软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(6):1602−1619 [doi: 10.3724/SP.J.1001.2012.04083]

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

基于广义污点传播模型的操作系统访问控制*

杨 智[1,2,3,4+], 殷丽华[1,4], 段淼毅[1], 吴金宇[1,4], 金舒原[1,4], 郭 莉[1,4]

[1](中国科学院 计算技术研究所,北京 100190)

[2](解放军信息工程大学,河南 郑州 450004)

[3](中国科学院 研究生院,北京 100049)

[4](信息内容安全技术国家工程实验室,北京 100190)

Generalized Taint Propagation Model for Access Control in Operation Systems

YANG Zhi[1,2,3,4+], YIN Li-Hua[1,4], DUAN Mi-Yi[1], WU Jin-Yu[1,4], JIN Shu-Yuan[1,4], GUO Li[1,4]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Information Engineering University of PLA, Zhengzhou 450004, China)

[3](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[4](National Engineering Laboratory for Information Security Technologies, Beijing 100190, China)

+ Corresponding author: E-mail: zynoah@163.com

Yang Z, Yin LH, Duan MY, Wu JY, Jin SY, GUO L. Generalized taint propagation model for access control in operation systems. *Journal of Software*, 2012,23(6):1602−1619. http://www.jos.org.cn/1000-9825/4083.htm

Abstract: Dynamically adjusting the security label of each subject is a main approach to improving the availability of MAC models. It includes the method of security label range and the method of taint propagation. The former lacks the support for a less proviledge subject, and the latter has a known covert channels. In this paper, a model called generalized taint propagation model (GTPM) is proposed to protect the confidentiality and integrity of operating systems. It inherits the least privilege characteristic of taint propagation model (TPM), expands the semantics of TPM to close the known covert channels, and introduces declassification and decontamination capacities of subjects to avoid accumulating contamination. The paper also introduces its specification using communicating sequential processes (CSP) language to clear the formal semantics of a GTPM operating system's behaviors of information flow control; Moreover, the study noninterference with declassification in CSP verification model of process equivalence, and proves that abstract GTPM system have the security property of noninterference with declassification in virtue of FDR tool. Finally, this paper uses an example to demonstrate its improvement of availability.

Key words: taint propagation; covert channel; CSP; noninterference; least privilege; information flow control; operating system

摘 要: 动态调整安全级是目前提高强制访问控制模型可用性的主要途径,它大致包括两类方法.其中,安全级范

* 基金项目: 国家自然科学基金(61070186); 国家高技术研究发展计划(863)(2009AA01Z438); 国家重点基础研究发展计划(973)(2007CB311100, 2011CB311801)

收稿时间: 2010-10-18; 修改时间: 2011-04-28; 定稿时间: 2011-06-24

围方法对主体权限最小化的支持不够,而污点传播方法存在已知隐蔽通道.提出了保护操作系统保密性和完整性的广义污点传播模型(generalized taint propagation model,简称 GTPM),它继承了污点传播在最小权限方面的特点,拓展了污点传播语义,以试图关闭已知隐蔽通道,引入了主体的降密和去污能力以应对污点积累;还利用通信顺序进程(CSP)语言描述了模型的规格,以明确基于 GTPM 的操作系统的信息流控制行为的形式化语义;基于 CSP 的进程等价验证模型定义了可降密无干扰,并借助 FDR 工具证明形式化构建的抽象 GTPM 系统具有可降密无干扰安全性质.最后,通过一个示例分析了模型的可用性提升.

关键词: 污点传播;隐蔽通道;通信顺序进程;无干扰;最小权限;信息流控制;操作系统

中图法分类号: TP316 **文献标识码**: A

计算机安全模型的主要目标包括:阻止非法的读取或修改信息;控制对信息的传播和间接污染.常见的访问控制模型如 DAC,RBAC 等模型较好地支持了前一目标,然而对后一目标支持不够.例如,DAC 或 RBAC 策略可以表达允许用户 *A* 读取用户 *B* 的数据,却无法有效表达 *B* 对 *A* 传播这些数据的控制意图.而有效实现后一目标,可以防范常见的利用不可信的或有漏洞的程序传播泄露信息或间接破坏系统完整性.同时,信息流模型试图控制系统内各实体间的信息流动,防止未经许可的信息流,可以有效支持第 2 个目标,它已成为现代操作系统主要的安全模型之一.然而,已有信息流模型如经典的 BLP 和 Biba 模型过于严格,要求绝对的信息单向流动,几乎不可能构造出一个可用的系统.目前,提高可用性的主要途径是动态调整安全级,包括两类方法:一是安全级范围方法[1],二是污点传播方法[2].

在安全级范围方法中,可信主体的当前安全级可在一个范围内动态变化,使得可信主体在不违背模型规则的情况下获得更多访问权限.这体现了弱静态原则.该方法的优点是实施简单,易于分析安全性.然而,主体的安全级函数是与系统状态无关的函数,在主体的生命周期内,主体的安全级范围不会发生变化,这意味着主体的权限保持不变.这并不符合最小权限原则[3],难以满足一些实际需求(可参见第 4 节).最小权限原则指系统只赋予每个主体完成每个工作时所必需的权限,而实施静态授权的安全级范围方法难以表达这种权限的动态变化.

还有一类方法借鉴了软件测试中的用于跟踪和分析输入数据在程序内部流向的污点传播机制[4].该方法跟踪和控制各种保密信息或恶意数据在主客体间的信息流动,根据这些数据流向动态调整主体安全级,我们称为污点传播方法.具体来讲,对于保密性模型,信息合法流动的条件是接收方的最大保密级(ceiling)不小于发送方的当前保密级.消息流动后,接收方当前保密级变为收发双方当前保密级的最小上界.该过程中,发送方作为污点,感染了接收方,造成接收方的保密级升高以及发送能力变弱;对于完整性模型,信息合法流动的条件是接收方的最小完整级(floor)不大于发送方的当前完整级.消息流动后,接收方当前完整级变为收发双方当前完整级的最大下界.该过程中,发送方作为污点,感染了接收方,造成接收方的完整级降低以及发送能力变弱.在污点传播方法中,主体的安全级与历史访问情况相关,主体的权限变化反映信息流动,这在一定程度上体现了最小权限原则.然而该方法存在隐蔽通道[5−8](见第 2 节),并且有污点积累问题.

基于以上分析,我们提出广义污点传播模型(generalized taint propagation model,简称 GTPM).模型的目标是为存在不可信应用程序的操作系统提供保密性和完整性保护.模型的基本思想是:定义实体的安全标记是由非等级的输入型和输出型类型标签组成的集合,实施对实体的动态隔离保护;引入主体能力概念,区分提高和降低安全级为两种不同能力,以有效表达主体发送和接受能力;进一步拓展污点传播语义.模型的特点是关闭污点传播方法中已知隐蔽通道,并且可有效防止污点积累.另外,形式化构建和分析抽象 GTPM 系统也是本文的一个主要工作.

本文在分析相关工作的基础上,首先定义 GTPM 的相关概念,然后用 CSP 语言描述模型规格,最后,在定义可降密无干扰安全的基础上,分析抽象 CTPM 系统的无干扰性质.

1 相关工作

污点传播模型由于在主体权限最小化方面的特点,一直是信息流模型研究中的主要方向之一.不少研究偏

重于污点传播机制的设计和实现,较少涉及污点传播控制和污点积累问题,也没有考虑隐蔽通道问题.例如:用于保密性保护的高水标模型 High-Water-Mark[9],要求主体新创建的客体级别等于该主体曾读/写过的客体的最高级别;用于完整性保护的 Biba 的低水标模型[2]Low-Water-Mark 要求主体下读客体时,降低该主体的级别到该客体级别.LOMAC[10]系统考虑了更多的实现技术;IX[11]系统设立了主体的最大污染级;UMIP 模型[12]研究了参考 DAC 策略来分配实体完整级和制定污点传播策略;文献[13]研究了根据系统日志和实体依赖关系自动化的为实体分配完整级和制定污点传播策略.这些模型和系统的不足之处一是除了 IX 系统外,均没有为主体设立最大污染级,主体可被任意污染,使得主体自己不能表达对信息的接受意愿;二是由于主体积累的污点越来越多,其安全级是单调变化的,这意味着它很快不能访问污点较少的主客体.而在实际应用中,主客体经过可信的软件检查无病毒后,可以适当去除它们的一些污点;三是存在隐蔽通道问题,像高水标模型允许渗透到系统内部高级别的木马程序,将读到的秘密信息写入到一个已有公开文件中去,这存在信息泄露的危险[9].另外,这类模型还存在一个共性的隐蔽通道.我们在下面给出描述.

一些研究通过分布式的降密和去污来缓解污点积累.IFEDAC 模型[14]中主客体的当前标记是所有污染过它的用户集合,当污染过主体的用户集合不大于对客体有标记管理权限的用户的集合时,一是该主体可以去除客体标记中部分用户标识(污点),二是它可以扩大对该客体有访问权限的用户集合.这从两方面缓解了污点积累.但是,这种以用户账号作为污点的模型,对于单用户使用的桌面操作系统中由第三方不可信软件发起的内部攻击的防护不是很有效.Asbestos 系统[5]中的主体标记由各个范畴内的级别组成,级别取值从小到大是[*,0,1,2,3],级别*表示主体是该范畴内信息的拥有者.在污点传播过程中,该主体可以不用提高级别(污点积累),该主体也可以降低其他主体的该范畴内的级别.但是,Asbestos 系统中的污点传播主要体现在进程间通信方面,没有考虑对客体的访问控制,也没有考虑特殊情况处理.如,有些信息流动不满足模型规则,但其实是应用场景要求的可信的信息流动.

在以上的这些污点传播模型中,一个非污点进程在收到另一个污点进程的信息时,系统会自动地将这个非污点进程调整为污点进程,这种标记调整机制存在一类共性的隐蔽通道问题.文献[5]给出了一个相应的隐蔽通道构造方案.考虑一个污点进程 A 试图传递 1 位敏感信息给一个未被污染的进程 C.攻击者构建了两个未被污染的进程 B_0 和 B_1,它们重复地发送心跳信息给进程 C.A 如果想发送值为 i 的信息给 C,A 就发送一条信息给 B_i.如果 B_i 尝试接受就会造成 B_i 被污染,变成污点进程.如果 C 在下一个心跳时间内没有收到 B_i 的心跳信息,就知道 A 发了值为 i 的信息给自己,从而实现了隐蔽通信.这是一次信息传递,由于现代操作系统可以在极短的时间内完成创建进程这样的系统调用,所以通过连续快速创建进程 B_0 和 B_1 这样的中间进程,进程 A 可以在很短的时间内传送几千字节的信息给 C.该隐蔽通道有很多变形.产生该隐蔽通道的原因是系统会隐式地改变 B_i(implicit change)的标记.

随着计算机性能的提高,该隐蔽通道带宽会极大地增加.为了防止此类隐蔽通道,作为 Asbestos 系统的后继 HiStar 系统[6],虽然沿用了 Asbestos 的标记方案,但主体的标记调整要由主体自己来完成,系统不再隐式地调整主客体标记,从而关闭该隐蔽通道.例如,让上面的示例运行在 HiStar 系统中,如果 B_i 不主动变为污点进程,则不论 A 是否发消息给它,B_i 尝试接受来自 A 的消息一定失败;如果 B_i 主动变为污点进程,则虽然 B_i 最终可以知道 A 是否给它发消息,但在 A 发和没有发消息的两种情况下,C 都没有收到来自 B_i 的心跳信息,所以隐蔽通道被关闭.Flume 系统[7,8]基于类型标签构建标记方案,该系统也采用了类似 HiStar 的标记调整方案,而没有采用污点传播策略,也不存在这类隐蔽通道.但是,这两种相似的方案会导致一些可用性问题:一是标记调整需要主体的主动参与,最终表现为使用者的参与,这对用户提出了更高的要求;二是需要改动应用程序,以支持这种标记显式调整;三是频繁地调用标记调整相关的系统调用,增加了系统开销.

针对上述模型和系统的不足,本文提出广义污点传播模型.一方面仍然支持标记的隐式调整,保证模型的可用性;另一方面通过拓展污点传播语义来关闭上述提到的隐蔽通道,以保证模型的安全性.同时,通过主体删除标签来体现主体降密去污能力,防止污点积累.通过引入有约束的特定访问能力,以保证主体能力的有效性.本模型引入了类型标签和基于标签的主体能力等概念,早期的文献[15]给出类似的标签概念,但是它的每个标签

表示系统内某个实体的标识,基于这些标签构建的标记难以表达实体的动态隔离,而且它的保护模型(如完整性保护)不支持污点传播.安全性上,我们的模型和先前污点传播模型的最主要的不同是,它不存在污点传播模型中已知的隐蔽通道.

2 基于 GTPM 的操作系统信息流控制

2.1 GTPM的非形式化描述

系统中所有被保护实体分为主体和客体两类,其中主体集合 $\tilde{S}$ 是系统中所有活动对象(如进程)的集合,客体集合 $\tilde{O}$ 是不活动实体(如文件)的集合.主体的访问权限集合 OP 包括读(read)、写(write)、执行(exec)、创建(create)、删除(del)客体权限以及读消息权限(ipcrecv).保密性保护域集合 DS 是代表信息获取方所属类型(领域)的输出保护型标签符号集合.例如,用标签 $d_k \in DS$ 标记用户 Carl 的私人数据,则可以读该数据的主体属性至少包含 d_k.相似地,完整性保护域集合 DI 是代表信息修改方所属类型(领域)的输入保护型标签符号集合.例如,用标签 $d_t \in DI$ 标记系统配置信息,则可以写该信息的主体属性至多包含 d_t.

定义 1(实体标记). 集合 $L \subseteq DS \times DI$ 定义了系统中所有实体的所有可能标记.每个实体 o 的标记 L_o 是保密性标记和完整性标记的二元组(S_o, I_o).对于保密性标记,S_o 支配(高于)$S_{o'}$,当且仅当 $S_{o'} \subseteq S_o$;对于完整性标记,I_o 支配(高于)$I_{o'}$,当且仅当 $I_o \subseteq I_{o'}$.

标记在偏序关系上可形成一个格.需要指出的是,最高完整级是ϕ(记作$\top_{DI}$),表示该级别实体不能被任何污点实体修改和污染.最低完整级是 DS(记作$\perp_{DI}$),表示不限制该级别实体被任何其他实体修改和污染,可用于标记系统中不需要修改保护的实体.相应的最高和最低保密级记为$\top_{DS}$和$\perp_{DS}$.

定义 2(能力). 集合$\Phi \subseteq C \cup E$ 定义了系统中所有可能的主体能力.其中:

(1) $C \subseteq (DS \cup DI) \times \{+,-\}$定义了系统中的主体调整自身标记的所有可能的能力.对于主体s拥有的能力 $C_s \subseteq C$,如果 $d^+ \in C_s$,则主体 s 可以添加标签 d 到自己相应的标记中;如果 $d^- \in C_s$,则主体 s 可以从自己相应的标记中删除标签 d.定义客体的调整标记能力为空集.

(2) $E \subseteq DS \times DI \times (\tilde{O} \times P)$ 定义了系统中主体的有约束的特定访问能力的所有可能.对于主体 s 拥有的能力 $E_s \subseteq E$,如果$(S_{constrain}, I_{constrain}, \langle o, op\rangle) \in E_S$,则条件 $S_s \cap S_{constrain} = \varnothing$且 $I_s \cap I_{constrain} = \varnothing$满足时,主体 s 可按权限 op 访问实体 o,以特定能力的访问不再受到 s 和 o 的标记之间关系的约束.定义客体的有约束特定访问能力为空.

对于主体 s 和保密性标签 d,若 $d^+ \in C_s$ 而 $d^- \notin C_s$,意味着 s 只可以提高保密性标记级别到 $S_s \cup \{d\}$,以获得保密性类型为 d 的信息,但不能泄露该信息给其他实体 $o(d \notin S_o)$;若 $d^- \in C_s$,意味着 s 可以降低自身保密性标记级别以传播保密性类型为 d 的信息;若 $d^+ \in C_s$ 且 $d^- \in C_s$,则 s 对保密性类型为 d 的信息具有获取和传播的完全控制权.对于主体 s 和完整性标签 d(如 d 表示从网络输入信息),若 $d^+ \in C_s$ 而 $d^- \notin C_s$,意味着 s 可降低自身完整性标记级别到 $I_S \cup \{d\}$,以读取来自完整性类型为 d 的信息,但降级后不能再修改完整性标记为 I_S 的信息(如 I_S 表示系统的重要配置信息);若 $d^- \in C_s$,意味着主体 s 读取完整性类型为 d 的信息后,仍然可以通过提高完整性标记级别来修改完整性标记为 $I_S - \{d\}$的信息;若 $d^+ \in C_s$ 且 $d^- \in C_s$,则 s 可接受完整性类型为 d 的信息污染,并且可以去污.

我们用 $C_s^+ = \{d \mid d^+ \in C_s\}$ 表示主体 s 可以添加的所有标签集合,C_s^+ 不仅反映了 s 的接受信息的能力,也反映了 s 的接受的意愿. $C_s^- = \{d \mid d^- \in C_s\}$ 表示主体 s 可以删除的所有标签集合,C_s^- 反映了 s 的发送信息的能力,$C_s^{\pm} = \{d \mid d^+ \in C_s \wedge d^- \in C_s\}$ 表示 s 可完全控制的所有标签集合,$C_s^{\pm}$ 反映了 s 对保护域完全控制的能力.我们用 $\top_{DS}^{+}$表示可以添加任何保密性标签到自己标记中的能力,用 $\perp_{DS}^{-}$ 表示可以删除自身保密性标记中任何标签的能力,用工 $_{DS}^{\pm}$表示前两种保密性处理能力的和.对应的,$\top_{DI}^{-}$表示可以删除自身完整性标记中任何标签的能力,$\perp_{DI}^{+}$ 表示可以添加任何完整性标签到自己标记中的能力,工 $_{DI}^{\pm}$表示前两种完整性处理能力的和.

主体的有约束的特定访问能力是主体调整自身标记能力的补充.信息流保密性保护要求一个使用了升密能力的主体如果没有降密能力,将会失去对所有公开文件写的能力,但是有的应用场景会允许该主体写指定的公开文件.这一特定行为被认为是安全可信的,不会造成信息泄露,但不认为它写信息到其他所有公开文件是安

全的(如果仍然被认为是安全的行为,可以通过赋予主体降密能力来解决).同样的,信息流完整性保护要求一个感染了污点的主体如果没有去污能力,将会失去对高完整级(无污点)信息的修改(写)的能力,但是有的应用场景会允许该主体修改指定的高完整级文件.这一特定行为被认为是安全可信的,不会造成完整性破坏,但不认为它修改其他所有高完整级文件是可信的(如果仍然被认为是安全的行为,可以通过赋予该主体去污能力来解决).针对这些情况,我们可以将那些指定的特殊访问需求看成是该主体的特定访问能力,可以不受严格的信息流保护约束.在有多个域时,需要指出在主体读了指定域的保密性信息或者感染了指定域的完整性污点后,不能再使用这些能力,即不将主体在任何标记状态下的特定访问都看成安全可信,以对主体使用该能力进行约束,防止主体滥用该特定能力.如果约束为空,则主体可以在任何标记状态下使用该能力.

对于主体 s 以权限 op 访问实体 o,引入一个判断该主体是否具有相应特定访问能力的谓词 $CEP(s,o,op)$.当且仅当存在一个$(S_{constrain},I_{constrain},\langle o,op\rangle)\in E_S$,$S_s\cap S_{constrain}=\varnothing$且 $I_s\cap I_{constrain}=\varnothing$时,该谓词为真.

主体能力取自产生该主体的可执行客体属性,由授权管理员设置和管理,在主体创建时由系统读取和载入.

第 1 节提到的隐蔽通道的原因是系统会隐式地改变 B_i(implicit change)的标记.如果 A 未发消息给 B_i,B_i 接受失败并不会被污染;如果 A 发消息给 B_i,B_i 接受成功就会自动被污染.为此,在 GTPM 的定义 3 中要求,如果 B_i 的能力允许 B_i 被污染,则不论 A 是否发来消息,B_i 只要尝试从污点进程 A 读消息都会被自动污染,因为此时的通信只是没有传送数据而已,是一种隐式通信.另外,定义 4 的标记显式改变(explicit change)允许 B_i 尝试接受消息前,主动改变标记变成污点进程(B_i 的能力要允许),则不论 A 是否发来消息,B_i 已是污点进程,也不存在隐蔽通道.

定义 3(安全的信息流动和标记隐式变化). 实体 p 以权限 op_p 访问实体 q 或者后者以权限 op_q 访问前者实现的从实体 p 到 q 的信息流动是安全的,当且仅当条件(a)或条件(b)满足:

(a) $S_p-C_p^{\pm}\subseteq S_q\cup C_q^{+}$ and $I_p-C_p^{\pm}\subseteq I_q\cup C_q^{+}$;

(b) $CEP(p,q,op_p)$ or $CEP(q,p,op_q)$.

如果条件(a)满足,在消息流动后,q 的标记被污染,从而发生标记的隐式变化,即

$$S_q\leftarrow S_q\cup(S_p-C_p^{\pm}),I_q\leftarrow I_q\cup(I_p-C_p^{\pm}).$$

定义 3 规定的安全的信息流动有两种情况:一是同时考虑信息流出方发送能力和信息流入方接受能力情况下,低保密级的信息向高保密级流动,高完整级的信息向低完整级流动,并实施污点传播.从便于实际应用考虑,这里规定信息流出方 p 的标记不发生隐式变化,为此涉及到 p 的能力是 $C_p^{\pm}$;二是信息流出方和流入方中至少一方对此信息流动被赋予特定能力,对于凭该能力实现的信息流动,不实施污点传播.针对不同类型的实体和操作,解释如下:

(i) 主体 p 可从主体 q 接收到消息,当且仅当条件(a)、条件(b)、条件(d)同时满足,或者条件(a)、条件(c)、条件(d)同时满足:

(a) q 存在;

(b) $S_q-C_q^{\pm}\subseteq S_p\cup C_p^{+}$ and $I_q-C_q^{\pm}\subseteq I_p\cup C_p^{+}$;

(c) $CEP(p,q,ipcrecv)$;

(d) q 已发消息给 p.

若 p 做出从 q 接收消息的动作(系统调用),则 p 的标记调整如下:

- 如果条件(a)、条件(b)满足,则 p 的标记被污染(调整)为 $S_p\leftarrow S_p\cup(S_q-C_q^{\pm}),I_p\leftarrow I_p\cup(I_q-C_q^{\pm})$.
- 如果条件(a)不满足或条件(b)不满足,则 p 的标记被调整为 $S_p\leftarrow S_p\cup(C_p^{+}\cap DS),I_p\leftarrow I_p\cup(C_p^{+}\cap DI)$.

(ii) 主体 p 可读或动态加载客体 q,当且仅当条件(a)、条件(b)同时满足,或者条件(a)、条件(c)同时满足:

(a) q 存在;

(b) $S_q\subseteq S_p\cup C_p^{+}$ and $I_q\subseteq I_p\cup C_p^{+}$;

(c) $CEP(p,q,read)$.

若 p 做出读或动态加载 q 的动作(系统调用),则 p 的标记调整如下:

- 如果条件(a)、条件(b)满足,则 p 的标记被污染为 $S_p \leftarrow S_p \cup S_q, I_p \leftarrow I_p \cup I_q$.
- 如果条件(a)不满足或条件(b)不满足,则 p 的标记被调整为 $S_p \leftarrow S_p \cup (C_p^+ \cap DS), I_p \leftarrow I_p \cup (C_p^+ \cap DI)$.

(iii) 主体 p 可写客体 q,当且仅当条件(a)、条件(b)同时满足,或者条件(a)、条件(c)同时满足:

(a) q 存在;

(b) $S_p - C_p^{\pm} \subseteq S_q$ and $I_p - C_p^{\pm} \subseteq I_q$;

(c) $CEP(p,q,write)$.

(iv) 主体 p 可创建标记为(S_q,I_q)的客体 q,当且仅当下面的条件成立:

(a) q 不存在;

(b) $S_p - C_p^{\pm} \subseteq S_q$ and $I_p - C_p^{\pm} \subseteq I_q$.

(v) 主体 p 可删除客体 q,当且仅当下面的条件成立:

(a) q 存在;

(b) $S_p - C_p^{\pm} \subseteq S_q$ and $I_p - C_p^{\pm} \subseteq I_q$.

(vi) 若主体 p 可执行客体 o,创建主体(进程)q,当且仅当条件(a)、条件(b)、条件(c)同时满足,或者条件(a)、条件(d)同时满足:

(a) o 存在;

(b) $S_o \subseteq S_p \cup C_p^+$ and $I_o \subseteq I_p \cup C_p^+$;

(c) $S_p - C_p^{\pm} \subseteq S_o \cup C_q^+$ and $I_p - C_p^{\pm} \subseteq I_o \cup C_q^+$;

(d) $CEP(p,o,exec)$ (其中,新主体 q 的能力取自客体 o 的属性).

若 p 做出执行客体 o 的动作(系统调用),则 p 的标记调整如下:

- 如果条件(a)、条件(b)满足,则 p 的标记被污染为 $S_p \leftarrow S_p \cup S_o, I_p \leftarrow I_p \cup I_o$;
- 如果条件(a)不满足或条件(b)不满足,则 p 的标记被调整为 $S_p \leftarrow S_p \cup (C_p^+ \cap DS), I_p \leftarrow I_p \cup (C_p^+ \cap DI)$.

创建 q 后,如果条件(a)、条件(b)、条件(c)满足,q 的标记设置为 $S_q \leftarrow (S_p - C_p^{\pm}) \cup S_o, I_q \leftarrow (I_p - C_p^{\pm}) \cup I_o$;否则,$q$ 的标记设置为 $S_q \leftarrow S_o, I_q \leftarrow I_o$.

该执行过程可以看做 p 读访问 o 和 p 写访问 q 两个过程:前一过程中,p 至少可以知道 o 是否存在的信息;后一过程中,如果 p 带参数启动 q,则 q 直接读到 p 发来的参数信息,即使不带参数执行,如果只有 p 才会启动 q,则启动本身就是 q 获得的来自 p 的信息.

删除主体通过该主体接受其他主体要求退出消息(情况(i))后主动退出完成.或由系统强制删除该主体,客体重命名,客体复制、移动可以通过创建、读、写和删除客体操作并遵循相应的安全要求来实施.

定义 4(安全的标记显式变化). X 是主体 s 的保密性或完整性标记,X'是 s 请求的关于 X 的新值,从 X 到 X' 的显式变化是安全的,当且仅当

$$X' - X \subseteq C_s^+ \text{ and } X - X' \subseteq C_s^-.$$

相应地,X 是主体 s 的保密性或完整性标记,Y 是客体 o 的与 X 同类型的标记,Y'是 s 请求的关于 o 的标记 Y 的新标记,从 Y 到 Y'的显式变化是安全的,当且仅当

$$X - C_s^{\pm} \subseteq Y \subseteq X \cup C_s^{\pm} \text{ and } X - C_s^{\pm} \subseteq Y'.$$

主体可以在能力范围内主动改变自己的标记,如文字处理进程主动提高密级以产生第 1 份机密文档;PGP 软件主动降密(如果 PGP 没有升密能力)以发送文档密文.结合主体标记的显式变化,安全信息流动条件放宽为 $S_p - C_p^- \subseteq S_q \cup C_q^+$ 和 $I_p - C_p^- \subseteq I_q \cup C_q^+$,可实现灵活的信息流控制,标记显式变化是隐式变化的有效补充.

主体可以修改客体标记的要求是与定义 3 一致的,可通过 s 读 o 内容→s 将内容写到临时文件→s 删除 o→s 创建标记为 Y'的 o→s 读临时文件→s 写 o→s 删除临时文件完成,给出客体标记改变要求是为了简化系统处理.

基于上述定义,下一节给出模型的确切语义.

2.2 GTPM的形式化规格

GTPM 形式化规格(specification)试图通过数学符号对模型实现系统的性质及行为进行精确、简洁的描述.这种确切描述反映了模型的精确要求,可以消除实现语义上的二义性,同时也是进行形式化验证(verification)和安全性分析的基础.本文采用通信顺序进程 CSP[16]描述模型的规格,CSP 是分布式并发软件系统规格和设计的进程代数方法,比较适合研究信息流安全性质[17].规格描述了创建主体和主体退出、进程间通信、主体对客体读、写、创建和删除、改变实体标记等实现规格.这里的规格是初步的,按照这些规格可能无法构建一个真正的系统.例如:设定进程通信的缓冲区已满时,新到消息覆盖还未取走的消息,而没有刻画可靠的通信过程;设定可以写无限多的数据到客体中,而没有刻画容量受限下的磁盘和客体行为;设定进程间可以任意顺序执行和占用 CPU,而没有刻画体现公平的进程调度.但是,本文给出的规格也是框架性和基础性的,表达了 GTPM 的本质安全要求,能够比较容易地扩展、细化和进行非安全环节的调整.

图 1 描述了 GTPM 实现系统的组成原理,引用监视器(reference monitor,简称 RM)依据授权策略控制主体对被保护实体的访问行为,限制非授权访问;安全服务器(SecServer)根据主客体标记以及访问动作做出决策;标记状态管理器(label state manager,简称 LSM)存储和维护系统实体的标记状态信息以及实体能力等信息.这 3 个组件构成系统安全核.安全核应该足够小,以保证效率和利于进行安全性分析和测试.主客体是系统的受保护对象.每个主体 i 通过信道 api_i 调用系统服务,调用被 RM 捕获,RM 通过信道 *intercom* 获得来自 SecServer 的策略,SecServer 通过信道 *intermgr* 获得主客体标记和其他相关信息.使用信道 *io* 描述主体通过 RM 对客体的访问.

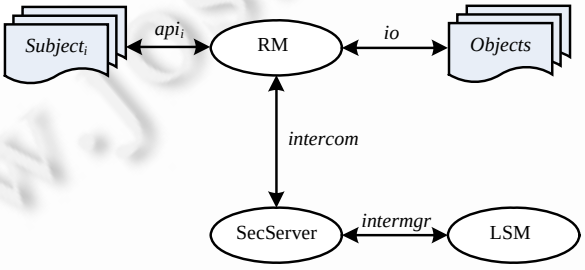

Fig.1 Composition and control principles of GTPM

图 1 GTPM 的组成与控制原理

2.2.1 用户进程

在常见的操作系统中,用户进程通过系统调用访问系统资源和服务,如进程通信、读或写文件等,不直接交互的用户进程间的并发执行用混合算子描述.用 CSP 进程 $USER_i$ 代表用户进程 i.所有用户进程行为描述:

$$USERS=(|||i:P-\{1\}@USER_i)|||USER\_LIVE_1,$$

其中,P 是系统中可能的主体标识符 PID 的集合;$USER\_LIVE_1$ 是系统中的第 1 个启动进程,设 $C_{USER\_LIVE_1}=DS\cup DI\times\{+,-\}$,它可以执行任何客体创建新的主体,从而实现系统初始化.虽然 CSP 没有进程派生算子,但是实际系统通常有最大进程个数限制,因此可以在 CSP 程序中预先建立用户进程,只是在未接受到创建出生信号前,这些 CSP 进程就像操作系统中的僵尸进程一样处于不可用(idle)状态,在进程接收到出生信号后,它们就处于活跃(live)状态,当接收到退出命令后,再次进入 idle 状态.单个用户进程行为描述:

$$USER_i=born_i?msg\rightarrow USER\_LIVE_i.$$

$USER_i$ 在 live 状态可以通过信道 api_i 选择执行系统调用,如下描述:

- $USER\_LIVE_i=SEND_i|RECV_i|WRITE_i|READ_i|CREATE_i|DEL_i|EXEC_i|EXIT_i|CHANGELABEL_i$.
- $SEND_i=api_i!(send,to,msg)\ldots\rightarrow USER\_LIVE_i$,表示发送消息 *msg* 给接受进程 *to*.
- $RECV_i=api_i!(recv,from)\rightarrow api_i?(result,msg)\ldots\rightarrow USER\_LIVE_i$ 表示接受来自进程 *from* 的消息 *msg*,*result*={*ok*,*error*}指出是否接受到消息.这里没有考虑超时.

- $WRITE_i=api_i!(write,which,msg)\ldots\rightarrow USER\_LIVE_i$ 表示写信息 *msg* 到客体 *which*.
- $READ_i=api_i!(read,which)\rightarrow api_i?(result,msg)\ldots\rightarrow USER\_LIVE_i$ 表示读客体 *which* 内容到 *msg* 中.
- $EXEC_i=api_i!(exec,which,msg)\rightarrow USER\_LIVE_i$ 表示执行客体 *which*.系统中客体有可执行和不可执行两种类型,客体的类型由管理员设置.创建的新主体的 *pid* 由系统随机分配.
- $EXIT_i=api_i!(exit)\rightarrow USER_i$ 表示进程退出.在系统调用后,$USER_i$ 处于 idle 状态.
- $CREATE_i=api_i!(create,which,label)\rightarrow USER\_LIVE_i$ 表示创建标记为 *label* 的客体 *which*,默认客体标记与主体相同.
- $DEL_i=api_i!(del,which)\rightarrow USER\_LIVE_i$ 表示删除客体 *which*.
- $CHANGELABEL_i=api_i!(changelabel,flag,which,S,I)\rightarrow USER\_LIVE_i$ 表示修改自己或客体 *which* 的标记为 *S*,*I*,其中,*flag*={*self*,*object*}.

部分系统调用没有返回结果(如写、执行客体等),是因为调用结果也是一种信息,可能导致隐蔽通道.为了提高可用性,同时不违背模型安全性要求,安全核可以将每次用户进程的系统调用结果发送给一个安全调用结果查询服务进程 $USER_{mgr}$,其 *S*,*I*,*C* 为{·},{·},$DS\cup DI\times\{+,-\}$,$USER_{mgr}$ 分析系统日志并监控所有用户行为,实时评价用户,产生黑(白)名单.当接收到 $USER_i$ 的调用结果查询请求 $api_i!(send,mgr,apirequery)$时,$USER_{mgr}$ 根据对 $USER_i$ 行为评价结果决定是否响应,并记录这次查询,双方的通信符合模型安全性要求.这涉及提高系统可用性问题,本文不再赘述.

2.2.2 引用监视器

RM 并发处理来自不同用户进程的系统调用,它用子进程 RM_ITEM_i 在信道 api_i 上监听来自 $USER_i$ 的调用请求,并实施访问控制;当多个系统调用涉及同一主体或客体时,进程 *SYN_SO* 要求 RM 串行处理这些系统调用,以保证处理的原子性子进程;子进程 *BUFF* 描述用于进程通信的缓冲区行为.

$$RM=((|||i:P@RM\_ITEM_i)\underset{aSYN}{\|}SYN\_SO)\underset{aBUFF}{\|}BUFF.$$

进程 RM_ITEM_i 用相应的子进程处理各种类型的系统调用,RM_ITEM_i 是这些子进程的选择复合.

$$RM\_ITEM_i=RM\_SEND_i|RM\_RECV_i|RM\_WRITE_i|RM\_READ_i|RM\_CREATE_i|RM\_DEL_i\ldots$$

进程 *SYN_SO* 通过锁机制和 CSP 的并发机制实现对同一主客体标记状态信息的互斥访问.

一个系统调用涉及到两个以上主客体时,加锁优先顺序定为标识号从小到大的客体,然后是标识号从小到大的主体,以防止死锁.

$$SYN\_SO=SYN\_S|||SYN\_O.$$

进程 *SYN_SO* 的事件集 *aSYN*={*enterS*,*leaveS*,*enterO*,*leaveO*}.其中:

- 主体同步进程 $SYN\_S=|||i:P@SYN\_S_i$,其中,$SYN\_S_i=enterS.i\rightarrow leaveS.i\rightarrow SYN\_S_i$.
- 客体同步进程 $SYN\_O=|||i:O@SYN\_O_i$,其中 $SYN\_O_j=enterO.j\rightarrow leaveO.j\rightarrow SYN\_O_j$,*O* 是所有可能客体标识集合.

进程 *BUFF* 缓存和中转进程间通信信息,为简单起见,这里为每对通信进程设置一个消息槽.

$$BUFF=|||from:P@BUFF\_1_{from},$$

$$BUFF\_1_{from}=(|||to:P@BUFF\_2_{from,to,0,msg,\{\cdot\},\{\cdot\}}).$$

消息槽提供放入/取出消息服务,消息还未取出时,再次写入新消息,旧消息被覆盖.由于主体的标记会动态变化,消息槽同时还会记录发送者发送消息时的标记信息.消息槽状态变迁如下:

$$BUFF\_2_{from,to,full,msg,S,I}=buff_{from,to}.in?(msg',S',I')\rightarrow BUFF\_2_{from,to,1,msg',S',I'}|buff_{from,to}.out!msg\rightarrow$$

$$BUFF\_2_{from,to,0,null,\{\cdot\},\{\cdot\}}|buff_{from,to}.test!(full,S,I)\rightarrow BUFF\_2_{from,to,full,msg,S,I}.$$

当 $USER_i$ 发消息 *msg* 给 $USER_j$ 时,*RM* 将 *msg* 放入消息槽 $BUFF\_2_{i,j,full,msg}$ 中,在 $USER_j$ 取出消息槽中消息前,*RM* 向 *SecServer* 询问访问控制策略,策略允许时,*RM* 才会将取到的消息发给 $USER_j$.

$$RM\_SEND_i=api_i?(send,j,msg)\rightarrow \text{if } i==j \text{ then } RM\_ITEM_i \text{ else } enterS.i\rightarrow intercom!send\rightarrow intercom?(S_i,I_i)\rightarrow$$

$$buff_{i,j}.in!(msg,S_i,I_i)\rightarrow leaveS.i\rightarrow RM\_ITEM_i,$$

$RM_RECV_i=api_i?(recv,j)\rightarrow$if $i==j$ then $api_i!(error,null)\rightarrow RM\_ITEM_i$ else $enterS.\min(i,j)\rightarrow enterS.\max(i,j)\rightarrow buff_{j,i}.test?(full,S_j,I_j)\rightarrow intercom_i!(irecv,j,full,S_j,I_j)\rightarrow intercom_i?policy\rightarrow$if $policy==accept$ then $buff_{j,i}.out?msg\rightarrow api_i!(ok,msg)\rightarrow leaveS.\min(i,j)\rightarrow leaveS.\max(i,j)\rightarrow RM\_ITEM_i$ else $api_i!(error,empty)\rightarrow leaveS.\min(i,j)\rightarrow leaveS.\max(i,j)\rightarrow RM\_ITEM_i$.

当 $USER_i$ 读、写、创建和删除客体 j 的时候,如果 RM 查询到的策略是允许,RM 通过信道 io 告诉系统执行相关的客体管理操作.下面是写客体处理,其他类似.

$RM\_WRITE_i=api_i?(write,j,msg)\rightarrow enterO.j\rightarrow enterS.i\rightarrow intercom_i!(iwrite,j)\rightarrow intercom_i?policy\rightarrow$ if $policy==accept$ then $io!(writefile,j,msg)\rightarrow leaveS.i\rightarrow leaveO.j\rightarrow RM\_ITEM_i$ else $leaveS.i\rightarrow leaveO.j\rightarrow RM\_ITEM_i$.

当 $USER_i$ 创建新主体请求被允许时,$SecServer$ 返回一个 $newid$,RM 发出出生信号以激活 $USER_{newid}$.

$RM_EXEC_i=api_i?(exec,j,msg)\rightarrow enterO.j\rightarrow enterS.i\rightarrow intercom_i!(iexec,j)\rightarrow intercom_i?(policy,newid)\rightarrow leaveS.i\rightarrow leaveO.j\rightarrow$if $policy==accept$ then $born_{newid}!msg\rightarrow RM\_ITEM_i$ else RM_ITEM_i.

主体退出时,除了通知系统更新标记状态信息外,还要清空该主体的发送缓冲区.

$$RM\_EXIT_i=api_i?exit\rightarrow enterS.i\rightarrow RM\_eEXIT_{i,|P|},$$

$RM\_eEXIT_{from,to}$=if $to==0$ then $intercom_{from}!iexit\rightarrow intercom_{from}?policy\rightarrow leaveS.from\rightarrow RM\_ITEM_{from}$ else $buff_{from,to}.out?msg\rightarrow RM\_eEXIT_{from,to-1}$.

改变主客体标记主要是通知系统更新标记状态信息.

2.2.3 安全服务器

SecServer 进程也是并发处理来自 RM 各个子进程的请求,其子进程 SS_ITEM_i 在信道 $intercom_i$ 上监听 RM_ITEM_i 的授权查询,通过信道 $intermgr_i$ 获得主客体状态信息,然后做出决策,更新系统状态和返回决策.

$$SecServer=(|||i:P@SS\_ITEM_i),$$

其中,

$$SS\_ITEM_i=SS\_SEND_i|SS\_RECV_i|SS\_WRITE_i|SS\_READ_i|SS\_CREATE_i|SS\_DEL_i|SS\_EXEC_i|SS\_EXIT_i|SS\_CHANGELABEL_i|CEP_i.$$

子进程 CEP_i 通过信道 cep_i 为其他授权查询服务子进程提供主体有约束的特定访问能力查询与判断服务.

$$CEP_i=cep_i?(S,I,E,j,op)\rightarrow CEPEXT_{i,S,I,E,j,op}$$

$CEPEXT_{i,S,I,remain,j,op}$=if $null(remain)$ then $cep_i!$false$\rightarrow CEP_i$ else if $head(remain).S\cap S=\varnothing\wedge head(remain).I\cap I=\varnothing\wedge head(remain).o=j\wedge head(remain).op=op$ then $cep_i!$true$\rightarrow CEP_i$ else $CEPEXT_{i,S,I,tail(remain),j,exec}$.

进程通信的控制点在接收方发出接受请求后,SS_SEND_i 将发送者发送时的标记信息返回给 RM,标记中去除了发送方完全可控的标签.SS_RECV_i 在发生 $api_i.(recv,j)$事件后,遵循定义 3(i)要求进行授权判断和实施污染策略.由于主体退出时系统会清空相应的缓冲区,使得如果缓冲区有消息则肯定是新鲜的.SS_RECV_i 通过信道 $intermgr_i$ 通知 LSM 调整好系统状态后才返回决策给 RM.

$$SS\_SEND_i=intercom_i.isend\rightarrow intermgr_{S_i}.getlabel?(S_i,I_i,C_i,E_i,exist_i)\rightarrow intercom?(S_i-C_i^{\pm},I_i-C_i^{\pm})\rightarrow SS\_ITEM_i,$$

$$SS\_RECV_i=intercom_i?(irecv,j,full,S_j,I_j)\rightarrow intermgr_{S_j}.getlabel?(S_{tmpt},I_{tmpt},C_j,E_j,exist_j)\rightarrow ermgr_{S_i}.getlabel?(S_i,I_i,C_i,E_i,exist_i)\rightarrow \text{if } exist_j\wedge S_j\subseteq S_i\cup C_i^+\wedge I_j\subseteq I_i\cup C_i^+\wedge full \text{ then } intermgr_{S_i}.setlabel!(S_i\cup S_j,I_i\cup I_j,C_i,1)\rightarrow intercom_i!accept\rightarrow SS\_ITEM_i \text{ else if } exist_j\wedge S_j\subseteq S_i\cup C_i^+\wedge I_j\subseteq I_i\cup C_i^+ \text{ then } intermgr_{S_i}.setlabel!(S_i\cup S_j,I_i\cup I_j,C_i,1)\rightarrow intercom_i!reject\rightarrow RECVEXT_{i,S_i,I_i,E_i,j} \text{ else } intermgr_{S_i}.setlabel!(S_i\cup C_i^+\cap DS,I_i\cup C_i^+\cap DI,C_i,1)\rightarrow intercom_i!reject\rightarrow RECVEXT_{i,S_i,I_i,E_i,j},$$

$$RECVEXT_{i,S_i,I_i,E_i,j}=cep!(S_i,I_i,E_i,j,recv)\rightarrow cep?result\rightarrow \text{if not } result \text{ then } intercom_i!reject\rightarrow SS\_ITEM_i \text{ else } intercom_i!accept\rightarrow SS\_ITEM_i.$$

进程 SS_EXEC_i 如果发现 $USER_i$ 执行客体 j 符合定义 3(vi)要求,则向 LSM 请求新主体的 pid.LSM 从可用主体标识符集合中随机选取,需要处理选取成功和失败两种情况.为了防止 *high* 进程和 *low* 进程利用可用主体标识符来构造隐蔽通道,执行结果不直接反馈给 $USER_i$.

$$SS\_EXEC_i = intercom_i?(iexec, j) \rightarrow intermgr_{O_j}.getlabel?(S_j, I_j, C_j, E_j, exist_j) \rightarrow$$
$$intermgr_{S_i}.getlabel?(S_i, I_i, C_i, E_i, exist_i) \rightarrow$$
$$\text{if } exist_j \wedge S_j \subseteq S_i \cup C_i^+ \wedge I_j \subseteq I_i \cup C_i^+ \wedge S_i - C_i^{\pm} \subseteq S_j \cup C_j^+ \wedge I_i - C_i^{\pm} \subseteq I_j \cup C_j^+$$
$$\text{then } intermgr_{S_i}.setlabel!(S_i \cup S_j, I_i \cup I_j, C_i, 1) \rightarrow$$
$$intermgr_{PID}.getpid \rightarrow intermgr_{PID}.choose?(result, newid) \rightarrow$$
$$\text{if } result \text{ then } intermgr_{S_j}.setlabel!(S_j \cup (S_i - C_i^{\pm}), I_j \cup (I_i - C_i^{\pm}), C_i, 1) \rightarrow$$
$$intercom_i!(accept, newid) \rightarrow SS\_ITEM_i \text{ else } intercom_i!(reject, 0) \rightarrow SS\_ITEM_i$$
$$\text{else if } exist_j \wedge S_j \subseteq S_i \cup C_i^+ \wedge I_j \subseteq I_i \cup C_i^+ \text{ then } intermgr_{S_i}.setlabel!(S_i \cup S_j, I_i \cup I_j, C_i, 1) \rightarrow$$
$$EXECEXT_{i,S_i,I_i,E_i,j} \text{ else } intermgr_{S_i}.setlabel!(S_i \cup C_i^+ \cap DS, I_i \cup C_i^+ \cap DI, C_i, 1) \rightarrow EXECEXT_{i,S_i,I_i,E_i,j},$$

$$EXECEXT_{i,S_i,I_i,E_i,j} = cep!(S_i, I_i, E_i, j, exec) \rightarrow cep?result \rightarrow \text{if not } result \text{ then } intercom_i!(reject, 0) \rightarrow$$
$$SS\_ITEM_i \text{ else } intermgr_{PID}.getpid \rightarrow intermgr_{PID}.choose?(result, newid) \rightarrow$$
$$\text{if } result \text{ then } intermgr_{S_j}.setlabel!(S_j, I_j, C_j, 1) \rightarrow intercom_i!(accept, newid) \rightarrow$$
$$SS\_ITEM_i \text{ else } intercom_i!(reject, 0).$$

读、写、创建和删除客体以及改变标记的决策过程与以上两个类似,创建客体时,客体以后执行该客体产生的新主体的能力由管理员设置客体属性来实现,这里通过信道 *nonceCE* 随机设置来模拟.

$$SS\_CREATE_i = intercom_i?(icreate, j, S'_j, I'_j) \rightarrow ... \rightarrow nonce.CE?(C'_j, E'_j) \rightarrow intermgr_{O_j}.setlabel!(S'_j, I'_j, C'_j, E'_j, 1) \rightarrow ...$$

主体退出时,需要更新 LSM 中标记信息和可用主体标识符集合.

$$SS\_EXIT_i = intercom_i?iexit \rightarrow intermgr_{S_i}.setlabel!(\{\cdot\}, \{\cdot\}, \{\cdot\}, 0) \rightarrow intermgr_{PID}.del!i \rightarrow intercom_i!accept \rightarrow SS\_ITEM_i.$$

2.2.4 标记状态服务器

LSM 进程维护着主体和客体的标记信息、能力信息和存在信息,子进程 $LSM\_S\_LIVE_{pid,s,i,c,exist}$ 提供了对主体 pid 的状态信息管理查询与修改服务,其状态变迁如下:

$$LSM\_S\_LIVE_{pid,s,i,c,e,exist} = intermgr_{S_{pid}}.getlabel!(s,i,c,e,exist) \rightarrow LSM\_S\_LIVE_{pid,s,i,c,e,exist} \mid$$
$$intermgr_{S_{pid}}.setlabel?(s',i',c',e',exist') \rightarrow LSM\_S\_LIVE_{pid,s',i',c',e',exist'}.$$

进程 LSM_S 则是所有 $LSM\_S\_LIVE_{pid,s,i,c,exist}$ 的并发,与前面相对应,初始化时,假设系统中只有一个启动进程,能力设置到最大,可以由它创建任意新的进程.

$$LSM\_S = (|||pid{:}P - \{1\}@LSM\_S\_LIVE_{pid,\{\cdot\},\{\cdot\},\{\cdot\},\{\cdot\},0})|||LSM\_S\_LIVE_{1,\{\cdot\},\{\cdot\},\{\cdot\},DS \cup DI \times \{+,-\},\{\cdot\},1}.$$

类似地,子进程 $LSM\_O\_LIVE_{oid,s,i,c,e,exist}$ 提供了对客体状态信息管理查询与修改服务,其状态变迁如下:

$$LSM\_O\_LIVE_{oid,s,i,c,e,exist} = intermgr_{O_{oid}}.getlabel!(s,i,c,e,exist) \rightarrow LSM\_O\_LIVE_{oid,s,i,c,e,exist} \mid$$
$$intermgr_{O_{oid}}.setlabel?(s',i',c',e',exist') \rightarrow LSM\_O\_LIVE_{oid,s',i',c',e',exist'}.$$

进程 LSM_O 是所有 $LSM\_O\_LIVE_{pid,s,i,c,exist}$ 的并发,客体初始状态信息由管理员维护,过程可被模型化为

$$LSM\_O = |||oid{:}O@nonce.SIC?(s,i,c,e,exist) \rightarrow LSM\_O\_live_{oid,s,i,c,e,exist}.$$

进程 $LSM\_ID\_MGR_{occupy,free}$ 维护着系统可用的主体标识符集合,用于创建主体时的分配和主体退出时的回收,其状态变迁如下:

$$LSM\_ID\_MGR_{occupy,free} = intermgr_{PID}getpid \rightarrow \text{if } empty(free) \text{ then } intermgr_{PID}.choose!(0,0) \rightarrow$$
$$LSM\_ID\_MGR_{occupy,free} \text{ else } nonce.Pid?i{:}free \rightarrow intermgr_{PID}.choose!(1,i) \rightarrow$$
$$LSM\_ID\_MGR_{occupy \cup \{i\},free-\{i\}} \mid intermgr_{PID}.del?i \rightarrow LSM\_ID\_MGR_{occupy-\{i\},free \cup \{i\}}.$$

LSM 的行为可描述如下:

$$LSM=LSM\_S|||LSM\_O|||LSM\_ID\_MGR_{\{1\},P-\{1\}}.$$

2.2.5 抽象的系统定义

基于以上组件,安全核可以被描述为 RM,SecServer 和 LSM 的并发:

$$KERNEL=(LSM \underset{intermgr}{\|} SecServer) \underset{intercom}{\|} RM.$$

由于整个系统是安全核和用户进程的并发,系统外在行为可以描述为

$$SYSTEM_{GTPM}=(USER \underset{api}{\|} KERNEL)\setminus aKERNEL,$$

其中,$aKERNEL=\{intercom,intermgr,buff,nonce,cep\}\cup aSYN$.

这里用到了屏蔽算子,$SYSTEM_{GTPM}$ 隐藏了系统内部状态变迁过程,如 RM 和 SecServer 之间的通信等,可见的只有用户进程和系统调用事件.实际上,一个系统的内部实现机制对用户进程应该是黑盒子,一个用户至多可以看到其他用户的行为,而不是内核的运行迹.

3 基于 GTPM 的操作系统无干扰性分析

本节将分析 $SYSTEM_{GTPM}$ 的安全性质.Goguen 等人提出的无干扰模型[18]是一种经典的形式化信息流模型,直观地说,所谓无干扰是指系统内一个用户行为不会影响到另一个用户所观察到的系统行为.无干扰模型形式化表述了更为纯粹的安全策略,更能反映安全本质,这类模型也被称为完美模型.通常认为,无干扰安全的系统不存在隐蔽通道.文献[19]采用有限状态机表示安全系统,给出了干扰关系会传递和干扰关系不会传递情况下系统无干扰安全的定义,后一情况模型化了存在保密性降级主体的系统.Ryan 等人使用 CSP 对无干扰模型进行了重新表述[17],将无干扰性质建立在 CSP 中进程等价[20,21]基础上.文献[22]研究了带降密的无干扰安全,而文献[8,23]将事件分类,定义了高级别、低级别和降密事件.我们也采用了类似的定义,文献[23]还修订了 CCS/SPA 进程代数的无干扰定义,文献[8]给出了可降密无干扰安全的公式,但是它没有采用标准的 CSP 进程等价验证模型定义,不利于对系统的自动化验证.首先描述 Ryan 的 CSP 无干扰定义[17],本文称其为基本的无干扰,再给出其在 $SYSTEM_{GTPM}$ 中的解释;然后描述我们提出的可降密的无干扰定义,最后证明抽象的 GTPM 系统是可降密无干扰安全的.

3.1 基本的无干扰

定义 5(基本的无干扰). 给定一个 CSP 进程 SYS,其事件集 $aSYS=aHIGH\cup aLOW$,且 $aHIGH\cap aLOW=\phi$,SYS 是基本的无干扰安全当且仅当

$$\left(SYS \underset{aHIGH}{\|} STOP\right)\setminus aHIGH \equiv SYS\setminus aHIGH.$$

基本的无干扰要求禁止 *high* 事件发生的系统和原系统之间在分别屏蔽所有 *high* 事件后是进程等价的关系.CSP 主要有 3 个进程等价验证模型,依次是:迹模型(trace model)、稳定失败模型 stable failures model 和失败/发散模型(failures/divergences model).其中:迹模型根据两个 CSP 进程的迹集合是否相同来判断它们是否等价;稳定失败模型根据两个 CSP 进程的迹集合和失败集是否分别相同来判断它们是否等价;失败/发散模型根据两个 CSP 进程的迹集合、失败集和发散迹是否分别相同来判断它们是否等价.3 个模型验证进程间等价性的程度和需要验证的状态空间个数都是递增的.迹模型可以验证安全性质[20,21],但不能区分不同的非确定性(non-deterministic)行为,主要是由于内部选择算子和外部选择算子虽然有相同的迹语义,但是它们的拒绝集语义并不同;失败稳定模型弥补了这一不足,可以增加验证活性(liveness)[20,21]属性,但是它不捕获不稳定状态.存在不稳定态的主要原因是,进程引入屏蔽算子后,进程表达式变成非监督表达式,使得进程会无限执行屏蔽事件,这一现象被称为发散;失败/发散模型被认为是 CSP 的标准模型,该模型同时考虑发散集和失败集情况,它可以增加验证活锁(livelock)[20,21]属性.为了避免因进程加入屏蔽算子导致的发散对无干扰分析带来的影响,Ryan 给出了一种改进的等式 $\left(SYS \underset{aHIGH}{\|} STOP\right)|||RUN_{aHIGH} \coloneqq SYS|||RUN_{aHIGH}$.显然,这又会使验证的状态空间大大增加.实际

上,如果不是实际的 SYS 存在发散迹,而是用于验证分析的 $SYS\backslash aHIGH$ 出现发散迹,则发散攻击是不存在的,不一定要求发散迹也等价.而且,迹模型可以对“不存在违背某个安全性质的反例”之类的断言进行验证[20,21].

分析 $SYSTEM_{GTPM}$ 的无干扰性,首先需要知道事件当事人的当前标记和事件的性质.而在 $SYSTEM_{GTPM}$ 中,主客体的标记和能力信息存储在 LSM 中,并且主体的标记动态变化,事件本身无状态信息.为此,分析前需要显示地为系统调用事件 $api_i.(\ldots)$ 标记上调用者的当前状态,将调用事件记为 $api_{i,S,I,C,E}.(\ldots)$,S,I,C,E 分别表示用户的保密性标记,完整性标记和标记调整能力和有约束的特定访问能力,用户 $USER_i$ 出生时就要获得状态信息.$USER_i$ 描述如下:

$$USER_i=born_i?(S,I,C,E,msg)\rightarrow USER\_LIVE_{i,S,I,C,E},$$

$$USER\_LIVE_{i,S,I,C,E}=SEND_{i,S,I,C,E}|RECV_{i,S,I,C,E}|WRITE_{i,S,I,C,E}|READ_{i,S,I,C,E}|CREATE_{i,S,I,C,E}|DEL_{i,S,I,C,E}|\ldots$$

根据模型要求,可能引发状态变化的系统调用有 $api_{i,S,I,C,E}.(recv,\ldots)$,$api_{i,S,I,C,E}.(read,\ldots)$,$api_{i,S,I,C,E}.(exec,\ldots)$和 $api_{i,S,I,C,E}.(changelabel,\ldots)$,需要及时更新用户状态.为此增加查询状态的系统调用 $api_{i,S,I,C,E}.(getlabel,\ldots)$,安全核也要增加对该请求的处理.用户进程中,涉及的子进程有 $RECV_{i,S,I,C,E}$,$READ_{i,S,I,C,E}$,$EXEC_{i,S,I,C,E}$ 和 $CHANGELABEL_{i,S,I,C,E}$.下面以 $RECV_{i,S,I,C,E}$ 为例:

$$RECV_{i,S,I,C,E}=api_{i,S,I,C,E}!(recv,from)\rightarrow api_{i,S,I,C,E}?(getlabel,S',I')\rightarrow api_{i,S',I',C,E}?(result,msg)\rightarrow USER\_LIVE_{i,S',I',C,E}.$$

其他子进程不需要更新状态,以 $SEND_{i,S,I,C}$ 为例:

$$SEND_{i,S,I,C,E}=api_{i,S,I,C,E}!(send,to,msg)\rightarrow\ldots\rightarrow USER\_LIVE_{i,S,I,C,E}.$$

规格调整后的 $SYSTEM_{GTPM}$ 记为 SYS_{GTPM},调整中,标记查询处理并不造成系统的安全状态变化,系统决策的依据还是安全核中的状态信息.这里显式标记和更新用户进程状态只是辅助验证,在系统实现时可以不考虑显示标记.SYS_{GTPM} 和 $SYSTEM_{GTPM}$ 在安全性上是等价的.

基于事件的标记信息,定义 6 对 SYS_{GTPM} 事件进行了分类.

定义 6(high 事件和 low 事件). 在 CSP 进程 SYS_{GTPM} 中,对于任意输出保护型标签 d:

- 定义 *high* 事件集:$aHIGH_d=\{api_{i,S,I,C,E}.(\ldots)|d\in S,i\in P\}\cup\{born_i.(S,I,C,E,\ldots)|d\in S,i\in P\}$,
- 定义 *low* 事件集:$aLOW_d=\{api_{i,S,I,C,E}.(\ldots)|d\notin S,i\in P\}\cup\{born_i.(S,I,C,E,\ldots)|d\notin S,i\in P\}$.

3.2 可降密的无干扰

基本的无干扰安全严格限制 *high* 事件干扰 *low* 进程,这可能没有反映出实际的安全需求.很多信息流模型允许有条件地从 *high* 进程到 *low* 进程的信息流动,但是需要对其管理,如通过可信中介完成或至少进行审计.在 GTPM 模型中,这些信息流动需要流经降密器(declassifer)以进行降密处理,降密器是有删除保密标签能力的主体或者是在指定状态下有特定访问能力的主体.降密器的行为可看做降密处理,这使得 GTPM 不满足定义 5.需要给出一种合理的无干扰安全定义,它允许 Declassifer 和 *low* 进程影响所有其他进程;同时,允许 *high* 进程影响其他 *high* 进程和 Declassifers,但是不允许 *high* 进程影响 *low* 进程.这是一种不传递的无干扰(intransitive noninterference)[8,19,22],考虑到 GTPM 模型的语义,本文称之为可降密的无干扰.

定义 7(可降密的无干扰). 给定一个 CSP 进程 SYS,其事件集 $aSYS=aHIGH\cup aLOW,aHIGH\cap aLOW=\varnothing$,$aMID\subseteq aSYS$,$SYS$ 是可降密的无干扰安全当且仅当

$$\left(SYS \underset{aHIGH-aMID}{\|} STOP\right)\backslash(aHIGH\cup aMID)\equiv SYS\backslash(aHIGH\cup aMID).$$

可降密的无干扰要求禁止 $aHIGH–aMID$ 中事件发生的系统 SYS' 和原系统之间在分别屏蔽所有 $aHIGH\cup aMID$ 事件后等价.也就是说,要求 $aMID$ 事件集对 *low* 进程的影响与 $aMID\cup aHIGH$ 事件集对 *low* 进程的影响相同,使得 *low* 进程无法推测降密器以外的 *high* 进程做了什么,*high* 不能试图通过降密器干扰 *low* 进程.如果 $aMID=\{\cdot\}$,定义 7 与定义 5 相同,此时不允许降密器存在.

分析 SYS_{GTPM} 的可降密无干扰性质,需要确定什么是 *mid* 事件.这里,降密器的降密行为是一个过程而不是一个独立事件,降密器改变自己的标记、发送信息和写客体都可能是降密过程的一个环节,为此,我们给出如下定义.

定义 8(mid 事件). 在 CSP 进程 SYS_{GTPM} 中,对于任意输出保护型标签 d,定义 *MID* 事件集

$$aMID_d=\{api_{i,S,I,C,E}.(\dots)|d\in C^{-},i\in P\}\cup\{born_i.(S,I,C,E,\dots)|d\in C^{-},i\in P\}\cup$$
$$\{api_{i,S,I,C,E}.(op,j,\dots)|i\in P,(S',I',op,j)\in E,S\cap S'=\varnothing,I\cap I'=\varnothing\}.$$

定义 8 允许 *aMID* 和 *aHIGH*(或 *aLOW*)之间存在交集,其实,*aMID* 包括降密器处于 *high* 状态和 *low* 状态下的所有行为事件集,这里面存在非降密事件,如降密器发信息给 *high* 进程和从 *low* 进程读信息等.不过在 SYS_{GTPM} 中,易知这些非降密事件并未导致系统安全状态变化(没有传播污点),也没有干扰 *low* 进程,不会影响到无干扰分析.

定理 1. 对于任意保密性标签 d,SYS_{GTPM} 是可降密无干扰安全.

证明:根据定义 7 和定义 8,即证以下等式成立:

$$\left(SYS_{GTPM}\underset{aHIGH_d-aMID_d}{\|}STOP\right)\setminus(aHIGH_d\cup aMID_d)\equiv SYS_{GTPM}\setminus(aHIGH_d\cup aMID_d).$$

在 SYS_{GTPM} 中没有用到内部选择算子,此时,进程之间的迹等价和拒绝集等价有相同的语义;另外,安全核 $KERNEL=\left(LSM\underset{intermgr}{\|}SecServer\right)\underset{intercom}{\|}RM$ 在接受到用户 *USERS* 系统调用请求时才会执行内部事件,处理后又继续等待,*KERNEL* 自身不会无限执行内部事件而使 SYS_{GTPM} 发散.综合以上考虑,我们采用迹模型验证进程间等价.

在 SYS_{GTPM} 中,主体使用有约束的特定访问能力的过程本身被看做降密过程.一方面,如果一个主体的特定访问能力约束为空,即该主体能否使用该能力与该主体的标记状态无关.也就是说,该主体可以在任何标记状态下使用该能力.这意味着包括 *high* 主体的所有其他主体无法干扰该主体对该能力的使用,*high* 主体也就不能通过干扰该主体对该能力的使用来干扰 *low* 主体;另一方面,如果一个主体的特定访问能力约束不为空,对于保密性标签 d 来说,则该主体运用该能力时,要求保密性标记不能包含 d.如果 *high* 主体可以干扰该主体的标记变化,则可以进一步干扰该主体对该能力的使用.然而根据模型规则,如果该主体的标记从$\{d\}$变为空,即使在它有删除自己标记中 d 能力的前提下,也只能由该主体自己显式地完成标记改变.*high* 主体无法干扰这一过程.如果主体的标记从空变为$\{d\}$,且该主体有添加 d 到自己标记中的能力,标记变化的一种可能是该主体自己通过显式改变标记完成,另一种可能是该主体尝试从 *high* 主体接收消息.根据模型规则,只要该主体试图接受,则不管 *high* 主体是否发消息,主体的标记一定变成$\{d\}$.这两种情况下,*high* 主体均构不成干扰.即 *high* 主体不能通过干扰该主体对有约束的特定能力的使用来干扰 *low* 主体.因此,下面的分析主要考虑主体标记调整能力的使用过程中是否出现了不被允许的信息流干扰.

在 SYS_{GTPM} 中,对于保密性标签 d,根据主体标记和标记调整能力的不同组合,主体有 8 中不同状态,分别是$(\{d\},\{\cdot\})$,$(\{d\},\{d^{+}\})$,$(\{d\},\{d^{-}\})$,$(\{d\},\{d^{+},d^{-}\})$,$(\{\cdot\},\{\cdot\})$,$(\{\cdot\},\{d^{+}\})$,$(\{\cdot\},\{d^{-}\})$,$(\{\cdot\},\{d^{+},d^{-}\})$.括号中前一项指出主体标记是否包含 d,后一项指出它对 d 的控制能力.在系统的某一时刻,如果两个主体状态相同,则它们对系统其他部分的影响力是相同的,其中一个主体能做到的影响,另一个主体也能做到.反过来,系统对两个主体也有相同的影响效果;至于它们之间的交互,由于双方同是污点或非污点主体,双方的标记不会因交互而变化,而能力又是不变化的,所以不会造成它们的状态变化.对于客体来说,状态除了包括标记以外,因为执行该客体时同样会引用该客体的能力属性,则客体也应有与主体相同的 8 种状态.如果两个客体状态相同,则系统对它们访问的返回效果也相同.

在 SYS_{GTPM} 中,设 P 是系统中可能的主体标识符 PID 集合,O 是系统中可能的客体标识符 PID 集合.令$|P|=8$,$|O|=8$,且初始时只有一个主体存在,设其状态为$(\{\cdot\},\{d^{+},d^{-}\})$,则它可以启动任何状态的主体和访问任何状态的客体,而且系统运行中可以最大同时出现 8 个不同状态的主体,假设此时等式成立.下面考虑$|P|=9$,其他情况不变时的情况,如果在新的条件下等式不成立,由于最大主体数目等于 8 时等式成立,则一定是系统同时出现 9 个主体以后的某个时刻出现干扰,并与第 9 个主体相关.而由前面的分析可知,系统的第 9 个主体出生后,它必定与某个存在的主体 x 的状态相同,而状态相同的主体对系统的影响力是一样的,系统对它们也有相同的影响效果,这

两个主体之间的交互也不影响系统状态.而 x 没有使系统出现干扰,第 9 个主体使系统出现干扰,这是矛盾的.因此,如果$|P|=8$ 时等式成立,则$|P|=9$ 时等式也成立,可以依次类推$|P|>9$ 的情况.我们可以用同样的方法分析$|O|=9$,其他情况不变的情况,如果系统同时出现 9 个客体,则必定有两个客体的状态相同,系统对它们访问的返回效果会相同.因此,如果$|O|=8$ 时等式成立,则$|O|=9$ 时等式成立,可以依此类推$|O|>9$ 的情况.

所以,如果我们能够证明在$|P|=8,|O|=8$ 而且初始时有一个状态为$(\{\cdot\},\{t^+,t^-\})$的主体时等式成立,则$|p|\geqslant 8,|O|\geqslant 8$,时等式也成立,从而证明了 SYS_{GTPM} 是可降密无干扰安全的.实际上,8 种不同状态中有些状态是等价的,可将状态压缩成 5 类:(1) $(\{d\},\{\cdot\}),(\{d\},\{d^+\})$;(2) $(\{d\},\{d^-\})$;(3) $(\{\cdot\},\{d^+,d^-\}),(\{d\},\{d^+,d^-\})$;(4) $(\{\cdot\},\{d^+\})$;(5) $(\{\cdot\},\{\cdot\}),(\{\cdot\},\{d^-\})$.例如在类(3)中两个该状态的主体发送和接受能力相同,这样的主体执行的所有事件都会被标注为 *mid* 事件,这样只需证明$|p|=5,|O|=5$ 时状态成立即可.

对于$|p|=5,|O|=5$,初始时有一个状态为$(\{\cdot\},\{t^+,t^-\})$的主体的 SYS_{GTPM},我们使用 CSP 的验证工具 FDR2 验证此时 SYS_{GTPM} 能否满足该等式.FDR[24]是一个面向有限状态机的模型检测工具,可以检测在系统的状态变迁过程中某些安全性质能否保持,FDR2 工具还通过状态压缩来提高验证效率.已知 $SYS_{GTPM}\backslash(aHIGH_d\cup aMID_d)$的迹集合至少包含 $(SYS_{GTPM} \underset{aHIGH_d-aMID_d}{\|} STOP)\backslash(aHIGH_d \cup aMID_d)$,因此验证等式成立,只需验证如下的 FDR 断言为真:

$$assert\ (SYS_{GTPM} \underset{aHIGH_d-aMID_d}{\|} STOP)\backslash(aHIGH_d \cup aMID_d)\ [T\ SYS_{GTPM}\backslash(aHIGH_d \cup aMID_d).$$

验证过程在环境 tool/FDR2.83,OS/Ubuntu 9.10,CPU/P4 2.8GHz,RAM/2GB 下进行,验证结果是断言为真,验证执行时间大约是 1680s.

基于上述分析和自动化验证结果,得知对于任意保密性标签 d,SYS_{GTPM} 是可降密无干扰安全.

以上分析了保密性保护中的无干扰,对于任意完整性标签 d,我们完全可以定义类似的完整性无干扰安全,同时将 *high* 事件定义标记包含完整性标签 d 的事件,将 *low* 事件定义为不包含 d 的事件,将污点程序看做 *high* 主体,非污点程序看做 *low* 主体,并采用相似的方法证明污点程序不能干扰非污点程序,从而不能影响非污点程序的完整性.

需要强调的是,由于我们的规格是在较高的层次上给出的,没有捕获低层次实现细节,如 CPU、Cache、内存、硬盘和网络等行为,因此证明的结论难以反证不存在模型化这些硬件实现细节的隐蔽通道,实现时遵守这些规格,系统只是有了获得无干扰安全性质的基础和可能,下一步需要继续细节化规格,以使验证结果反映更加实际的系统的性质.

4 示例分析及可用性比较

在本节中,我们通过一个典型的应用示例来分析说明 GTPM 在可用性方面比传统的信息流模型有明显的提高.

(1) 系统环境

图 2 描述了一个较常见的桌面系统组成,其中:IM 是用户的即时通信软件,如各种聊天工具.IM 会将可能涉及用户隐私的信息放到 IM data 中;Office 是一套办公软件,用户用它来处理办公文档 Office files,其中部分文档涉及公司秘密;PGP 是一款可以提供邮件加密功能的安全电子邮件软件,用户用它与自己的公司同事进行安全的办公文件传输;Norton Antivirus 是一款杀毒软件,通过杀毒,可以有效、及时地清除系统中的病毒和木马程序,但是它需要定时更新病毒库;OS Updated packages 是用户从网上下载的系统更新安装包,用于操作系统更新;Explorer 是操作系统与用户之间的交互接口程序;Firefox 是一款户用来访问万维网的浏览器.另外,用 Download data 表示 Firefox 在网络通信时下载的临时文件;OS config files 存储了操作系统运行所需的重要配置信息.

(2) 安全需求

① IM 软件不能窃取秘密的 Office files,也不能破坏 office files 和 OS config files 的完整性.这是考虑到有些即时通信软件不可信或者因有漏洞而存在被利用可能,从而变成间谍或病毒程序;也不允许 Office 软件窃取

IM 的数据,但允许 IM 的网络通信.

② Norton Antivirus 可以检测检测 IM data,Office files 这些包含个人隐私或商业秘密的数据中是否含有病毒,但是不能泄露这些信息到网络.即使 Norton Antivirus 不可信或有漏洞也不能泄露这些信息,但是允许它在线的病毒库更新.

③ 秘密的办公文档 Office files 可通过 PGP 软件加密后发送出去,PGP 收到网上发来的邮件经过杀毒后可被 Office process 读取.

④ 允许 Explorer 修改 OS config file,但是当 Explorer 加载了从网上下载的 Download data 时,应禁止 Explorer process 修改 OS config file.这是由于如果 Explorer 或者浏览器存在漏洞,导致 Exlprer 动态加载不可信的动态链接库,则存在系统被劫持的可能.这也是比较常见的一种网络攻击方式.

⑤ 从网上下载的 OS Updated packages 需经过查毒,确保无病毒后才能允许其运行时修改 OS config file,从而成功的更新操作系统.

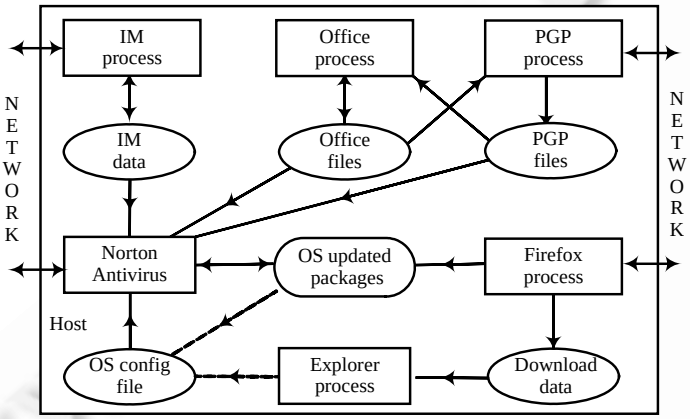

Fig.2 A typical application scenario of a desktop operating system

图 2 一个典型的桌面操作系统应用场景

(3) 已有其他主要模型的可用性分析

(i) 安全级范围模型

该模型目前未发现隐蔽通道,它是 BLP 模型的改进.在该模型中,主体的当前安全级是一个二元组(α-min, v-max),用 L_1 dom L_2 表示标记 L_1 大于标记 L_2,用 L_1 domrel L_2 表示 L_1 dom L_2 或者 L_2 dom L_1.规定主体 S 可以读客体 O,当且仅当 v-max(S) dom *level*(O),且 α-min(S) domrel *level*(O);规定主体 S 可以写客体 O,当且仅当 *level*(O) dom α-min(S)且 v-max(S) domrel *level*(O).这使得主体对强制访问控制策略的超越被限制在一个范围内.但是在该方法中,主体的权限在其生命周期内实际并没有发生变化,与访问历史和系统状态无关.

安全需求②允许 Norton Antivirus 读秘密文档、读/写非秘密文档和网络,但是当它读取秘密文档后,系统应该立刻回收该进程对非秘密文档和网络的写权限.以后所有该进程写操作的对象都应该升级到秘密类型,当进程重新启动(reinitialized)后,可恢复其对非秘密文档和网络的写权限,这是一种要求随着主体访问历史变化而调整主体权限的策略.安全级范围方法要么允许 Norton Antivirus 读秘密文档,要么不允许 Norton Antivirus 读秘密文档,它难以表达安全需求②要求的策略,该策略从某种程度上体现了保密性最小权限原则.

安全需求④允许 Explorer process 修改 OS config file,但是当它动态加载从网上下载的低可信度的动态库或组件时,系统应该立刻回收 Explorer process 对 OS config file 的修改权限.这种对主体权限随历史访问而变化的要求,也是安全级范围模型无法用策略表达的.该需求要求的策略从某种程度上体现了完整性最小权限原则.

安全需求②解决不好,存在秘密泄露或者无法正常杀毒的问题;安全需求④解决不好,存在无法访问万维网或者系统感染病毒的问题.而安全级范围方法不能制定出有效的策略来满足这些安全需求.

(ii) Histar/Flume 系统

已有的未发现隐蔽通道的 Histar 或 Flume 系统由于可以表达主体的实际权限变化的策略,可满足以上安全需求.由于篇幅的关系,我们只分析它们在可用性方面存在的不足.由于 Histar 或 Flume 系统均不是污点传播模型,不支持标记的隐式调整,需要主体自己主动地调整自己的标记,这会导致以下不足:

(a) 由于进程的标记调整包括各种升级和降级可能,而且为了防止隐蔽通道,进程不能获得通信对方或者访问客体的标记信息,进程自己未必能做出及时和正确选择,最终可能还是需要系统的使用者完成.这意味着用户在系统的使用过程中需要经常地手动调整主体的标记,这不仅让用户体验不好,而且也需要用户有相关的专业知识才行;

(b) 每个应用程序需要增加标记调整功能,这需要改动程序和应用软件厂商的合作,操作起来有难度;

(c) 显示标记调整需要系统调用来完成,这种频繁的调用会增加系统开销.

(4) GTPM 的可用性分析

构建 GTPM 的策略是:为保护 IM data 和 Office data 的保密性,分别创建保密性标签 ds_{IM} 和 ds_{office},以对这些保密信息扩散的流向进行跟踪控制.为防止 IM process 破坏系统内其他部分的完整性,建立完整性(污点)标签 di_{IM},对它的写操作影响到的实体进行跟踪控制.网络被看成完整级和保密级均最低的客体,对它流到系统内的信息进行完整性(污点)跟踪.各个实体标记初始设置见表 1,其中,$\perp_{DI}$,$\perp_{DI}^{+}$和$\top_{DS}^{+}$的定义见第 2.1 节.

Table 1 GTPM system Settings of the initial labels and capabilities of the subjects and objects in the scenario mentioned above

表 1 应用示例中各个实体在 GTPM 系统中的能力和初始标记设置

Subjects	Secrecy label	Integrity label	Capabilities	Objects	Secrecy label	Integrity label
IM process	$\{\cdot\}$	$\{di_{IM}\}$	$\{ds_{IM}^{\pm}\}$	IM data	$\{ds_{IM}\}$	$\{di_{IM}\}$
Office process	$\{\cdot\}$	$\{\cdot\}$	$\{ds_{office}^{\pm}\}$	Office file	$\{ds_{office}\}$	$\{\cdot\}$
PGP process	$\{\cdot\}$	$\{\cdot\}$	$\{ds_{office}^{\pm},\perp_{I}^{+}\}$	PGP data	$\{\cdot\}$	$\{\cdot\}$
Norton antivirus	$\{\cdot\}$	$\{\cdot\}$	$\{\top_{DS}^{+},工_{DI}^{\pm}\}$	OS config files	$\{\cdot\}$	$\{\cdot\}$
Firefox process	$\{\cdot\}$	$\{\cdot\}$	$\{\perp_{DI}^{+}\}$	Download data	$\{\cdot\}$	$\{\perp_{DI}\}$
Explorer process	$\{\cdot\}$	$\{\cdot\}$	$\{\perp_{DI}^{+}\}$	OS updated packages	$\{\cdot\}$	$\{\perp_{DI}\}$

在表 1 中,由于 IM process 是隐私信息 IM data 的拥有者,将其能力设置成 $\{ds_{IM}^{\pm}\}$,表示它可以传播这些数据.相似地,Office process 的能力设置成 $\{ds_{office}^{\pm}\}$.PGP process 能力被设置为 $\{ds_{office}^{\pm},\perp_{I}^{+}\}$,前一个能力使它可以读取 Office files 并将它们加密后发送,后一个能力使它可以在线收取邮件.Norton Antivirus 的能力被设置为$\{\top_{DS}^{+},工_{DI}^{\pm}\}$,表示它可以读取任何秘密信息,但是没有降密能力而不能传播这些秘密信息;而且它可以被任何完整性标签污染以用于扫描任何实体,也可以在完成扫描和查杀病毒后从自己标记中去掉任何完整性标签,以提高完整级.Firefox 和 Explorer process 的能力被设置为 $\{\perp_{I}^{+}\}$,但是它们均没有被赋予去除完整性标签(污点)的能力.

对于安全需求①的可满足性:因为 IM process 的标记被添加了污点标签 di_{IM},而域 di_{IM} 中实体对 office files 和 OS config files 写访问均未被授权,则 IM process 不能破坏这些数据的完整性(定义 3(iii)).因为 IM process 的保密性标签中没有$\{ds_{office}\}$,它没有读权限访问秘密的 Office files(定义 3(ii)).因为 IM process 通过为自己创建的 IM data 数据添加保密性标签防止这些信息的扩散,保证了 IM data 的保密性(定义 3(ii)).最后,根据它的能力和可能的标记状态以及网络被看成完整级和保密级均最低的客体,知道它可以读写网络(定义 3(ii)、定义 3(iii)).

对于安全需求②的可满足性:因为 Norton Antivirus 的标记为空,所以平时允许该进程主动连接网络以更新程序(定义 3(iii)).但是如果它读取秘密的 IM data,Office files 信息(能力$\{\top_{DS}^{+},工_{DI}^{\pm}\}$允许这种读行为)后,由于它没有降密能力,它因密级升高也就失去了对网络的输出而不能将这些信息传播出去(定义 3(iii));而且以后写的对象都感染了对应的保密性标签,用户可以通过重新启动 Norton Antivirus 程序,恢复其对非秘密文档和网络的写权限,从而更新病毒库.

对于安全需求③的可满足性:因为 PGP 对 ds_{office} 有升密和降密能力,它可以读取秘密的 Office files,在对内容用密码算法加密后,可以通过网络发给接收方(定义 3(iii)).PGP 从网上收到的文件 PGP files 带有完整性标记 $\perp_{DI}$,,经过 Norton Antivirus 杀毒,Norton Antivirus 可为其显式去除污点.令其完整性标记为空(定义 4),可以被 office process 读取.

对于安全需求④的可满足性:因为 Explorer 的初始标记支配 OS config file,所以允许 Explorer 在未动态加载(读)Download data 之前修改 OS config file,但是在 Shell 动态加载(读)Download 加载后,因为它没有去污能力,也就失去了这种修改权限(见定义 3(iii)).

对于安全需求⑤的可满足性:从网上下载的 OS Updated packages 的完整性标记为 $\perp_{DI}$,如果直接运行安装,则由于感染了完整性标签 di_{net},则无法修改 OS config file(见定义 3(iii)).当 Antivirus Scanner 查毒并确认其无毒后,显式去除 OS Updated packages 标记中的标签 di_{net}(见定义 4),OS Updated packages 就可以在安装时修改 OS config file.

可以看出,GTPM 系统可以较好地满足这些安全需求.同时,由于它属于污点传播模型,支持标记隐式变化,系统根据访问需求合理地自动调整主客体标记,另外也支持标记的显示变化,使系统有较大的灵活性,提高了可用性.

5 结束语

常见的污点传播模型只在信息流动时传播污点,本文提出了广义污点传播模型拓展了污点传播语义:一是通过分析信息流动态势,增加了特殊情况下的污点传播;二是允许主体通过显式改变自身标记以实现自污染.两种方法都试图防止污点传播中的已知隐蔽通道.另外,通常污点传播会导致污点积累和主体的发送信息能力越来越弱,GTPM 通过赋予主体删除自身标记中标签的权限来体现主体降密或去污能力,以维持主体发送能力.

在分析模型安全性时,本文用 CSP 语言建模 CTPM 系统,并给出了主体、引用监视器、安全服务器和标记状态管理器的行为的形式化语义,且在已有的无干扰理论基础上定义了基于 CSP 标准模型的可降密无干扰安全,证明了抽象的 GTPM 系统具有可降密无干扰安全性质.这种验证方法具有一定的扩展性和通用性,可以分析验证实际操作系统的安全性.最后,通过一个示例分析了模型的可用性.

下一步研究将考虑将主体的能力大小与主体的行为是否安全可信结合起来,对主体能力的使用进行监管以完善模型;并在保证安全性的前提下,研究如何进一步提高模型的可用性.

References:

[1] Bell DE. Security policy modeling for the next-generation packet switch. In: Proc. of the IEEE Symp. on Security and Privacy. IEEE Computer Society Press, 1988. 212−216. [doi: 10.1109/SECPRI.1988.8113]

[2] Biba KJ. Integrity consideration for secure computer systems. Technical Report, MTR-3153, Bedford: MITREE Corporation, 1977.

[3] Wu YJ, Liang HL, Zhao C. A multi-level security model with least privilege support for trusted subject. Journal of Software, 2007, 18(3):730−738 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/730.htm [doi: 10.1360/jos180730]

[4] Newsome J, Song D. Dynamic Taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proc. of the 12th Annual Network and Distributed System Security Symp. 2005.

[5] Van De Bogart S, Efstathopoulos P, Kohler E, Krohn M, Frey C, Ziegler D, Kaashoek F, Morris R, Mazieres D. Labels and event processes in the asbestos operating system. ACM Trans. on Computer Systems, 2007,25(4):11:1−11:43. [doi: 10.1145/1314299.1314302]

[6] Zeldovich N, Boyd-Wickizer S, Kohler E, Mazieres D. Making information flow explicit in HiStar. In: Proc. of the Usenix Association 7th Usenix Symp. on Operating Systems Design and Implementation (OSDI 2006). Seattle: USENIX Association, 2006. 263−278.

[7] Krohn M, Yip A, Brodsky M, Cliffer N, Kaashoek MF, Kohler E, Morris R. Information flow control for standard OS abstractions. In: Proc. of the 21st ACM Symp. on Operating Systems Principles (SOSP 2007). Stevenson: ACM Press, 2007. 321−334. [doi: 10.1145/1294261.1294293]

[8] Krohn M, Tromer E. Noninterference for a practical DIFC-based operating system. In: Proc. of the 2009 IEEE Symp. on Security and Privacy. Berkeley: IEEE Computer Society, 2009. 61−76. [doi: 10.1109/SP.2009.23]

[9] Landwehr CE. Formal models for computer security. ACM Computing Surveys, 1981,13(3):247−278. [doi: 10.1145/356850.356852]
[10] Fraser T. LOMAC: Low water-mark integrity protection for COTS environments. In: Proc. of the 2000 IEEE Symp. on Security and Privacy. Berkeley: IEEE Computer Society, 2000. 230−245. [doi: 10.1109/SECPRI.2000.848460]
[11] McIlroy MD, Reeds JA. Multilevel security in the UNIX tradition. Software-Practice and Experience, 1992,22(8):673−694. [doi: 10.1002/spe.4380220805]
[12] Li NH, Mao ZQ, Chen H. Usable mandatory integrity protection for operating systems. In: Proc. of the 2007 IEEE Symp. on Security and Privacy. Berkeley: IEEE Computer Society, 2007. 164−178. [doi: 10.1109/SP.2007.37]
[13] Sun WQ, Sekar R, Poothia G, Poothia G, Karandikar T. Practical proactive integrity preservation: A basis for malware defense. In: Proc. of the 2008 IEEE Symp. on Security and Privacy. Oakland: IEEE Computer Society, 2008. 248−362. [doi: 10.1109/SP.2008.35]
[14] Mao ZQ, Li NH, Chen H, Jiang XX. Trojan horse resistant discretionary access control. In: Proc. of the 14th ACM Symp. on Access Control Models and Technologies (SACMAT 2009). Stresa: ACM Press, 2009. 237−248. [doi: 10.1145/1542207.1542244]
[15] Myers AC, Liskov B. Protecting privacy using the decentralized label model. ACM Trans. on Software Engineering and Methodology, 2000,9(4):410−442. [doi: 10.1145/363516.363526]
[16] Hoare CAR. Communicating Sequential Processes. Englewood Cliffs: Prentice Hall International, 1985.
[17] Ryan PA, Schneider SA. Process algebra and non-interference. Journal of Computer Security, 2001,9(1-2):75−103. [doi: 10.1109/CSFW.1999.779775]
[18] Goguen JA, Meseguer J. Security policies and security models. In: Proc. of the 1982 IEEE Symp. on Security and Privacy. IEEE Computer Society Press, 1982. 11−20. [doi: 10.1109/SP.1982.10014]
[19] Rushby J. Noninterference, transitivity, and channel-control security policies. Technical Report, CSL-92-02, Menlo Park: Stanford Research Institute, 1992.
[20] Roscoe AW, Hoare CAR, Bird R. The Theory and Practice of Concurrency. London: Prentice Hall, 1997.
[21] Schneider S. Concurrent and Real-Time Systems: The CSP Approach. Chichester: John Wiley & Sons, LTD, 2000.
[22] Roscoe AW, Goldsmith MH. What is intransitive noninterference? In: Proc. of the 12th IEEE Computer Security Foundations Workshop (CSFW'99). Mordano: IEEE Computer Society, 1999. 228−238. [doi: 10.1109/CSFW.1999.779776]
[23] Bossi A, Piazza C, Rossi S. Modelling downgrading in information flow security. In: Proc. of the 17th IEEE Computer Security Foundations Workshop (CSFW 2004). Pacific Grove: IEEE Computer Society, 2004. 187−201. [doi: 10.1109/CSFW.2004.14]
[24] Formal Systems (Europe) Ltd. Failures divergences refinement: FDR2 manual. 2005. http://www.fsel.com

附中文参考文献:

[3] 武延军,梁洪亮,赵琛.一个支持可信主体特权最小化的多级安全模型.软件学报,2007,18(3):730−738. http://www.jos.org.cn/1000-9825/18/730.htm [doi: 10.1360/jos180730]

杨智(1975－),男,河南开封人,讲师,主要研究领域为访问控制模型,系统安全和软件.

吴金宇(1984－),男,博士生,主要研究领域为网络与信息安全,风险评估.

殷丽华(1973－),女,博士,副研究员,主要研究领域为信息内容安全,安全属性计算.

金舒原(1974－),女,博士,副研究员,主要研究领域为网络安全.

段洣毅(1953－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络通信,信息安全.

郭莉(1969－),女,博士,研究员,CCF 高级会员,主要研究领域为信息内容安全.