed. Initially, a noninterference model is leveraged to explain different mainstream definitions of trust, indicating that noninterference model can be an appropriate option of general mathematical model for trusted computing. Next, a noninterference model-based real-time trust measurement theory is presented. In the proposed trust measurement theory, a system call is processed as an atomic action, and the sequence of system calls is constructed as the real behaivior of a process. Note that every system call belongs to a security domain, and different security domains are of noninterference with each other. Therefore, after obtaining a real behavior α, the theoretically expected behavior β can be calculated based on the noninterference relations between security domains to which system calls in α belong. Once obtaining α and β, the trust of a process can be measured by determining whether two behaivors α and β deviates. Finally, a trust measurement algorithm is given. The algorithm can determine whether a process trust or not, i.e., whether the real behavior α and the theoretically expected behavior β deviates, within the time complexity of $O(1)$. The proposed theory is also applied into real information system, and experimental results show that the proposed approach is effective and efficient.
Key words: trust measurement; noninterference; behavior trust; trusted computing; software security

可信计算经过长期发展,取得了令人瞩目的成果[1−24].随着可信计算成为高价值体系结构(如可信云等)的关键安全支撑技术之一,其自身安全性面临着更高的挑战.在这些挑战中,本文关注于如下 3 个问题.

(1) 第 1 个问题是:当前可信计算缺少理论支撑,且理论研究与实践应用部分脱节.

可信计算在诞生之初即面临着"实践超前于理论"的情况[1],经过近 20 年的发展,仍然没有从数学上建立一个较为完善的可信计算模型.理论上,可信计算首先要明确定义"可信".当前主流的定义包括"完整性"[3]、"行为符合预期"[3]和"可靠+安全"[1]等几种不同形式.已有的工作大多从单一的角度研究可信[4−24],而并没有考虑建立一个统一的、涵盖以上 3 种不同定义的模型.另一方面,即便理论上取得了一些成果,这些成果也难以应用到实践当中.以国内广受关注的无干扰模型在可信计算领域的研究为例[22−24],由于几乎没有工作解决无干扰属性自动化验证的难题[25−27],使得大多理论成果并不能实际应用于实时信息系统,如进程信任链传递[22]、可信度量[23]等当中.理论成果和实践应用之间仍然存在着较大差距.

(2) 第 2 个问题是:当前的可信度量属于"被动发现"机制,缺少"主动防御"能力.

根据可信计算标准,可信计算有三大核心功能:度量、存储和报告,三者协同运作确保了第三方能够可靠地判定本机的可信性.其基本过程是:本机在运行的过程中不断度量自身的可信性,并将度量结果存储到专用安全芯片 TPM(trusted platform module)和内存 log 当中.当其他机器想与本机通信时,由本机将存储的度量结果报告给对方,并由对方来判定本机的可信性.上述过程是一种"事后被动发现"机制,换句话说,这种机制只能确保其他机器可靠地判定本机是否被攻击而处于不可信状态,但却无法及时检测到机器被攻击的情况.事实上,"事前防御、事中审计、事后发现"应当作为环环相扣的整体.然而,现有的可信度量侧重于"事后被动发现"、而缺少"事前主动防御"的研究工作.

(3) 第 3 个问题是:可信计算如何在层出不穷的攻击手段下可靠地实现可信度量.

可信计算"度量、存储和报告"三大核心功能中,度量居于基础地位,存储和报告的内容均来自于度量的结果.随着新型攻击方法(如 ROP[28]等)层出不穷、传统攻击不断应用包括多态/混淆/变形/编码等在内的多种对抗手段以及未公开攻击机理的威胁等,使得度量信息系统的可信性变得日益复杂和困难.特别地,运行时(runtime)度量才能真正反映系统行为的可信性[1],但这对度量的实时性提出了很高的要求.

针对上述问题,本文提出了一种基于无干扰理论的软件实时可信度量与防护方法,主要贡献如下:

(1) 针对问题(1)和问题(2),首先,利用无干扰模型解释了主流可信定义——即“完整性”“行为符合预期”和“可靠+安全”——的数学涵义(参见第 2.2 节的定义 9);其次,通过将系统调用看做原子动作,提出了基于无干扰的实时可信度量与防护方法,其基本思想是:根据不同系统调用所属安全域之间的无干扰关系,建立软件预期行为 EB(expected behavior),并利用无干扰模型判定软件真实行为与预期行为之间的一致性,从而实现对软件的实时可信度量与主动防护(参见第 3 节);最后,给出了实时可信性判定算法(参见第 3.5 节的算法 1),算法的时间复杂性为 $O(1)$.

(2) 针对问题 3,本文基于如下观察:绝大多数攻击都需要前后相继的两个步骤,即首先定位可利用的漏洞,然后通过漏洞执行攻击代码(shellcode).打断以上环环相扣的任何一步,就可以有效地防御攻击.本文从第 2 步入手,注意到攻击代码几乎都需要系统调用的支持,因而不论是何种类型的攻击,我们只需要利用系统调用序列构建代码的真实行为,利用系统调用安全域之间的无干扰关系刻画软件的预期行为 EB,即可以通过判定当前代码的真实行为与预期行为 EB 之间是否存在偏差来度量是否发生了攻击.实验结果表明:方法是有效的,其不仅能够实时度量到新型攻击如 ROP 等,而且可以有效地发现多态、混淆、变形和编码等多种传统对抗方法所生成的攻击代码,理论上也可以防御未知机理攻击(只要其利用系统调用实现攻击).

(3) 所提出的方法无需任何学习训练过程,无需任何先验知识.使用者亦不必关心底层理论机制,只要给出安全策略即可,具体来说,只需要根据实际安全需求,依照预期行为规范 EBS(expected behavior specification)语法编写 EBS(参见第 3.3 节和附录),系统即可实现实时可信度量与防护.

1 相关研究工作

1.1 可信度量相关研究工作

国际上第一个基于 TCG(trusted computing group)标准的完整性度量架构是 IMA[4].IMA 是一种“加载时度量”,这种度量方式只能说明软件没有被篡改(哈希不变),并不能说明软件运行时的动态行为是否可信.随后,卡内基梅隆大学的 Shi 等人提出了为分布式系统建立可信环境的 BIND 框架[5].BIND 把完整性度量对象的范围缩小到关键代码段,并对关键代码段的每一组数据生成一个认证器,实现了细粒度的动态度量验证.但 BIND 需要事先确定关键代码段,并建立关键代码段和输入数据/输出数据之间的绑定关系.这种绑定与本文是有差别的:本文对进程所绑定的是基于系统调用描述的预期行为 EB,同时,绑定过程也无需确定关键代码段.在 IMA 的基础上,Jaeger 等人通过引入 CW-Lite 模型设计了 PRIMA 架构[6],实现了运行时度量.但 PRIMA 所基于的 CW-Lite 模型缺少形式化验证能力,因而无法形式化地对系统设计进行可信验证.从内核完整的角度,Loscocco 等人[7]提出了 Linux 内核完整性监视器 LKIM,用以监控 Linux 内核运行时关键数据结构的动态完整性;Carbone 等人[8]实现了 KOP 系统,用以监控内核指针对象的动态完整性.LKIM 和 KOP 的贡献在于它们对于可信计算平台的关键基础软件(操作系统)的动态完整性度量具有重要的意义,不足之处在于它们无法扩展到一般应用程序.针对云计算环境下的动态完整性度量问题,Schiffman 等人[9]实现了 VMV(virtual machine verifier).VMV 能够对虚拟机及其内部的应用程序进行动态完整性度量,以确保云端计算的可信性.不过,VMV 构建的基础仍然是 PRIMA[6],从而缺少形式化验证能力.从软件运行时的局部结构化约束完整性和全局不变式完整性入手,Kil 等人[10]提出了一种基于完整性动态属性的远程证明系统 ReDAS.ReDAS 需要一个学习训练过程,这使得它只能局限于度量经过训练的软件.Xu 等人[11]通过将系统的可信性归结为系统状态变化的可信性,提出了一个远程证明框架 DR@FT.但 DR@FT 只能适用于基于 Clark-Wilson 或 CW-Lite 模型的完整性度量框架,因而也无法验证自身设计架构的可信性.John 等人[12]发现:在 Dell Latitude E6400 Laptop 上,可信度量根核 CRTM(core root of trust measurement)的实现没有满足 TPM PC 规范和 NIST 800-155 指导规范,这使得攻击者可以回放伪造的度量结果给 TPM,从而导致证明方错误地认为本机被攻击过的 BIOS 仍然处于“干净可信”的状态.该方法侧重于解决 CRTM 和信任链传递的安全性问题,而并非针对软件行为的实时度量.针对动态度量时动态可信度量根 DRTM(dynamic root of trust for measurement)的存在性(activated/enabled)攻击问题,Zhang 等人[13]提出了一种强

制性、最小化、用户参与的解决方案.但该方案只确保 DRTM 确实存在(activated),同样不涉及软件行为的实时度量.Abera 等人[14]设计并实现了控制流证明 C-FLAT(control flow attestation)架构,通过软件的控制流路径,而不是源码,来证明软件的可信性,尝试为物联网设备上的远程证明提供一种解决方案.针对越来越深入的硬件可信度量和防护方法,Zhang 等人[15]提出了一种化整为零的攻击技术,以绕过现有的硬件检查措施,其基本思想是:将整体的、恶意的硬件功能分解成零散的、正常的子模块,在子模块通过可信验证之后再将其组合成整体,以实现恶意硬件功能.由于硬件层位于系统调用层之下,因而本文无法对这种攻击进行检测和防御.Matetic 等人[16]提出了一种称为 ROTE 的分布式回滚保护机制,攻击者若要利用恶意回归攻击破坏 Intel SGX 等安全机制下的信息完整性,就必须将 ROTE 分布式平台中的所有参与者同时回滚到初始状态,这大大增加了攻击的难度.

国内对于可信度量相关问题也展开了深入的研究.为了解决可信平台模块 TPM 在云计算、网络功能虚拟化等场景下低成本可信扩展和前向安全问题,余发江等人[17]提出了一种动态信任扩展方法,有效保证了虚拟可信平台模块 vTPM 的可信性,为虚拟环境下的度量、存储、报告、密码、证书等功能构建了可靠的硬件基础.邓良等人[18,19]用 ring 0 层的内可信基(inner TCB)替代传统的虚拟机监视器,实现对内核代码和控制流的完整性度量以及对硬件操作和软件行为的验证,有效保证了上层应用的安全,且开销很小.林杰等人[20]针对虚拟机软件的完整性问题,提出了一种利用关键数据结构内存映射和地址动态转换技术,实现对上层应用运行时完整性度量的方法.与本文工作相比,上述完整性度量方法[18−20]的共有问题在于其无法回答软件(即便代码和数据被度量是完整的)在运行过程中是否会出现不可信的行为.陈志锋等人[21]基于内存取证技术提出了一种内核实时完整性度量方法.与本文相比,本文针对上层软件,陈志锋等人[21]针对的内核,两者的研究对象并不一致.另一方面,若将文献[21]的技术应用于上层应用,其需要“完整性基准库”,换句话说,文献[21]需要一个学习训练的过程,对于未经学习的软件可能无法准确实施度量.相反,本文的方法无需学习训练,只要编写好软件预期行为规范,即可以度量任何软件的实时行为.理论上,甚至还可以发现未知攻击.张兴等人[22,23]利用无干扰模型,对信任链传递[22]以及进程[23]的可信性度量问题进行了研究.他们工作的重要性在于首次将无干扰模型应用于可信计算领域,但不足之处是没有给出可实践的方法,而如何将无干扰模型应用于实践,一直是无干扰研究的一个关键所在.

1.2 无干扰及其应用相关研究工作

Goguen 等人[29]以状态机的形式首先引入了传递无干扰的概念,随后,Rushby 建立了标准的非传递无干扰模型[25].Rushby 的研究表明,传递无干扰只是非传递无干扰的一个特例.因此,如果没有特殊说明,“无干扰”一般都特指“非传递无干扰”.在 Rushby 的模型中,仅仅考虑了观察者是否能够区分状态变迁所导致的状态差异,而并不关心观察者是否能够同时观察到所执行的不同动作的差异.Meyden 注意到了这个问题,由此定义了安全约束更强的 TA-Security 和 TO-Security[30].随后,Eggert 等人研究了经典的传递无干扰和非传递无干扰以及 TA-Security 和 TO-Security 这 4 种无干扰属性验证的时间和空间复杂度[26].需要指出的是,Eggert 等人仅仅给出了时空复杂度推演结果,但没有给出具体的属性验证算法.事实上,到目前为止,状态机形式的无干扰模型属性验证仍然是一个难题[25−27],当前最好的工作是 Hadj-Alouane 等人给出的验证算法[27],其不仅支持任意多个安全域,而且支持无干扰属性的可靠性和完备性条件,然而其时间复杂性却高达双指数 $O(2^{|X||2^D|})$,无法实用.

除了状态机以外,进程代数的互模拟特性使得它成为描述无干扰的另一个有力工具[31,32].特别地,利用进程代数的验证工具,可以很容易地对基于进程代数描述的无干扰属性进行验证.然而需要指出的是,3 种不同的无干扰模型,即 Rushby 建立的无干扰模型[25]、Meyden 和 Eggert 等人建立的无干扰模型[26,30]以及 Ryan 和 Focardi 等人建立的无干扰模型[31,32],三者之间两两语义不等价.这意味着基于进程代数[31,32]的无干扰属性验证算法和工具对于状态机形式[25,26,30]的无干扰属性验证并无任何帮助.

在国内,无干扰模型在可信计算领域受到了广泛关注,应用在包括可信测评[33]、信任链可信[22,34]、进程可信[23]和终端隔离[24]等在内的众多领域.但是如前所述,由于没有有效的无干扰属性验证算法,上述工作中采用状态机作为工具的工作几乎都只能停留在理论分析层面而难以实用[22−24].为了解决可实践的验证问题,当前国际上的主流方法[35,36]是基于 Rushby 的展开定理(unwinding theorem)[25],利用公理语义,结合定理证明器进行属

性验证.但定理证明器的证明时间相对于实时要求而言并不可控,并且有可能无法给出证明,因而这类方法也无法应用在实时可信防护领域.

1.3 基于系统调用的安全防护

系统调用在检测异常行为方面具有较好的特异性,早期的系统调用主要应用在入侵检测领域,研究人员通过考察系统调用序列切片(不考虑参数)或者系统调用参数集合关系(不考虑系统调用自身)与预期标准的差异来判定一个进程是否被入侵,代表性方法如文献[37]等.这类方法在早期开启了系统调用在入侵检测领域的新应用,但这些方法大多只考虑了系统调用的语法特征,而并未考虑其语义规律,因而有较大的局限性.之后,研究人员尝试综合考虑系统调用或者 API 的语法和语义,以图、状态机、马尔科夫链、机器学习等作为工具对进程行为进行研究,取得了较为丰硕的成果[38−43].现阶段,采用系统调用方法的研究困难和热点在于,底层系统调用与上层应用之间存在着语义断层(semantic gap),仅仅从系统调用本身无法与上层应用行为之间建立映射,从而必须具体问题具体分析,尚没有一种较为通用的方法或者框架解决这个问题[38,41].

系统调用与无干扰模型的结合很少见,主流工作中相近的是 Calvin 等人[44]基于无干扰实时检测和防御 TOCTTOU(time-of-check-to-time-of-use)竞争条件攻击,其核心思想在于构建语义等价、但执行顺序不同的系统调用序列以消除竞争条件.多数情况下,这种构建是困难的,文献[44]也未提及构建方法(算法),因此,我们认为更多的是理论意义[44].不同于文献[44],由于前期我们找到了无干扰属性基于状态等价的充要条件[45],因而为系统调用与无干扰模型相结合、对软件行为进行实时无干扰属性验证探索了一条可能的道路.

1.4 携带证明的代码

携带证明的代码 PCC(proof carrying code)亦可以对代码可信性进行验证.PCC 框架[46]由代码提供者 CP (code producer)和代码使用者 CC(code consumer)两部分构成.其中,CP 必须为其提供的代码提供一个安全证明(safety proof),该安全证明实质上是 CP 对 CC 所规定的安全策略(safety policy)的形式化描述.CC 在运行 CP 所提供的代码之前,会先用证明验证器对 CP 提供的安全证明进行验证:如果验证通过,则说明安全策略得到了遵守,从而 CP 所提供的代码是可信的,可以安全地在 CC 上执行;否则,代码不可信,CC 拒绝代码的执行.

PCC 为代码的可信执行建立坚实理论基础,但应用到实时可信度量领域还存在如下难点需要解决.

(1) PCC 破坏了已部署代码的兼容性.PCC 框架中,CP 在根据安全策略编写了形式化规范之后,需要将原应用和形式化规范一起重新打包编译生成新的、包含形式化规范的二进制代码[47],我们称其为 PCC 风格的二进制代码.显然,若采用 PCC 架构进行软件实时可信度量,则度量平台上所有的应用(即代码)都必须重新编译以形成 PCC 风格的二进制代码,否则,应用无法通过平台验证,进而无法运行.这意味着如果采用 PCC 架构,则必然会破坏已部署应用的兼容性.

(2) PCC 安全策略编写代价较大.由于不同应用所使用的编程语言以及语言的安全特性等存在差异,使得难以构造一个统一、标准的形式化规范.换句话说,每一个应用的 CP 都需要单独提供其应用所对应的形式化规范.当安全策略发生变化,例如当发生网络安全事件进行应急响应时,每个应用的形式化规范必须要对应更新,并重新编译生成新的 PCC 代码.这大大增加了安全策略规范编写、维护和更新的代价.

与 PCC 相反,本文的方法不存在上述问题:(1) 从兼容性的角度,由于本文的方法不需要代码自身携带安全证明,因而无需对应用做任何改动,从而对已部署应用的兼容性没有任何影响;(2) 从安全策略规范角度,本文的方法以所有应用所共用的系统调用为对象来构造安全策略.通用情况下,使用者可以基于系统调用建立适用于所有应用的统一、标准的安全策略规范;在此基础上,对于特定应用的特定需求,亦可以建立与其一一对应的特殊的安全策略规范.当安全策略发生变化时(如前述网络安全事件应急响应时),只需要更新适用于所有应用的统一、标准的安全策略规范即可.与 PCC 相比,本文的方法大大减小了安全策略规范编写、维护和更新的代价.然而,PCC 方法也有其优势,例如 PCC 的度量粒度可以更细,且已发展出坚实的理论基础和较多的工具,如何尝试解决 PCC 的前述两大难题,以结合 PCC 和本文的方法进一步提升本文方法的理论基础和可实践性,是一个值

得研究的工作,但这超出了本文的研究范围.

2 理论基础

2.1 无干扰模型定义

无干扰的思想最早由Goguen和Meseguer[29]提出,其基本思想是:一组用户G_A,使用一组命令集合C操作之后,如果对另一组用户G_B所能观察到的结果没有影响,则称用户组G_A对用户组G_B是无干扰的.既然G_B无法感知G_A的操作,那么也就无法从G_A的操作中逆推出任何信息,从而也就防止了G_A和G_B之间的隐通道信息流传输.文献[29]解决了困扰BLP的隐通道问题,但只能处理传递信息流,对于非传递信息流是无法处理的.例如,不失一般性,假设某个涉密系统中划分有文件域u_f、加密机域u_e和网络域u_n这3个安全域.如果我们用$\rightsquigarrow$和$\not\rightsquigarrow$分别表示允许和不允许信息在安全域之间的流动,则此涉密系统中机密文件的传输方式可以表示为$(u_f \not\rightsquigarrow u_n)\wedge(u_f \rightsquigarrow u_e \rightsquigarrow u_n)$.其含义是:机密信息不能直接传输到网络$u_f \not\rightsquigarrow u_n$,而必须首先流经加密机加密$u_f \rightsquigarrow u_e$、再由加密机将加密后的密文发送到网络进行传输$u_e \rightsquigarrow u_n$.换句话说,$u_f$只能通过加密机中介$u_e$间接地将信息传递到$u_n$.直觉地,对于这种信息流非直接(间接)流动的情形,我们称其为非传递信息流,文献[29]无法对其进行处理

随后,众多研究人员对非传递信息流情况下的无干扰模型进行了深入研究,但是直到Rushby才首次提出完全正确的非传递无干扰模型[25].由于传递无干扰只是非传递无干扰的一个特例[25],因而本文中我们以一般性的非传递无干扰模型为基础,研究如何构建可实践的软件实时度量和防护方法.以下若没有特殊说明,则文中的无干扰模型均特指非传递无干扰模型,且文中所有的定义均特指在非传递安全策略前提下.

定义1. 无干扰模型定义.一个系统M可以用一个状态机表示,其包含如下元素.

(1) 状态机S,其中包含一个唯一的初始状态s_0.

(2) 一个原子动作集A,其中包含了所有的原子动作.

(3) 一个行为集B,其中包含了所有由原子动作的连接所构成的行为.如果用$\circ$表示原子动作的连接,则一个行为的示例是$\alpha=a_0\circ a_1\circ\ldots\circ a_n$,其中,$a_{i(0\leqslant i\leqslant n)}\in A$.

(4) 一个输出集O,其中包含了所有利用原子动作所观察到的输出结果.

(5) 一个安全域集D,其中包含了系统中所有的安全域.

(6) 干扰关系$\rightsquigarrow$和无干扰关系$\not\rightsquigarrow$,分别表示信息授权/禁止在两个安全域之间流动,两者互为补集.

(7) 动作-安全域映射函数$dom:A\rightarrow D$.返回每一个原子动作$a\in A$所属的安全域$dom(a)$.

(8) 单步状态变迁函数:$step:S\times A\rightarrow S$,描述了系统$M$的单步状态变迁.

(9) 多步状态变迁函数:$multisteps:S\times B\rightarrow S$.如果用$\Lambda$表示空动作,则$multisteps$可以右递归表示为

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

其描述了系统M从状态$s\in S$执行行为(即多个原子动作序列)$\alpha=a_0\circ\ldots\circ a_m\in\beta$之后所到达的新状态.

(10) 行为结果函数(behavior consequence):$behcon:S\times A\rightarrow O$,给出了系统$M$在某个状态$s\in S$,外界利用动作$a\in A$所能观察到的结果$o\in O$.

约定使用…,s,t,…表示系统状态,约定使用α,β,…表示行为(原子动作序列),使用u,v,…表示安全域.

定义2. 结构化机器M.结构化机器M包含如下部分.

(1) 存储单元集N.机器的每一个存储单元都有一个独一无二的名字,所有这些名字的集合构成存储单元集N,又叫名字集N.存储单元包含寄存器单元、内存单元、外存单元、缓存等.

(2) 值集V.当状态一定的时候,每一个存储单元都有一个具体的值v.所有存储单元值的集合构成值集V.

(3) 内容函数$contents:S\times N\rightarrow V$.内容函数返回在状态$s\in S$,名字$n\in N$的值$v\in V$.

(4) 观察函数$observe:D\rightarrow\mathcal{P}(N)$.观察函数给出安全域$u\in D$所能观察的名字集合.这里,$\mathcal{P}$是幂集计算.

(5) 修改函数$alter:D\rightarrow\mathcal{P}(N)$.修改函数给出安全域$u\in D$所能修改的名字集合.这里,$\mathcal{P}$是幂集计算.

定义 1 中的原子动作粒度可大可小,可以是输入(input)、函数调用(API 或者 syscall)、命令(command)或者机器指令(instruction)等[25].当执行原子动作之后,机器从一个状态变迁到里另外一个状态.根据结构化机器假设,机器 M 的状态由 M 中的所有存储单元取值构成.存储单元包括寄存器(registers)、内存单元(memory)或者磁盘扇区(disk)等.上述存储单元有一些是不可观察的(如内部存储单元),有一些是暴露给观察者的;有一些是只读的,有一些是可读可写的;有一些是遗失性的,有一些是非遗失性的,等等,所有这些存储单元的取值情况决定了机器 M 的当前状态.在实践中,用户可以根据实际需要,基于定义 1、定义 2,将实际系统与机器 M 一一对应起来.

定义 3. $\overset{u}{\sim}$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: $dom(a) \rightsquigarrow u \wedge s \overset{dom(a)}{\sim} t \wedge s\overset{u}{\sim}t \supset step(s,a)\overset{u}{\sim}step(t,a)$.

本文全文遵从文献[25],约定使用$\supset$表示蕴含关系,使用$\rightarrow$定义函数.

定义 4. 引用监视器假设 RMA(reference monitor assumption).RMA 由如下 3 条假设构成.

(1) 假设 1: $s\overset{u}{\sim}t$ **iff** $\forall n \in observe(u).contents(s,n)=contents(t,n)$.其含义是: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.假设 1 保证了定义 3 中的输出一致性.

(2) 假设 2: $s\overset{u}{\sim}t \wedge [contents(step(s,a),n)\neq contents(s,n) \vee contents(step(t,a),n)\neq contents(t,n)] \supset contents(step(s,a),n)=contents(step(t,a),n)$.其含义是:如果两个状态等价 $s\overset{u}{\sim}t$,则当它们使用相同的动作修改某个存储单元时,必须保证该存储单元被修改为同样的值.

(3) 假设 3: $contents(step(s,a),n)\neq contents(s,n) \supset n \in alter(dom(a))$.其含义是:若动作修改了某 a 个存储单元的值,则安全域 $dom(a)$必须拥有对该存储单元修改授权.

定义 5. 干扰源集 $interfsrcs: B\times D\rightarrow \mathcal{P}(D)$.干扰源集的递归定义为 $interfsrcs(\Lambda,u)=\{u\}$,且

$$interfsrcs(a\circ\alpha,u)=\begin{cases}\{dom(a)\}\cup interfsrcs(\alpha,u), & \text{if } \exists v.v\in interfsrcs(a,u)\supset dom(a)\rightsquigarrow v\\ interfsrcs(\alpha,u), & \text{otherwise}\end{cases}.$$

干扰源集的含义是:抽取所有对 u 有(直接或间接)干扰关系的安全域形成(干扰源)集合.

定义 6. 弱预期函数 $wexpected: B\times D\rightarrow B$.其递归定义为 $wexpected(\Lambda,u)=\Lambda$,且

$$wexpected(a\circ\alpha,u)=\begin{cases}a\circ wexpected(\alpha,u), & \text{if } dom(a)\in interfsrcs(a\circ\alpha,u)\\ \Lambda\circ wexpected(\alpha,w), & \text{otherwise}\end{cases}.$$

弱预期函数的含义是:将所有对 u 有(直接或间接)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 7. 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 8. 局部无干扰属性: $dom(a) \nrightarrow u \supset s\overset{u}{\sim}step(s,a)$.

2.2 基于无干扰的可信性判定

根据文献[25],一个信息系统满足无干扰,当且仅当如下判定等式(1)成立.

定义 9. 无干扰判定等式:

$$\forall\alpha\forall a.behcon(exec(s_0,\alpha),a)=behcon(exec(s_0,wexpected(\alpha,dom(a))),a) \quad (1)$$

关于什么是“可信”,国际上可信计算组织 TCG 将其定义为:(1) 完整性;或(2) 行为符合预期;国内有专家指出:(3) 可信≈可靠+安全.形式上,等式(1)对这些“可信”的定义均可以进行解释.

(1) “完整性”的定义

等式(1)的左边表示行为α的实际执行结果,右边表示在完整性策略控制下,行为 $wexpected(\alpha,dom(a))$的理论执行结果.如果等式成立,则表明真实执行结果与完整性策略控制下的执行结果一致,换句话说,当前的执行是符合完整性策略的.因而完整性没有遭到破坏.

事实上,利用无干扰研究和解释完整性是完全可行的.我们曾询问文献[31]的作者 Ryan 是否可以利用无干扰研究完整性,Ryan 回复指出:“利用无干扰研究完整性是完全可行的.唯一的细微差别在于,完整性无干扰模型可能需要一个更弱一点的形式化:对于安全性/机密性必须确保低(low)安全等级观察者不能够推断出任何高(high)安全等级的信息;而对于完整性很可能不需要担心高(high)完整性等级能够推断出低(low)完整性等级的信息,只要低完整性等级信息不能干扰高完整性等级信息即可”.

(2) “行为符合预期”的定义

类似的,等式(1)的左边表示行为α的实际执行结果,右边表示在安全策略控制下,行为 $wexpected(\alpha,dom(a))$ 的理论执行结果(预期执行结果).如果等式成立,则表示真实执行结果与预期执行结果一致.换句话说,机器 M 是以预期的方式朝着预期的目标在执行,因而是可信的.

(3) “安全性(机密性)”的定义

这是无干扰模型提出的本义,因而自然成立.

综上所述,对于“可信”的几种主流定义,等式(1)均可以给予解释.我们认为,无干扰模型是研究可信的一个非常合适的数学工具.

接下来的困难在于如何将无干扰模型应用于实时动态系统.Rushby 曾明确指出了无干扰模型所面临的挑战[25]:一是寻求有效(efficient and effective)的无干扰属性自动化验证算法,二是探索无干扰模型的实际应用.本质上,这两个问题可以归结为第 1 个问题.因此,虽然无干扰模型非常适用于可信研究,但是要真正应用于实时动态度量和防护,就必须解决无干扰属性的自动化验证问题.幸运的是,我们在文献[45]中找到了一种与等式(1)等价、以状态递归形式表达、可靠且完备的无干扰表达形式(参见如下定理 2)解决了这个问题.详细说明如下.

定理 1(无干扰属性验证展开定理[25]). 如果机器 M 满足如下属性,则机器 M 是满足无干扰的,或者说可信的.

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

(3) 局部无干扰属性: $dom(a) \not\rightsquigarrow u \supset s \overset{u}{\sim} step(s,a)$,

证明:参见文献[25]定理 7. □

Rushby 将等式(1)看作一个状态机,利用归纳法证明了定理 1 所示的展开定理.在文献[45]中,我们将等式(1)的左右两边看作两个状态机,分别称为等价自动机 EMA(equivalent machine’s automaton)和弱预期等价自动机 WEMA(weakly equivalent machine’s automaton).不失一般性,假设左右两个状态机的状态集分别是 S 和 T,则等式(1)中的 EMA 和 WEMA 将从相同的初始状态 $s_0=t_0(s_0\in S\wedge t_0\in T)$出发,分别执行行为$\alpha$以及安全策略控制下的行为$\beta=wexpected(\alpha,dom(a))$.利用 EMA 和 WEMA,可以证明定理 1 的 3 条属性的数学本质是:EMA 和 WEMA 在同步执行的过程中,总是保持状态等价的[45].该数学本质可以形式地表达为定理 2.

定理 2(无干扰(或者说可信)的状态递归定义[45]). 机器 M 是满足无干扰的,或者说可信的,当且仅当:

$$\forall i_{(0\leqslant i\leqslant N)}.s_i \overset{IS_{\gamma_i}}{\approx} t_i \supset s_{i+1} \overset{IS_{\gamma_{i+1}}}{\approx} t_{i+1} \tag{2}$$

证明:参见文献[45]的定理 3. □

定理 2 中的符号含义如下.

(1) N:对于任意行为γ,N 是γ中原子动作的个数.例如,不失一般性,设$\alpha=a_0\circ a_1\circ\ldots\circ a_m$,则$\gamma$中共有 m 个原子动作,故 $N=m$.

(2) $\gamma_i(0\leqslant i\leqslant N)$:对于任意行为$\gamma$,随着机器 M 对γ中原子动作的不断执行,用γ_i 表示剩下的行为,称为“执行子行为串”(参见文献[45]的定义 16),并令$\gamma_0=\gamma,\gamma_N=\Lambda$.例如,不失一般性,设$\alpha=a_0\circ a_1\circ\ldots\circ a_m$,则有:初始时,$\gamma_0=\gamma=a_0\circ a_1\circ\ldots\circ a_m$;当机器 M 执行原子动作 a_1 后,有$\gamma_1=a_2\circ\ldots\circ a_m$;...;当机器 M 执行 a_{m-1} 后,有$\gamma_{m-1}=a_m$;当机器 M 执行 a_m 后,有$\gamma_m=\gamma_N=\Lambda$.

(3) IS_{γ_i}:如果用 w 表示等式(1)中的 $dom(a)$,即 $w=dom(a)$,则为了书写简单起见,文献[45]中约定 $IS_{\gamma_i}=interfsrcs(\gamma_i,w)$,即 IS_{γ_i} 的含义是:所有对 $w=dom(a)$有直接或者间接干扰的安全域构成的集合.

(4) s_i 和 s_{i+1}:s_i 表示 EMA 的当前状态,s_{i+1} 表示 EMA 从当前状态 s_i 执行 $\gamma_i=a_{i+1}\circ\ldots\circ a_m$ 中的第 1 个原子动作 a_{i+1} 之后到达的新状态,$s_{i+1}=step(s_i,a_{i+1})$.

(5) t_i 和 t_{i+1}:t_i 表示 WEMA 的当前状态,t_{i+1} 表示 WEMA 从当前状态 t_i 执行 $\beta_i=wexpected(\gamma_i,w)$中的第 1 个原子动作之后到达的新状态 t_{i+1}.由公式(4)中的$\gamma_i=a_{i+1}\circ\ldots\circ a_m$,根据定义 6,当 $dom(a_{i+1})\in interfsrcs(\gamma_i,w)$,即 $dom(a_{i+1})$对 $w=dom(a)$有直接或者间接干扰时,$\beta_i=wexpected(\gamma_i,w)$的第 1 个原子动作为 a_{i+1},此时 $t_{i+1}=step(t_i,a_{i+1})$;否则,若 $dom(a_{i+1})\notin interfsrcs(\gamma_i,w)$,即 $dom(a_{i+1})$对 $w=dom(a)$没有任何干扰时,$\beta_i=wexpected(\gamma_i,w)$的第 1 个原子动作为$\Lambda$,此时 $t_{i+1}=step(t_i,\Lambda)=t_i$.

综上,定理 2(公式(2))的含义是:机器 M 是无干扰的,或者说可信的,当且仅当 EMA 和 WEMA 两个状态机在同步运行的过程中,总是在对 $w=dom(a)$(参见等式(1))有直接或者间接干扰的安全域上保持状态等价的.

定理 2 是无干扰成立的可靠和完备性条件.限于篇幅,其证明思想简单说明如下.

(1) 可靠性证明

递归展开公式(2),有:$s_0 \overset{IS_{\gamma_0}}{\approx} t_0 \supset \ldots \supset s_n \overset{IS_{\gamma_n}}{\approx} t_n \supset s_{n+1} \overset{IS_{\gamma_{n+1}}}{\approx} t_{n+1}$.既然 $s_{n+1} \overset{IS_{\gamma_{n+1}}}{\approx} t_{n+1}$ 成立,由 IS(文献[45]的定义 14)和执行子行为串定义(文献[45]的定义 16)立即可得:$s_{n+1} \overset{w}{\sim} t_{n+1}$.对 $s_{n+1} \overset{w}{\sim} t_{n+1}$ 利用输出一致性有:$behcon(s_{n+1},a)=behcon(t_{n+1},a)(dom(a)=w)$.再根据 EMA 和 WEMA 的定义代入 s_{n+1} 和 t_{n+1},有公式(1)成立,故 M 满足无干扰.可靠性得证.

(2) 完备性证明

完备性证明采用反证法.根据引用监视器假设,机器 M 的状态具体表现为其名字集的取值.因此,EMA 和 WEMA 状态等价,本质上等价于 EMA 和 WEMA 对应名字集的取值相同.再注意到某个名字的值发生变化时,是由该名字当前本身的值加上当前对其进行干扰的名字的值共同决定的.综合以上两点,要保证 EMA 和 WEMA 在 w 上的状态等价关系,就必须要保证 EMA 和 WEMA 在同步运行的过程中,在所有那些对 w 有直接或者间接干扰的安全域上总是保持状态等价的,亦即当 EMA 和 WEMA 到达任意状态对(s_i,t_i)时,必须在 IS_{γ_i} 上是保持状态等价的,此即公式(2).否则,就一定可以构造一个恶意子行为 γ_i',使得当 M 从状态 s_i 出发执行 γ_i' 后,EMA 和 WEMA 最终在 w 上不等价,从而导致公式(1)不成立,M 不满足无干扰属性,矛盾!完备性得证. □

更详细的证明请参见[45]的定理 3.

接下来以一个例子说明定理 2 的含义及应用.假设某个涉密系统中划分有输入域 u_i(input)、过滤域 u_f(filter)、加密机域 u_e(encryption)和网络域 u_n(network)共 4 个安全域.系统安全策略是:从输入域 u_i 接收所有的输入,经过滤域 u_f 过滤识别出敏感信息,再经加密域 u_e 的加密机加密,最后将加密后的密文经网络域 u_n 传出,输入域 u_i 和过滤域 u_f 禁止直接向网络传输信息.安全策略可以表达为:$u_i \rightsquigarrow u_f \rightsquigarrow u_e \rightsquigarrow u_n \wedge u_i \not\rightsquigarrow u_n \wedge u_f \not\rightsquigarrow u_n$.

再令安全域 u_i,u_f,u_e 和 u_n 中发出的动作分别用 a_i,a_f,a_e 和 a_n 表示.不失一般性,假设有如下动作序列(即行为):$\alpha=a_{i1}\circ a_{f1}\circ a_{i2}\circ a_e\circ a_{f2}$,则根据安全策略计算 WEMA 中的对应行为$\beta=wexpected(\alpha,u_n)$,由定义 6 可得:$\beta=a_{i1}\circ a_{f1}\circ\Lambda\circ a_e\circ\Lambda$.根据定理 2,行为$\alpha$是否可信的自动化验证过程如图 1 所示.图 1 中的圆圈表示状态,圆圈之间的箭头表示引发状态变迁的动作,圆圈之间的虚线表示 EMA 和 WEMA 同步执行的过程.

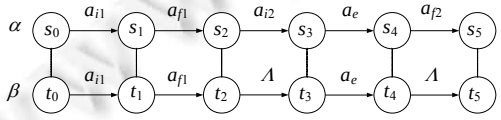


Fig.1 Trust verification/measurement of software based on noninterference (Theorem 2)

图 1 基于无干扰理论的(定理 2)的软件可信验证/度量

对$\alpha=a_{i1}\circ a_{f1}\circ a_{i2}\circ a_e\circ a_{f2}$的可信性验证过程如下.

(1) 首先,EMA 和 WEMA 两者从相同的初始状态$s_0=t_0$出发同步运行.

根据定理 2 的符号解释:

- 对 EMA:此时,$\alpha_0=\alpha=a_{i1}\circ a_{f1}\circ a_{i2}\circ a_e\circ a_{f2}$.EMA 执行首个动作$a_{i1}$之后,从状态$s_0$变迁到状态$s_1$,并将继续执行子行为$\alpha_1=a_{f1}\circ a_{i2}\circ a_e\circ a_{f2}$.
- 对 WEMA:此时,$\beta_0=\beta=a_{i1}\circ a_{f1}\circ\Lambda\circ a_e\circ\Lambda$.WEMA 执行首个动作$a_{i1}$之后,从状态$t_0=s_0$变迁到状态$t_1$,并将继续执行子行为$\beta_1=a_{f1}\circ\Lambda\circ a_e\circ\Lambda$.

计算$IS_{\alpha_0}=interfsrc(\alpha_0,u_n)=\{u_n,u_e,u_f,u_i\}$,$IS_{\alpha_1}=interfsrc(\alpha_1,u_n)=\{u_n,u_e,u_f\}$.将$(s_0,IS_{\alpha_0},t_0)$和$(s_1,IS_{\alpha_1},t_1)$分别代入公式(2)的左边和右边,如果公式(2)的前件和后件均成立(事实上,只需要代入(s_1,IS_{α_1},t_1)验证后件即可,因为$s_0=t_0$故前件总是成立),则转到步骤(2)继续递归验证;否则,如果公式(2)不成立,则说明行为不可信,报警.

(2) 其次,继续根据公式(2)递归验证行为是否可信.

- 对 EMA:此时,$\alpha_1=a_{f1}\circ a_{i2}\circ a_e\circ a_{f2}$.EMA 执行动作$a_{f1}$之后,从状态$s_1$变迁到状态$s_2$,并将继续执行子行为$\alpha_2=a_{i2}\circ a_e\circ a_{f2}$.
- 对 WEMA:此时,$\beta_1=a_{f1}\circ\Lambda\circ a_e\circ\Lambda$.WEMA 执行动作$a_{f1}$之后,从状态$t_1$变迁到状态$t_2$,并将继续执行子行为$\beta_2=\Lambda\circ a_e\circ\Lambda$.

计算$IS_{\alpha_2}=interfsrc(\alpha_1,u_n)=\{u_n,u_e\}$.将$(s_1,IS_{\alpha_1},t_1)$和$(s_2,IS_{\alpha_2},t_2)$代入公式(2)的两边,如果公式(2)的前件和后件均成立(事实上,只要代入(s_2,IS_{α_2},t_2)验证后件即可,因为本次验证的前件即上一验证步骤的后件,故而本次验证的前件一定是成立的),则转到步骤(3)继续递归验证;否则,如果公式(2)不成立,则行为不可信,报警.

(3),(4) 然后,继续根据公式(2)验证行为是否可信.

在步骤(3)和步骤(4),依次计算并代入"(s_2,IS_{α_2},t_2)和(s_3,IS_{α_3},t_3)"以及"(s_3,IS_{α_3},t_3)和(s_4,IS_{α_4},t_4)"到公式(2)进行验证即可.限于篇幅,具体过程从略.

(5) 最后,对于最后一个动作a_{f2},继续根据公式(2)验证其运行之后行为是否可信.

- 对 EMA:此时,$\alpha_5=a_{f2}=a_{f2}\circ\Lambda$.EMA 执行动作$a_{f2}$之后,从状态$s_4$变迁到状态$s_5$,并将继续执行子行为$\alpha_6=\Lambda$.
- 对 WEMA:此时,$\beta_1=\Lambda=\Lambda\circ\Lambda$.WEMA 执行空动作$\Lambda$,状态保持不变$s_4=s_5$,并将继续执行子行为$\beta_6=\Lambda$.

计算$IS_{\alpha_5}=interfsrc(\alpha_1,u_n)=\{u_n\}$.将$(s_4,IS_{\alpha_4},t_4)$和$(s_5,IS_{\alpha_5},t_5)$代入公式(2)的两边,如果公式(2)的前件和后件均成立(事实上,只需要代入(s_5,IS_{α_5},t_5)验证后件即可,原因同上),则 EMA 和 WEMA 在同步执行的过程中总是保持状态等价的,因而可以立即判定行为$\alpha=a_{i1}\circ a_{f1}\circ a_{i2}\circ a_e\circ a_{f2}$是可信的;否则行为不可信,报警.

示例结束.

由于传递无干扰是非传递无干扰的特例,对于传递无干扰,定理 2 可以简化为推论 1.

推论 1(传递无干扰的状态递归定义). 在传递无干扰安全策略下,机器 M 是满足无干扰的,或者说可信的,当且仅当:

$$\forall i_{(1\leqslant i\leqslant N+1)}.s_i\overset{w}{\sim}t_i \tag{3}$$

其中,$w=dom(a)$是定义 9 中观察动作 a 所属的安全域.

证明:根据传递无干扰安全策略定义,所有对 w 发生干扰的安全域,无需通过任何中间(intermediate)媒介安全域,即可直接对 w 进行干扰.这意味着无需保证 EMA 和 WEMA 在中间安全域(公式(2)中IS_{γ_i}和$IS_{\gamma_{i+1}}$)上的状态等价性,只要确保两者总是在 w 上等价即可.故公式(2)可简化为$\forall i_{(0\leqslant i\leqslant N)}.s_i\overset{w}{\approx}t_i\supset s_{i+1}\overset{w}{\approx}t_{i+1}$,即

$$\forall i_{(0\leqslant i\leqslant N)}.s_i\overset{w}{\sim}t_i\supset s_{i+1}\overset{w}{\sim}t_{i+1} \tag{4}$$

注意到,如下公式(5)和公式(6)是公式(4)前后相继的两次验证,且公式(6)的前件正好是公式(5)的后件.

$$s_i \overset{w}{\sim} t_i \supset s_{i+1} \overset{w}{\sim} t_{i+1} \tag{5}$$

$$s_{i+1} \overset{w}{\sim} t_{i+1} \supset s_{i+2} \overset{w}{\sim} t_{i+2} \tag{6}$$

再注意到,只有公式(5)验证通过之后,才会验证公式(6).这意味着当验证公式(6)时,其前件必然成立,故而对公式(6),只需要验证后件即可.

考察公式(4),令 $j=j+1$,其后件下标满足 $1\leqslant j\leqslant N+1$,故只需要验证 $s_1 \overset{w}{\sim} t_1,\ldots,s_{(N+1)} \overset{w}{\sim} t_{(N+1)}$ 即可.

此即公式(3),推论 1 得证. □

3 方法及实现

3.1 基本思想

大多数攻击都需要两个步骤:首先,定位可利用的漏洞;其次,通过漏洞执行攻击代码(称为 shellcode).只要打断以上环环相扣的两个步骤中的任何一步,即可以有效防御攻击.本文从第 2 步入手,建立在如下观察之上:几乎所有的 shellcode 都必须通过系统调用达成攻击.由此,一旦检测到系统调用序列的语义破坏了定义 9 的等式(1),即可以认定破坏了系统的可信性.

图 2 给出了基于无干扰理论的软件实时可信防护方法,包括 3 部分.

(1) 预期行为规范 EBS(expected behavior specification).

预期行为规范基于系统调用编写,描述了进程的预期行为,即进程预期可以可以做什么,或者预期不可以做什么.EBS 的说明和示例请分别参见第 3.3 节和附录.

(2) 每一个进程绑定一个预期行为规范 EBS.

所绑定的 EBS 描述了该进程的预期行为,并且 EBS 放置在内核中.注意:在原型实验中,我们默认内核是可信的,因而将 EBS 放置在内核中可以保护其完整性.

需要指出的是,内核完全有可能是不可信的,但如何确保内核可信超出了本文的研究范围.对于内核不可信情况下保证 EBS 完整性的可能方法有:采用现有的技术对内核进行加固[7,8,18−21],从而增强内核中 EBS 的完整性;或者将 EBS 放置在 TPM 安全协处理芯片、TrustZone 的安全世界等类似硬软件安全架构中,保证 EBS 的完整性.

(3) 安全增强的内核实时监测软件行为是否偏离预期.

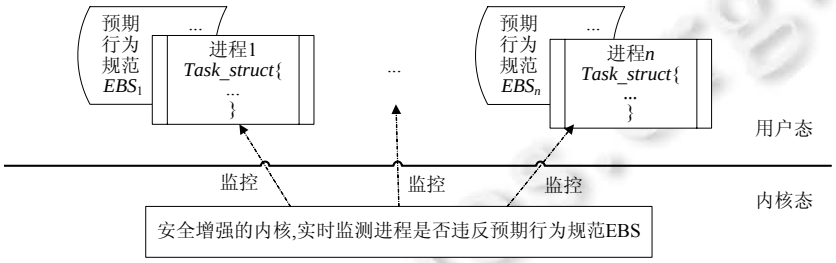


Fig.2 Approach for real-time-trusted protection of software based on noninterference

图 2 基于无干扰理论的软件实时可信防护方法

安全增强后的内核,对每个进程实时捕获其系统调用,并依据定理 2(推论 1)实时判定进程的行为是否偏离了预期(即 EBS),由此发现各类复杂攻击(如新型的 ROP 攻击以及传统的多态、混淆、变形和编码攻击等),理论上也可以识别未知攻击(因为即使是未知攻击,绝大多数情况下也需要执行 shellcode,而只要 shellcode 的行为偏离了 EBS,即可以被捕获).

根据文献[25],构成进程行为的原子动作可以是输入(input)、命令(command)、机器指令(instruction)等任意

可以被机器所执行的内容.本文选择系统调用作为原子动作,并以 Linux 操作系统作为原型环境.系统调用、EBS 和机器 M 这三者之间的关系如下:系统调用作为原子动作,其执行导致机器 M 发生状态变迁;EBS 作为安全策略,描述了系统调用所属安全域之间的干扰/无干扰关系;在此基础上,软件动态执行是可信的,当且仅当在 EBS 的控制下,其系统调用执行满足推论 1 所示的充要条件.

3.2 进程系统调用所属安全域之间的无干扰关系

根据根据 Linux 手册,基于不同的功能,Linux 系统调用可以分为如下 8 类[48],分别是:(1) 进程控制类(u_0);(2) 文件系统控制类(u_1);(3) 系统控制类(u_2);(4) 内存管理类(u_3);(5) 网络管理类(u_4);(6) 网络 Socket 控制类(u_5);(7) 用户管理类(u_6);(8) 进程间通信类(u_7).这 8 类系统调用可以看作 8 个不同的安全域,分别用 u_0~u_7 表示.由于系统调用之间的功能隔离关系(操作系统内核设计决定了系统调用的功能划分不会存在耦合关系,否则会导致非预期行为),这 8 个安全域之间是互相无干扰的,形式地: $\forall i \neq j.u_i \nrightarrow u_{j(0\leqslant i,j\leqslant 7)}$.这意味着安全策略由复杂的非传递安全策略退化为简单的传递安全策略,因而在判定软件真实行为与 EBS 之间是否存在偏差时,只需要使用推论 1 即可,这大大简化了行为可信验证算法的时间复杂度(参见第 3.5 节).需要指出的是,这种简化具有一般性,因为其是由系统调用所属安全域之间的无干扰关系决定的.

3.3 基于系统调用的EBS编写

限于篇幅,附录给出了本文所采用的一种 EBS 编写形式作为示例.事实上,如何编写和对应解析 EBS,并不限于示例 1 所示的方法,可以由使用者自己决定,只要基于系统调用刻画软件预期行为即可.

EBS 具有如下特点.

(1) EBS 可以采用无干扰信息流安全策略 $\nrightarrow$,也可以采用干扰信息流安全策略 $\rightsquigarrow$.由无干扰模型定义可知,两种安全策略本质上是等价的.经验表明:一般情况下,推荐使用 $\nrightarrow$ 安全策略,因为这可以得到复杂度更低的 EBS.

(2) EBS 描述了软件预期的行为,与具体攻击行为无关.因此,任何攻击,包括 Zero-Day 攻击在内,理论上只要导致软件行为偏离了预期,仍然可以被本文方法度量到.

(3) EBS 有两种类型:一是适用于平台所有软件的通用 EBS;二是绑定于特定软件、满足特定安全需求的特殊 EBS.两种 EBS 可以同时使用.当两种 EBS 同时存在的情况下,特殊 EBS 优先级高于通用 EBS 优先级,增加了灵活性.

3.4 系统调用实时获取

接下来,需要实时捕获被监控进程的系统调用(含参数)行为并与 EBS 进行对比,以判定进程的运行是否偏离了预期行为.方便起见,本文原型实验通过 ptrace 实现:将 PTRACE_PEEKUSER 作为 ptrace 的第 1 个参数进行调用,即可以实时取得与待度量进程相关的系统调用信息.其中,

(1) EAX 当中存储的是系统调用的索引号,亦即指明了具体的系统调用.

(2) 在 32 位架构中,当系统调用参数的个数小于 5 个时,EBX,ECX,EDX,ESI,EDI 当中依次存储着系统调用所有的参数;当系统调用参数的个数大于 5 个时(这种情况不多见),EBX 指向一块内存区域,这个区域中就依序存放着系统调用的所有参数.

(3) 在 64 位架构当中,%rdi,%rsi,%rdx,%r10,%r8,%r9 依次存放着系统调用参数信息.

3.5 实时可信性验证

第 3.2 节说明,Linux 系统调用的 8 个安全域之间互相无干扰,即 $\forall i \neq j.u_i \nrightarrow u_{j(0\leqslant i,j\leqslant 7)}$.因此任何两个安全域之间都无法利用第三安全域作为中间安全域(intermediate domain)来间接传递信息,这本质上退化为传递无干扰,故使用推论 1 来对进程 u_p 的实时可信性进行验证.推论 1 可以表述为如下验证算法 1.

算法 1. 进程 u_p 实时可信度量(验证)算法.

1. 任意时刻,实时获取进程 u_p 的本次系统调用,不失一般性,假设为本次执行的系统调用为 a.

2. 遍历 EBS,计算原子动作 a 在 EBS 安全策略控制下的预期执行动作β=*wexpected*(a,w).根据*wexpected*(·)的定义,最终有$\beta=a$(EBS 安全策略预期允许执行 a)或者$\beta=\Lambda$(EBS 安全策略预期不允许执行 a).
3. 如果$\beta=a$,则进程真实执行与预期执行一致(真实执行动作和预期执行动作都是 a,两者执行之后状态一致,故在 EBS 关注的 w 上等价),根据推论 1,u_p 执行 a 后可信,转步骤 1;如果$\beta=\Lambda$,则进程真实执行与预期执行不一致(真实执行动作 a,预期执行空动作Λ,两者执行之后导致状态空间不一致,故在 EBS 所关注的 w 上不等价),u_p 执行 a 后不可信,报警.

算法 1 结束.算法 1 只需要在每次系统调用 a 执行时,判定 a 是否被 EBS 安全策略所允许即可,因而其时间复杂度为 $O(1)$.

4 实验评估

实验平台采用 TOSHIBA 笔记本电脑,CPU Intel i5 2.5G,内存 Samsung 8G.操作系统 Linux Fedora,内核版本 2.6.43.8-1.fc15.i686.PAE.实验测试了在 ROP 攻击以及传统代码多态、代码混淆、代码变形和代码编码等对抗形式下对攻击行为的可信度量功能.表 1 给出了实验结果.

Table 1 Attacks that can be measured and protected in real time

表 1 可被实时度量和防护的攻击方法

攻击方法	是否可以实时度量和防护
ROP(return-oriented programming)	√
代码多态(polymorphism)	√
代码混淆(obfuscation)	√
代码变形(mutation)	√
代码编码(encoding)	√

表 1 显示,本文的方法可以检测到 ROP(return-oriented programming)攻击.ROP 代表了现代攻击的发展趋势,其不寻求注入代码,而是从已有的“干净代码”中寻找合适的“指令片段”(gadget),通过“ret 或者与 ret 语义等价的指令”将前述指令片段链接形成完整的攻击代码,以达成攻击目的.ROP 可以高度自动化地构造图灵完全的攻击,危害巨大.为此,研究人员提出了多种对抗方法[49−53],取得了长足的进展.早期的 ROP 防御侧重于检测可疑的 ret 指令,但没有处理与 ret 语义等价的指令(如 jmp/pop 等)[49],对 ROP 的防御是不完全的.随后的工作实现了对各类 ROP 攻击的全覆盖,其基本思想在于消除 ROP 指令片段,或者检测并阻止(ROP 指令片段所导致)的非法间接跳转.但这些方法均需要额外的条件,如,需要被保护应用的非直接跳转指令和地址[50],或者被保护应用的指令片段 IG(instruction & gadget)信息[51];需要硬件的特殊功能[52];需要特殊的编译器[53];需要对受保护应用进行重写[50,53]等等.不同于这些工作,本文的方法基于如下观察:ROP 是一种代码重用方法,其本身并不能达成攻击,而仍然要通过调用敏感系统调用来实现恶意功能.因此,通过度量应用程序系统调用与 EBS 规范的偏离可以发现基于 ROP 技术所发起的攻击.严格来说,本文方法并不能检测 ROP 攻击本身,这是与文献[49−53]相比的不足之处.但从另外一个角度,在面向大量商业应用和通用度量平台的一般情况下——此时可能难以进行预处理(因而难以获得应用本身的信息[50,51]),不一定具备特殊硬件和软件(因而不一定满足特定功能的 CPU[52]和编译器[53]要求),且不一定能够进行代码重写(如开启了完整性度量架构 IMA),本文的方法可以作为上述方法的补充,结合应用.

对于表 1 中余下的 4 种经典攻击代码对抗方法,实验结果表明亦可以有效度量.从实验结果还可以发现,通过分析和抽取已有 shellcode 的特征,利用较少的 EBS 规范(实验中采用了 7 条主要 EBS 规范),就可以覆盖主要的攻击流程和大量的攻击代码,实现对软件可信性的实时度量,这表明方法是高效的.

表 2 对实时度量的时间负载进行了分析.实时度量所引起的时间负载由两部分构成:一是捕获系统调用及其参数所花的时间 t_1;二是分析系统调用是否偏离了 EBS 安全策略所花的时间 t_2.t_2 主要是字符串比较等指针操作,时间相对较快,大致取 $t_2\approx t_1$.再假设在未进行实时度量时(no real-time measurement)软件的运行时间为 T_{NRM}=

t_0,则进行实时度量时(real-time measurement)软件的运行时间为 $T_{RM}=t_0+t_1+t_2\approx T_{NRM}+2\times t_1$,此时,可求时间负载$\zeta$为如下等式(7),故只要得到 t_1/t_0 即可.

$$\zeta=T_{RM}/T_{NRM}=1+2\times t_1/t_0 \tag{7}$$

利用 time 命令和 strace 命令,可以立即求出 $T_0=t_0$ 以及 $T_1=t_0+t_1$,两者相比有:

$$t_1/t_0=T_1/T_0-1 \tag{8}$$

将等式(8)代入等式(7),即得ζ.

注意,time 命令会给出 3 种时间.

(1) 真实时间(real):软件从开始执行到结束执行的时间.

(2) 用户 CPU 时间(user):软件在用户态花费的 CPU 时间.

(3) 系统 CPU 时间(sys):软件在内核态花费的 CPU 时间.

那么对应可以定义 $T_{r0},T_{r1};T_{u0},T_{u1};T_{s0},T_{s1}$.由于真实时间包含了软件由于等待如 I/O 等资源的挂起时间,其意义不大,一般关注后两者(后两者之和 user+sys 给出了软件运行的真实 CPU 时间),则等式(7)和等式(8)可以对应改写如下.

$$\zeta=(\zeta_u+\zeta_s)/2,\zeta_u=1+2\times t_{u1}/t_{u0},\zeta_s=1+2\times t_{s1}/t_{s0} \tag{9}$$

$$t_{u1}/t_{u0}=T_{u1}/T_{u0}-1,t_{s1}/t_{s0}=T_{s1}/T_{s0}-1 \tag{10}$$

表 2 给出了 T_{u1}/T_{u0} 和 T_{s1}/T_{s0} 的实验结果.

Table 2 Time-load analysis of real-time measurement

表 2 实时度量时间负载分析

用户时间比(user time ratio):T_{u1}/T_{u0}			系统时间比(sys time ratio):T_{s1}/T_{s0}		
Max	Min	Average	Max	Min	Average
15.00	1.00	5.05	11.50	1.30	5.58

取平均时间比,将 $T_{u1}/T_{u0}=5.05$ 和 $T_{s1}/T_{s0}=5.58$ 代入等式(9),有 $t_{u1}/t_{u0}=4.05$ 以及 $t_{s1}/t_{s0}=4.58$;再代入等式(8),立即可得:$\zeta_u=9.1,\zeta_s=10.16,\zeta=9.63$.因此在实时度量时,一般情况下,用户时间负载$\zeta_u$约为 9.10 倍,系统时间负载$\zeta_s$约为 10.16 倍,总体时间负载$\zeta_s$约为 9.63 倍.

以上各类时间负载总体约为 10 倍,但这已经本方法所能达到的接近最优结果,且实际测试表明,该负载并不影响用户的实际使用体验.原因如下.

(1) 实验所选取的测试数据是接近最坏情况下的测试数据.本文方法对系统调用进行捕获和度量,对其他函数和指令则不做任何处理,显然,被测试代码中系统调用相关代码所占的比例越高,则对时间负载影响越大.注意到 shellcode 正好满足上述特征:考虑到存储空间,shellcode 会尽可能短小精干,去除冗余指令以调用系统调用完成攻击功能.换句话说,系统调用相关代码所占比例高.因此,本文直接选择 shellcode 代码为测试对象,所得的 10x 时间负载约为最坏情况下的结果.一般情况下,测试结果表明,时间负载会介于 1.6x~10x 之间.

(2) 前面已说明,时间负载由两部分构成:捕获系统调用和参数的 t_1 以及分析系统调用是否偏离 EBS 的 t_2,且取 $t_2\approx t_1$.实验中,为方便起见,使用 strace 命令获取系统调用和参数信息(strace 仍然基于 ptrace 实现).由于 strace 是 Linux 标准命令,因此 t_1 可以认为是优化后的最优结果,从而基于表 2 所计算到的时间负载ζ_u,ζ_s和ζ也可以认为是接近最优的结果.

(3) 以用户交互性强的应用为测试对象表明,本文方法所引起的时间负载并不影响用户使用感受.以 Firefox 为例,利用“time strace firefox”命令启动火狐浏览器并浏览文本、图片和视频等各类网页,统计时间信息并计算可得:若将 real time 视作 1,则 user time 约占 23%,sys time 约占 8%,即陷入内核 CPU 的运行时间(sys time)仅仅约占 8%,由于内核 CPU 时间(sys time)由系统调用和非系统调用时间两部分构成,因而实际捕获系统调用和参数信息的时间不到 8%.

根据测试结果,上述时间占比不影响用户使用感受.

5 结 论

运行时可信度量是可信计算有待解决的一个关键问题,本文提出了一种基于无干扰的软件实时可信度量方法,尝试为可信度量的理论构建和实践应用提供一种新的思路.选择无干扰模型是因为研究表明:无干扰模型可以以一种统一的形式描述“行为符合预期、完整性和机密性”等不同的主流“可信”定义,从而可以在统一的框架下对不同内涵的可信性进行研究.在此基础上,给出了行为可信性的实时判定算法(时间复杂度 $O(1)$),解决了无干扰模型难以应用于实时系统的难题.针对理论的实践应用问题,根据无干扰模型的定义,选择系统调用作为基本原子动作,将软件真实行为定义为原子动作的序列,并根据系统调用所属安全域之间的无干扰关系定义预期行为规范 EBS.最后,通过判定软件真实行为与预期行为之间是否存在偏差,实现对软件行为的实时可信度量.原型实验证明了本文方法的有效性.

后续的研究拟从如下几个方面入手:一是探索如何结合人工智能技术自动抽取恶意代码样本特征并形成预期行为规范库;二是研究适用于实时或者准实时环境的漏洞定位技术;三是研究移动智能终端、物联网等环境下的实时可信度量和防护问题.

References:

[1] Shen CX, Zhang HG, Wang HM, Wang J, Zhao B, Yan F, Yu FJ, Zhang LQ, Xu MD. Research on trusted computing and its development. Science in China: Information Science, 2010,53(3):405−433.

[2] Feng DG, Qin Y, Wang D, Chu XB. Research on trusted computing technology. Journal of Computer Research and Development, 2011,48(8):1332−1349 (in Chinese with English abstract).

[3] Trusted Computing Group. TCG architecture overview (Revision 1.4). 2007. https://trustedcomputinggroup.org/wp-content/uploads/TCG_1_4_Architecture_Overview.pdf

[4] Sailer R, Zhang XL, Jaeger T, Doorn VL. Design and implementation of a TCG-based integrity measurement architecture. In: Blaze M, ed. Proc. of the 2004 USENIX Security Symp. (Security 2004). Berkeley: USENIX Association Press, 2004. 223−238.

[5] Shi E, Perrig A, Doorn LV. BIND: A fine-grained attestation service for secure distributed systems. In: Martin DC, ed. Proc. of the 2005 IEEE Symp. on Security and Privacy (Oakland 2005). New York: IEEE Computer Society Press, 2005. 154−168.

[6] Jaeger T, Sailer R, Shankar U. PRIMA: Policy-reduced integrity measurement architecture. In: Ferraiolo D, ed. Proc. of the ACM Symp. on Access Control Models and Technologies (SACMAT 2006). New York: ACM Press, 2006. 19−28.

[7] Loscocco PA, Wilson PW, Pendergrass JA, McDonell CD. Linux kernel integrity measurement using contextual inspection. In: Ning P, ed. Proc. of the 2007 ACM Workshop on Scalable Trusted Computing (STC 2007). New York: ACM Press, 2007. 21−29.

[8] Carbone M, Cui WD, Lu L, Lee WK, Peinado M, Jiang XX. Mapping kernel objects to enable systematic integrity checking. In: Al-Shaer E, ed. Proc. of the 2009 ACM Conf. on Computer and Communications Security (CCS 2009). New York: ACM Press, 2009. 555−565.

[9] Schiffman J, Moyer T, Shal C, Jeger T, McDaniel P. Justifying integrity using a virtual machine verifier. In: Gates C, ed. Proc. of the 2009 IEEE Annual Computer Security Applications Conf. (ACSAC 2009). New York: IEEE Computer Society Press, 2009. 83−92.

[10] Kil C, Sezer EC, Azab AM, Ning P, Zhang X. Remote attestation to dynamic system properties: Towards providing complete system integrity evidence. In: Verissimo P, ed. Proc. of the 2009 IEEE/IFIP Int'l Conf. on Dependable Systems & Networks (DSN 2009). New York: IEEE Computer Society Press, 2009. 115−124.

[11] Xu W, Ahn GJ, Hu H, Zhang X, Seifert JP. DR@FT: Efficient remote attestation framework for dynamic systems. In: Gritzalis D, Preneel B, Theoharidou M, eds. Proc. of the 2010 European Conf. on Research in Computer Security (ESORICS 2010). Berlin: Springer Press, 2010. 182−198.

[12] John B, Corey K, Xeno K, Amy H. BIOS chronomancy: Fixing the core root of trust for measurement. In: Sadeghi AR, ed. Proc. of the 2013 ACM Conf. on Computer and Communications Security (CCS 2013). New York: ACM Press, 2013. 35−36.

[13] Zhang ZK, Ding XH, Tsudik G, Cui JH, Li ZJ. Presence attestation: The missing link in dynamic trust boostrapping. In: Thuraisingham B, ed. Proc. of the 2017 ACM Conf. on Computer and Communications Security (CCS 2017). New York: ACM Press, 2017. 89−102.

[14] Abera T, Asokan N, Davi L, Ekberg, JE, Nyman T, Paverd A, Sadeghi AR, Tsudik G. C-flat: Control-flow attestation for embedded systems software. In: Weippl E, ed. Proc. of the 2016 ACM Conf. on Computer and Communications Security (CCS 2016). New York: ACM Press, 2016. 743−754.

[15] Zhang J, Yuan F, Xu Q. DeTrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans. In: Ahn GJ, ed. Proc. of the 2014 ACM Conf. on Computer and Communications Security (CCS 2014). New York: ACM Press, 2014. 153−166.

[16] Matetic S, Ahmed M, Kostiainen K, Dhar A, Sommer D, Gervais A, Juels A, Capkun S. ROTE: Rollback protection for trusted execution. In: Kirda E, Ristenpart T, eds. Proc. of the 2017 USENIX Security Symp. (Security 2017). Berkeley: USENIX Association Press, 2017. 1289−1306.

[17] Yu FJ, Chen L, Zhang HG. Virtual trusted platform module dynamic trust extension. Ruan Jian Xue Bao/Journal of software, 2017, 28(10):2782−2796 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5174.htm [doi: 10.13328/j.cnki.jos.005174]

[18] Deng L, Zeng QK. Inner TCB based application protection. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):1042−1058 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5016.htm [doi: 10.13328/j.cnki.jos.005016]

[19] Deng L, Zeng QK. Method to efficiently protect applications from untrusted OS kernel. Ruan Jian Xue Bao/Journal of Software, 2016,27(5):1309−1324 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5017.htm [doi: 10.13328/j.cnki.jos.005017]

[20] Lin J, Liu CY, Fang BX. IVirt: Runtime environment integrity measurement mechanism based on virtual machine introspection. Chinese Journal of Computers, 2015,38(1):191−203 (in Chinese with English abstract). http://cjc.ict.ac.cn/qwjs/view.asp?id=4414

[21] Chen ZF, Li QB, Zhang P, Wang W. Kernel integrity measurement method based on memory forensic. Ruan Jian Xue Bao/Journal of Software, 2016,27(9):2443−2458 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4875.htm [doi: 10.13328/j.cnki.jos.004875]

[22] Zhang X, Huang Q, Shen CX. 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 with English abstract). http://cjc.ict.ac.cn/qwjs/view.asp?id=3015

[23] Zhang X, Chen YL, Shen CX. Non-interference trusted model based on processes. Journal on Communications, 2009,30(3):6−11 (in Chinese with English abstract).

[24] Qin X, Chang CW, Shen CX, Gao L. Research on trusted terminal computer model tolerating untrusted components. Acta Electronica Sinica, 2011,39(4):1−6 (in Chinese with English abstract).

[25] Rushby J. Noninterference, transitivity, and channel-control security. Technical Report, CSL-92-02, Menlo Park: SRI Int'l, 1992. 1−47.

[26] Eggert S, Meyden RVD, Schnoor H, Wilke T. The complexity of intransitive noninterference. In: Fríncke D, ed. Proc. of the 2011 IEEE Symp. on Security and Privacy (Oakland 2011). New York: IEEE Computer Society Press, 2011. 196−211.

[27] Hadj-Alouane NB, Lafrance S, Lin F, Mullins J, Yeddes MM. On the verification of intransitive noninterference in multilevel security. IEEE Trans. on Systems, Man, and Cybernetics—Part B: Cybernetics, 2005,35(5):948−958.

[28] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: Ning P, ed. Proc. of the 2007 ACM Conf. on Computer and Communications Security (CCS 2007). New York: ACM Press, 2007. 552−561.

[29] Goguen JA, Meseguer J. Security policy and security models. In: Schell R, ed. Proc of the 1982 IEEE Symp. on Security and Privacy (Oakland'82). New York: IEEE Computer Society Press, 1982. 11−20.

[30] Meyden RVD. What, indeed, is intransitive noninterference? In: Biskup J, López J, eds. Proc. of the 2007 European Symp. on Research in Computer Security (ESORICS 2007). Berlin: Springer-Verlag, 2007. 235−250.

[31] Ryan PYA. Mathematical models of computer security. In: Focardi R, Gorrieri R, eds. Proc of the 2001 Foundations of Security Analysis and Design. Berlin: Springer-Verlag, 2001. 1−62.

[32] Focardi R, Gorrieri R. Classification of security properties (Part I: Information flow). In: Focardi R, Gorrieri R, eds. Proc of the 2001 Foundations of Security Analysis and Design. Berlin: Springer-Verlag, 2001. 331−396.

[33] Zhang HG, Yan F, Fu JM, Xu MD, Yang Y, He F, Zhan J. Research on theory and key technology of trusted computing platform security testing and evaluation. Science in China: Information Science, 2010,40(2):167−188.

[34] Xu MD, Zhang HG, Zhao H, Li JL, Yan F. Security analysis on trust chain of trusted computing platform. Chinese Journal of Computers, 2010,33(7):1165−1176 (in Chinese with English abstract). http://cjc.ict.ac.cn/qwjs/view.asp?id=3120

[35] Toby M, Daniel M, Matthew B, Peter G, Timothy B, Sean S, Corey L, Xinn G, Gerwin K. SeL4: From general purpose to a proof of information flow enforcement. In: Sommer R, ed. Proc. of the 2013 IEEE Symp. on Security and Privacy (Oakland 2013). New York: IEEE Computer Society Press, 2013. 415−429.

[36] Mads D, Roberto G, Oliver S. Formal verification of information flow security for a simple ARM-based separation kernel. In: Sadeghi AR, ed. Proc. of the 2013 ACM Conf. on Computer and Communications Security (CCS 2013). New York: ACM Press, 2013. 223−234.

[37] Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA. A sense of self for Unix process. In: McHugh J, Dinolt G, eds. Proc. of the '96 IEEE Symp. on Security and Privacy (Oakland'96). New York: IEEE Computer Society Press, 1996. 120−128.

[38] Yang J, Sangho L, Evan D, Weiren W, Mattia F, Taesoo K, Alessandro O, Wenke L. RAIN: Refinable attack investigation with on-demand inter-process information flow tracking. In: Thuraisingham B, ed. Proc. of the 2017 ACM Conf. on Computer and Communications Security (CCS 2017). New York: ACM Press, 2017. 377−390.

[39] Enrico M, Lucky O, Panagiotis A, Emiliano DC, Gordon R, Gianluca S. MaMaDroid: Detecting Android malware malware by building markov chains of behavioral models. In: Bauer L, ed. Proc of 2017 ISOC Network and Distributed System Security Symp. (NDSS 2017). Reston: Internet Society Press, 2017. 1−15.

[40] Marko D, Simone A, Zvonimir R, Ivo U. Evaluation of Android malware detection based on system calls. In: Verma R, ed. Proc. of 2016 ACM Int'l Workshop on Security and Privacy Analytics (IWSPA 2016). New York: ACM Press, 2016. 1−8.

[41] Kimberly T, Salahuddin JK, Aristide F, Lorenzo C. CopperDroid: Automatic reconstruction of Android malware behaviors. In: Hutton T, ed. Proc of the 2015 ISOC Network and Distributed System Security Symp. (NDSS 2015). Reston: Internet Society Press, 2015. 1−15.

[42] Yang C, Xu ZY, Gu GF, Yegneswaran V, Porras P. DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications. In: Kutyłowski M, Vaidya J, eds. Proc. of the 2014 European Symp. on Research in Computer Security (ESORICS 2014). Berlin: Springer-Verlag, 2014. 163−182.

[43] Alessandro R, Aristide F, Lorenzo C. A system call-centric analysis and stimulation technique to automatically reconstruct Android malware behaviors. In: Stamatogiannakis M, ed. Proc. of the 2013 European Workshop on Systems Security. New York: ACM Press, 2013. 1−6.

[44] Calvin K, Timothy R. Noninterference and intrusion detection. In: Hinton H, ed. Proc. of 2002 IEEE Symp. on Security and Privacy (Oakland 2002). New York: IEEE Computer Society Press, 2002. 177−188.

[45] Zhang F, Zhang C, Chen W, Hu FN, Xu MD. Noninterference analysis of trust of behavior in cloud computing system. Chinese Journal of Computers, 2019,42(4):736−755 (in Chinese with English abstract). http://cjc.ict.ac.cn/online/onlinepaper/zf-20194179 2414.pdf

[46] George CN. Proof-carrying code. In: Lee P, ed. Proc. of the ACM '97 SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'97). New York: ACM Press, 1997. 1−14.

[47] George CN, Peter L. Safe kernel extensions without run-time checking. In: Peterson K, Zwaenepoel W, eds. Proc. of the '96 USENIX Symp. on OS Design and Implementation (OSDI'96). Berkeley: USENIX Association Press, 1996. 229−243.

[48] IBM DeveloperWorks. List of Linux system calls. 2002 (in Chinese). 2018. https://www.ibm.com/developerworks/cn/linux/kernel/ syscall/part1/appendix.html

[49] Li JK, Wang Z, Jiang XX, Grace M, Baharam S. Defeating return-oriented rootkits with "return-less" kernels. In: Morin C, ed. Proc. of the 2010 European Conf. on Computer Systems (EuroSys 2010). New York: ACM Press, 2010. 195−208.

[50] Zhang C, Wei T, Chen ZF, Duan L, Szekeres L, McCamant S, Song D, Zou W. Practical control flow integrity & randomization for binary executables. In: Sommer R, ed. Proc. of the 2013 IEEE Symp. on Security and Privacy (Oakland 2013). New York: IEEE Computer Society Press, 2013. 559−573.

[51] Cheng YQ, Zhou ZW, Yu M, Ding XH, Deng RH. Ropecker: A generic and practical approach for defending against ROP attacks. In: Hutton T. Proc. of the 2014 ISOC Network and Distributed System Security Symp. (NDSS 2014). Reston: Internet Society Press, 2014. 1−14.

[52] Vasilis P, Michalis P, Angelos DK. Transparent ROP exploit mitigation using indirect branch tracing. In: King S, ed. Proc. of the 2013 USENIX Security Symp. (Security 2013). Berkeley: USENIX Association Press, 2013. 447−462.

[53] Kaan O, Leyla B, Andrea L, Davide B, Engin K. G-free: Defending return-oriented programming through gadget-less binaries. In: Gates C, ed. Proc. of the 2010 IEEE Annual Computer Security Applications Conf. (ACSAC 2010). New York: ACM Press, 2010. 49−58.

附中文参考文献:

[2] 冯登国,秦宇,汪丹,初晓博.可信计算技术研究.计算机研究与发展,2011,48(8):1332−1349. http://crad.ict.ac.cn/CN/abstract/abstract2333.shtml

[17] 余发江,陈列,张焕国.虚拟可信平台模块动态信任扩展方法.软件学报,2017,28(10):2782−2796. http://www.jos.org.cn/1000-9825/5174.htm [doi: 10.13328/j.cnki.jos.005174]

[18] 邓良,曾庆凯.引入内可信基的应用程序保护方法.软件学报,2016,27(4):1042−1058. http://www.jos.org.cn/1000-9825/5016.htm [doi: 10.13328/j.cnki.jos.005016]

[19] 邓良,曾庆凯.一种在不可信操作系统内核中高效保护应用程序的方法.软件学报,2016,27(5):1309−1324. http://www.jos.org.cn/1000-9825/5017.htm [doi: 10.13328/j.cnki.jos.005017]

[20] 林杰,刘川意,方滨兴.IVirt:基于虚拟机自省的运行环境完整性度量机制.计算机学报,2015,38(1):191−203. http://cjc.ict.ac.cn/qwjs/view.asp?id=4414

[21] 陈志锋,李清宝,张平,王炜.基于内存取证的内核完整性度量方法.软件学报,2016,27(9):2443−2458. http://www.jos.org.cn/1000-9825/4875.htm [doi: 10.13328/j.cnki.jos.004875]

[22] 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. http://cjc.ict.ac.cn/qwjs/view.asp?id=3015

[23] 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.

[24] 秦晰,常朝稳,沈昌祥,高丽.容忍非信任组件的可信终端模型研究.电子学报,2011,39(4):934−939.

[34] 徐明迪,张焕国,赵恒,李峻林,严飞.可信计算平台信任链安全性分析.计算机学报,2010,33(7):1165−1176.

[45] 张帆,张聪,陈伟,胡方宁,徐明迪.基于无干扰的云计算环境行为可信性分析.计算机学报,2019,42(4):736−755.

[48] IBM 开发工厂.Linux 系统调用.2002. https://www.ibm.com/developerworks/cn/linux/kernel/syscall/part1/appendix.html

附录. EBS 编写示例

攻击者在定位漏洞之后,往往会执行一些通用攻击操作,以达成进一步攻击目的,这些通用攻击操作包括(但不限于)生成 shell、开启反向连接、清空防火墙规则、关闭地址空间布局随机化 ASLR 等等.针对这些攻击,这里给出部分 EBS 示例如下:

1. EBS 规范 1:预期不能生成 shell.

$u_p \nrightarrow u_{[0][0]} == \{2\}$

FOR INDEX==2:{1==“*bin*/*bash*”∨1==“*bin*/*sh*”∨1==“*bin*/tc*bash*”∨1==“*bin*/*csh*”∨1==“*bin*/*dash*”∨}

$u_{[i][j]}$的下标分别是 i,j,其表明:i 是安全域 u_i 的编号,j 是 u_i 中子安全域编号.因此,这里 $u_{[i][j]}=u_{[0][0]}$的含义是 u_0 安全域中的第 1 个子安全域.类似的,如果是 u_1 安全域中的第 3 个子安全域,则表示为 $u_{[1][2]}$.

回到 $u_{[0][0]}=\{2\}$,查阅[48]可知,u_0 安全域为进程控制类系统调用,其中,编号 2 对应 execve,故 $u_{[0][0]}=\{2\}=\{execve\}$.再结合 **FOR INDEX==2**:语句的参数安全策略,由于 execve 的第 1 个参数是字符串类型,指明了要运行的二进制文件名,因此规范 1 的含义是:进程 u_p 预期不能运行/bin 目录下的 bash,sh,tcsh,csh 和 dash 文件以开启 shell.

解释:攻击者突破系统之后,一般需要开启 shell 以进一步执行命令.规范 1 度量并禁止了这种行为.需要说明的是:Linux 的 shell 并不只有上述 5 种类型(规范 1 是根据实验者机器的配置情况编写的),使用者可以根据实际安全需求容易地更新规范 1.

2. 规范 2:预期不能开启非法监听端口连接或者不能进行反向连接.

$u_p \nrightarrow u_{[5][0]} == \{2,3\}$

FOR INDEX==2:{2→*sin_port*!=*PORT_LIST*$_1$}

FOR INDEX==3:{2→*sa_family*==AF_INET,2→*sin_port*==*PORT_LIST*$_2$,2→*sin_addr*==*IP_LIST*}

规范 2 的含义是:进程 u_p 预期对子安全域 $u_{[5][0]}$无干扰.查阅文献[48]可知,u_5 中 2 和 3 号系统调用分别为 bind 和 connect.进一步查阅 bind 和 connect 参数信息,两者的第 2 个参数均是常量结构体指针 const struct sockaddr*,该结构体的内部元素 sa_familiy 指明了协议族;sin_port 指明了端口号;sin_addr 指明了远程连接的 IP 地址.结合 **FOR INDEX==2**:语句的参数安全策略可知,进程 u_p 预期对所有不在端口列表 *PORT_LIST*$_1$ 当中的端口都不能开放监听.类似地,结合 **FOR INDEX==3**:语句的参数安全策略可知,进程 u_p 预期不能对端口列表 *PORT_LIST*$_2$ 和地址列表 *IP_LIST* 中的机器发起远程连接.

解释:攻击者突破系统之后,往往会开启受害机器监听端口监听外部连接,以方便进一步入侵.为此,规范 2 的安全策略预期进程 u_p 不会开启必要端口(即 *PORT_LIST*$_1$ 中列出的端口)之外的任何端口,从而能够度量并禁止攻击者非法开启监听端口的行为.另一方面,在内网中,由于防火墙的存在,外网机向内网机的连接会触发警报.此时,攻击者往往会从受害的内网机向外网特定机器发起反向连接.规范 2 度量并禁止了这种反向连接(反弹端口)行为.

3. 规范 3:预期不能清空防火墙规则.

$u_p \nrightarrow u_{[0][1]} == \{2\}$

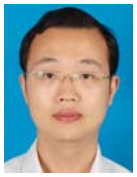
FOR INDEX==2:{1==/*sbin*/*iptables*}∧{2==[“/*sbin*/*iptables*”,“*–F*”]}

规范 3 的含义是:进程 u_p 预期对子安全域 $u_{[0][1]}$无干扰.由于规则 1 中 $u_{[0][0]}$和这里 $u_{[0][1]}$均是来自于 u_0 的子安全域,因此两者赋予不同的编号 j=0 和 j=1.类似地,查阅文献[48]可知:规范 3 表明,进程 u_p 预期不能带参数“*–F*”执行/*sbin*/*iptables* 文件以清空所有防火墙规则.

解释:规范 3 度量并禁止了清空防火墙规则行为.

限于篇幅,这里给出 3 个 EBS 示例.

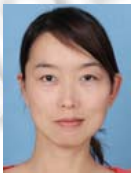
根据我们的分析结果,大多 shellcode 的代码有很强的功能重合性.换句话说,绝大多数 shellcode 的恶意功能都局限于生成 shell、开启反向连接、清空防火墙规则、关闭地址空间布局随机化 ASLR、修改/etc/passwd 文件、提升权限和重启等,只是其机器码形式或者对抗手段(多态、混淆、变形和编码)不一致.在我们的实验中,通过提炼这些重合功能,只需要不到 10 条通用 EBS,即可检测到大多数 shellcode,表明了方法的有效性.



张帆(1977－),男,湖北当阳人,博士,副教授,CCF 专业会员,主要研究领域为信息系统安全,软件安全,机器学习在网络空间安全中的应用.

张聪(1968－),男,博士,教授,主要研究领域为多媒体信号处理,模式识别,多媒体安全.

徐明迪(1980－),男,博士,研究员,CCF 专业会员,主要研究领域为网络空间安全,信息系统安全.



刘小丽(1981－),女,博士,讲师,主要研究领域为信息安全,移动计算.

赵涵捷(1963－),男,博士,教授,博士生导师,主要研究领域为移动计算,云计算,物联网,量子计算,网络及信息安全.

胡方宁(1976－),女,博士,讲师,主要研究领域为通信,嵌入式系统.