基于DTE技术的完整性保护形式模型*

季庆光** 卿斯汉 贺也平

(中国科学院软件研究所信息安全技术工程研究中心, 北京 100080)

摘要 为实现结构化保护级操作系统的完整性保护, 本文在深入分析完整性策略的结构和涵义的基础上, 提出了基于DTE技术的完整性保护的形式模型; 该模型由两部分组成: 配置DTE的基本规则和状态迁移模型, 前者解决如何设置相应的域和型, 后者解决在系统变迁过程中如何维持初始设置获得的安全不变量. 模型设置了10个不变量, 特别提出了新的处理信息流的不变量, 并探讨了与文献中相关不变量的关系. 以可操作性很强的方式描述了13个具有良好原子性的迁移规则. 随后, 证明了相应的安全基本定理. 在深入分析文献中现有模型与本模型异同的基础上, 探讨了不变量设置的合理性. 最后, 指出未来要完成的工作, 特别, 用本模型去分析SELinux的安全性是可能的.

关键词 完整性策略 形式模型 信息流 域 类型

在文献[1]中, Jueneman表达了系统安全设计与分析业界的一个共识: 桔皮书TCSEC几乎是排它性地关注防止信息的泄露, 即防止未授权用户获取有效信息, 提到的系统完整性是仅就系统功能的维护而言的; 随着信息系统互联程度的不断加强, 桔皮书不关心信息的破坏、蓄意修改及服务的未授权使用的不足已越来越为人们所意识到并很快展开富有成效的工作. 1989年美国国家标准局召开完整性策略研讨会[2], 对桔皮书的网络解释已明确提出要保护通信完整性; 1991年美国国家计算机安全中心发表"面向完整性的对象控制: 对TCSEC的修订本提案"[3]. 1999年发表的CC标准[4]的安全功能要求部分, 其中的FIA, FTP和FCO功能类直接用于完整性保护, 而且还有不少功能类都有与完整性保护有关的功能族, 例如FDP-SDI, FDP-UIT; FPT-ITI, FPT-TDC, FPT-TRC等; 虽然在信息系统中实现完整性保护的重要性已获得共识, 但是人们对完整性概念及如何实现完整性的认识却经历了一个痛苦而漫长的过程, 即使现在也还是一个有价值的研

2004-12-22收稿, 2005-04-01收修改稿

* 国家重点基础研究发展规划(G1999035802)和国家自然科学基金(60083007)资助项目

** E-mail: qgji@ercist.iscas.ac.cn

究领域，正如在1995年著名的系统安全专家Abrams所说：“最近几年所完成的关于完整性的工作，少得令人吃惊；就我们所知：目前尚未见到关于Clark-Wilson完整性实现的报道；完整性的精确定义仍是一个琢磨不透的东西”[5].

为深化对安全策略的理解，Bell在文献[6]中把它划分为四个层次：自然语言组织表达层次(organizational)，概念化表达层次(conceptual)，抽象设计层次(abstract-design)及实现设计层次(implemented-design). 在这个划分框架下，结合Mayfield等在文献[7]中的工作，可以对完整性策略及其研究现状作出较深入的分析.

在自然语言组织表达层次上，完整性策略现在已是相当清楚了，它包括数据完整性及系统完整性两部分，其目标是阻止非授权用户对保护对象的修改行为；阻止授权用户对保护对象的不恰当的修改行为；及保持保护对象的内在及外在的一致性. Sandhu[8]所说的5种定义，实际上是该概念应用于不同对象或环境时的不同表达.

概念化表达层次的策略是使用术语对自然语言组织表达层次策略进行抽象的、概念化的部分表达，所以这个层次的策略必须在适当的抽象层次上利用概念化的术语描述上一层次的策略，并保持与实现的抽象环境无关的特性，提出有较好原子性的、恰当的、抽象的、概念化的术语，这些术语是精确表达策略可操作性的语言组分；对这个层次上的完整性策略，人们提出行为模式编码(code of behavior)的概念，把某种行为模式编码附加到策略的主动实体上，以保证它们的行为不超出限定的范围，这个概念在不同的模型中有不同的叫法，但它们的实质作用是相同的；一体化(wholeness)的概念则是确保策略的被实施的实体的完备性(completeness)，这个概念有时也被具体化为完整性检验过程及可信过程；另一个有用的概念是约化风险(rish reduction)，它确保完整性策略失效时其危害性极小化. 进一步细化这些概念，人们提出大量的完整性原理，它们深化了对完整性策略语义内涵的理解，同时也揭示了实现完整性策略的基本原则.

在抽象设计层次上，必须把上一层次的概念抽象地细化为可执行性和可操作性更强的策略族，按LaPadula[9]对模型设计的五个层次的划分，这一层次的策略应该属于前两个模型层次描述的主要对象，即可信对象层次及外在模型层次描述的主要对象. 根据RFC3060，我们认为这个层次上的策略必须考虑：策略条件；策略目标；策略动作；策略规则；策略角色；及策略冲突等. 对这个层次上的完整性策略，人们提出了大量的完整性机制及抽象模型，这些机制及其相关的模型都是为实现各种完整性原理而设计的；其中抽象模型有：Biba模型，Goguen-Meseguer模型，Sutherland模型，Clark-Wilson模型，Brewer-Nash模型等. 在这些模型中，Biba提出了完整性级及完整性范畴的概念；Goguen-Meseguer提出了互不干扰性(noninterference)；Sutherland提出了互不可推理性(nondeducibility)；Clark-Wilson提出了良构事务(well-formed transactions)；Brewer-Nash提出了中国墙(Chinese Wall)的概念. 这些概念为形式地刻画完整性机制奠定了基础.

策略设计的最后一个层次是实现设计，这时候必须把抽象设计与具体系统有机结合起来．这一层次的策略应该属于 LaPadula 对模型设计的五个层次的划分中后三个模型层次描述的主要对象，即内在模型，操作规则，及功能设计．对于完整性策略的设计，目前大量的研究工作都集中在这一层次上．在论文级已完成的工作包括：Lipner 实现，Boebert-Kain 实现，Lee-Shockley 实现，Karger 实现，Gong 实现及 Jueneman 实现．其中 Boebert-Kain 实现使用了多域技术 (multi-domain), 也常称域型强制执行技术 (domain and type enforcement), 据 Mayfield 等在文献 [7] p.91 中说 (1991 年): 这是在实际系统中真正被实现的仅有的模型实现．

对于完整性策略在实际系统中的实现，目前，我们认为工业界主要使用的完整性保护抽象模型还是 Biba 模型和类似于 Clark-Wilson 模型的模型，1999 年 IBM 的卡操作系统安全内核的模型就是一个改进的 Biba 模型 [10]; 2000 年 Fraser[11] 在 Linux 上实现了一个完整性保护原型 Lomac, 主要基于改进的 Biba 低标模型; Ott[12] 在 Linux 上实现的基于任务的隐私策略模型，在很大程度上与 Clark-Wilson 模型是可比的．Sidewinder[13] 的完整性保护机制类似于 Clark-Wilson 模型．

如何有效地实现完整性保护，正如前面所指出的，是一个困难的问题，因为一方面，尽管 Clark-Wilson 模型相比之下是目前最能反映完整性保护需求的模型，但在操作系统中完整地实现它是困难的，甚至是不可能的；另一方面，域型强制执行技术作为实现完整性策略的有效技术，在目前的文献 (例如：文献 [14]) 中主要是探讨 DTE 技术的实施原理以及针对某些特定攻击和小的具体应用实例的 DTE 配置方法 [15,16]; 尽管著名的 SELinux, DTE-Linux 及 SAIC-DTE 都使用 DTE 来实现系统的安全配置，但是没有文献说明或者探讨过这些内核系统实现的具体安全目标及如何达到这些目标；也没有探讨过在系统中为什么要配置这些域，这些型以及相应的域与域之间的关系和域与型之间的关系，特别，改动策略库后，例如：添加 (或者删除) 某个型或者域，不危及系统的安全性的理论依据是什么？目前文献中没有发现一个用 DTE 技术实现的安全系统，它具有类似 Bell-LaPadula 模型那样的形式模型．为了获得高可靠性的安全系统，CC 标准 [4] 要求系统必须给出形式化策略模型，因此提出一个基于 DTE 技术的实现完整性保护的形式模型是有意义的．

1 基于 DTE 实现完整性保护的形式模型

1.1 在操作系统中完整性保护面临的问题

在文献 [1] 中，Jueneman 指出对完整性的威胁主要来自三方面：程序或数据在 TCB 之外存储或传送信息时被外部攻击者进行主动修改；程序或数据在存储或传送过程中，被硬件随机的错误或程序的 bugs 的偶然性修改；数据被 Trojan horse 或病毒的恶意修改．我们关心的主要是来自第一方面及第三方面的威胁；修改数据有两层含义：一是直接的主动修改，在这种情况下它是通过主动进程直接的写操作来完成修改的；二是间接的被动修改，它是这种类型数据的合法拥有

者，由于它对其他类型数据的信任，通过读取这些数据而变成恶意代码的受害者，在这种情况下修改是通过读取恶意代码，由恶意代码完成修改的．这里恶意代码的目的与Boebert等在文献[17]中详细描述的Trojan horse的目的完全不同，前者是为破坏系统，使系统瘫痪；而后者则只是为获取非法的信息，本身对系统并不造成从文献[17]的深刻分析，可以说该文已证明了Bell-LaPadula模型的$*$-性质能有效阻止Trojan horse造成的信息泄露；也正是从他们的分析中可以看出：$*$-性质只是阻止Trojan horse有效工作，并不阻止Trojan horse进入高安全级的环境中，因此Bell-LaPadula模型并不能抗病毒攻击，这已被Cohen[18]用实验证实．由此可以看出完整性保护与机密性保护的不同．

1.2　模型实现完整性保护的原理及相关术语

在我们的模型中，为实现完整性保护，被保护数据的完整性是通过类似于Lee[19]的完整性范畴(integrity category)来描述的，在DTE技术中，叫做型(type)，一个完整性范畴(或者型)就是被保护数据的一种特殊的类型名．为了阻止授权用户对保护对象的不恰当的修改，基本思想是：限制替代用户的主动进程的活动范围，这个活动范围在DTE技术中，叫做域(domain)，使主动进程获得它应该完成目标以外的能力极小化；也就是说，为了完成被授权的任务，每个主动进程仅仅被提供完成这个任务所需的极小特权集，这正是极小特权原理对主动进程的应用；为了让同一进程进入不同的域，DTE技术必须规范域之间的关系；通过设置使不同域的处理对象完全不同来实现的域隔离是不现实的，本模型将采用文献中的方法[20],1)，在DTE技术中引入良构事务，使域与域之间实现受到限制的信息交换．为了保持保护对象的内在的一致性，本模型描述的多域策略将通过增加策略执行的保障规则来实现，也就是说，通过保障规则确保保护对象的内在的一致性得到检验．

本模型将使用如下术语：

• 域(domain)：一个域就是进程可以活动的抽象逻辑空间，一个进程一旦进入该域，它的活动空间就被限定在这个域内，因此它从逻辑上界定了一个进程受到限制的执行环境．它由三部分组成：1) 通过路径名来标识的入口守护进程(这些入口守护进程可能有多个)，它是进入一个域的惟一通道．2) 域的管辖范围及权限，它标明了在这个域中进程能执行的特权程序及访问的各种对象，同时也指明了访问权限．3) 该域的关系域列表及相关关系，它指出了与它有关联的所有域及对这些域中进程的访问权限，例如：该域的进程可以向它的某些关系域中的进程发软中断信号；该域控制了某些关系域，该域的进程可以自动激活它的某些关系域入口守护进程，进入这些域中；完成在该域中的任务后，该域中的进程有能力通过执行某些关系域的入口守护进程来创建在这些关系域中的新进程．

1) Nyanchama M. Commercial integrity, roles and object orientation. Ph D Dissertation. The University of Western Ontario, London, Ontario, 1994

• 类型 (type): 它是被保护数据对象的一种特殊属性，对于某一类型，如果没有明确的授权，任何主体都不能修改或者创建具有该类型的数据. 一个类型也可视为完整性属性的等价类，在这个等价类里的数据均具有相同的完整性保护的要求. 完整性属性具有原子属性，因此被保护数据对象不能同时分属不同的类型.

• 完整性检验过程 IVP: 当主体试图对受完整性保护的数据进行操作时，对主体的授权及身份进行检验的过程.

• 良构事务 WFT: 它是一个被确证能执行特殊操作的程序，该操作能为对象保持或者引入完整性属性. 它主要用于防止完整性 Trojan horse 对系统的按 Thomsen 的定义，完整性 Trojan horse 是一种恶意代码，它不关心敏感信息的泄露，它的目的在于修改或者伪造信息.

1.3 组织级安全策略

本模型的目的就是通过域隔离，确保对系统的完整性破坏的极小化. 其策略是：

1. 根据系统的非管理用户的情况，定义非管理用户域；这些域的多少与系统面向的具体服务相关. 如果操作系统是多个应用系统的支持平台，那么使用不同的应用系统应该进入不同的域；Lipner[21] 把用户分为常规用户、应用开发程序员及系统开发程序员，它们应该进入不同的域.

2. 根据系统的管理用户的设置情况，定义管理用户域，这些域在功能上完全不同于非管理用户域，因此这些域应设为与非管理用户域不同的域；Lipner 把管理用户分为系统控制人员、系统管理员及系统审计员，它们应该进入不同的域.

3. 根据系统数据的组成情况及完整性控制的粒度，定义数据的类型. Lipner 把系统数据分为开发代码及测试数据、系统产品代码、系统产品数据、系统开发中可修改程序、系统程序、系统及应用日志及软件工具；它们应该分属不同的数据类型.

4. 确定域与域的关系表 DDT, 域、类型关系表 DTT; 确保域与域之间不发生损害完整性的信息流动. 如果域与域之间必须进行信息的流动，它们必须通过 WFT 来实现.

5. 确证 WFT 实现了它应该实现的功能，而且它必须被特殊标识为一种数据类型，防止非授权用户的使用及修改，对 WFT 的使用情况进行审计，防止授权用户的不恰当修改.

6. 对试图操作完整性保护数据的用户要进行身份认证，实现 IVP, 对 IVP 软件的安装和维护必须由专人负责；IVP 必须被特殊标识为一种数据类型. 对创建完整性保护数据的行为，必须实现 IVP, 即检查行为特权的授权情况.

7. 域与域的关系表 DDT 及域、类型的关系表 DTT 必须由专人负责，确保配置的正确性，及这些关系的权威性.

1.4 模型变量及常量

• Domains := $\{Admin_d,\ System_d,\ Net_Applied_d,\ User_d,\ Trusted_d,\ \cdots\}$.

• Types, Subjects, Objects, Users, WFT, IVP.

• Roles := $\{SecAdmin, SysAdmin, NetAdmin, AudAdmin, \cdots,\}$.

• 决策集 D := $\{YES, NO, UNDEFINED\}$, 请求集 R, 时间集 Time.

• 域间关联模式集 DCM := $\{auto, exec, signal, null\}$.

• 域对类型的访问模式集 DTAM := $\{read(r),\ write(w),\ execute(x),\ append(a),\ create(c),\ delete(l),\ directory_traversal(d)\}$.

• DDT $\subseteq Domains \times Domains\times$ DCM.

• $role_domains$: $Domains \longrightarrow Roles$, 它给每一个角色 r 分配一个该角色可以执行的域的集合.

• DTT $\subseteq Domains \times Types \times$ DTAM, $(d, t, a) \in$ DTT 的含义是：域 d 中的主体，能以方式 a 访问具有类型 t 的对象. 本模型规定，每一个域 d, 都必须存在一个型 t_d, $(d, t_d, read), (d, t_d, write) \in$ DTT. 这个型称为域 d 的特征型. 显然，特征型可以有多个.

• 可信管道集 T $\subseteq seq(Types\times$ DTAM$\times Domains\times$ DTAM$\times Types)$, 如果 $p \doteq ((t_{11}, z, d_1, y, t_{12}), (t_{21}, z, d_2, y, t_{22}), \cdots, (t_{k1}, z, d_k, y, t_{k2})) \in T$, 那么 $z = r \vee x,\ y = w \vee a \vee c \vee l,\ t_{(s-1)2} = t_{s1},\ (d_m, t_{m1}, z),\ (d_m, t_{m2}, y) \in$ DTT, $s = 2, \cdots, k,\ m = 1, \cdots, k$. 称 t_{11} 是可信管道 p 的输入，t_{k2} 为输出. 记为 $t_{11}pt_{k2}$.

• $T_$WFT: T $\longrightarrow seq(WFT)$, 这是可信管道与良构事务的分配关系，要实施可信管道必须经由良构事务，该关系确定了实现可信管道所需的良构事务集合，它把可信管道映射到为实施可信管道而授权使用的良构事务序列.

• $role_authorized_$T: $Roles \longrightarrow 2^T$, 可信管道必须获得授权才能被使用，它给每个角色分配一个具有该角色的主体可以使用的可信管道.

• $current_$T: $Subjects \longrightarrow T$, 它为主体分配一个当前用于实施受限信息流动的可信管道.

• $subject_authorized_domains$: $Subjects \longleftrightarrow Domains$, 这是主体与域间的分配关系，它是一个多对多的关系；也就是说，多个主体可以授权进入同一个域，同时一个主体可以被授权进入多个域.

• $subject_role$: $Subjects \longrightarrow Roles$, 系统中的每个主体都被分配一个角色.

• $user_roles$: $Users \longrightarrow 2^{Roles}$, 系统中的每个用户都被分配一个角色集.

• $subject_user$: $Subjects \longrightarrow Users$, 系统中的每个主体都由用户生成.

• 不可控制关系 $\neg control \subseteq Domains \times Domains$, 若 $(d_1, d_k) \in \neg control$, 则其含义是：域 d_1 的主体绝对不允许进入域 d_k 或者向域 d_k 中的主体发信号，记为 $\neg control(d_1, d_k)$. 该关系的反面就是控制关系，记为 $control(d_1, d_k)$.

如果 $(d_1, d_2, auto \vee exec) \in$ DDT, 我们称 d_1 可以调用 d_2, 记为 $invoke(d_1, d_2)$. 如果存在 $d_1, d_2, \cdots, d_k \in Domains$, 满足 $invoke(d_i, d_{i+1}),\ i = 1, \cdots, k-1$, 那么

显然 d_1 可以控制 d_k. 不可控制关系是策略库中主动定义的关系，它确定了哪些域之间绝对不能发生调用关系，它不是由 DDT 导出的关系.

• 非依赖关系 $\neg clingto \subseteq Types \times Types$, 若 $(t_k, t_0) \in \neg clingto$, 记为 $\neg clingto(t_k, t_0)$, 则其含义是：型为 t_k 的对象中的内容绝对不允许传到型为 t_0 的对象中. 该关系的反面就是依赖关系，记为 $clingto(t_k, t_0)$.

记 $Type(d, observe) := \{t|(d, t, read \vee x) \in \text{DTT}\}$, $Type(d, alter) := \{t|(d, t, w \vee a \vee c \vee l) \in \text{DTT}\}$, 如果存在 $d_1, d_2, \cdots, d_k \in Domains$, 满足 $d_i \neq d_j$, $i \neq j$, 而且 $Type(d_2, alter) \cap Type(d_1, observe) \neq \emptyset$, $Type(d_3, alter) \cap Type(d_2, observe) \neq \emptyset$, $\cdots$, $Type(d_k, alter) \cap Type(d_{k-1}, observe) \neq \emptyset$, $t_0 \in Type(d_1, alter)$ $t_k \in Type(d_k, observe)$, 那么显然型 t_k 依赖于 t_0. 非依赖关系也是策略库中主动定义的关系，它确定了哪些型之间绝对不能发生信息流动，它不是由 DTT 导出的关系.

• $object_type$: $Objects \longrightarrow Types$ 这是客体与类型间的分配关系，它是一个多对一的关系.

• $active_domains$: $Subjects \longrightarrow Domains$, 这是主体与域间的分配关系，它把主体映射到当前主体可以执行的域. 因为一个进程在系统调用 exec 之后将成为新的进程，而用户进程要进入域必须调用 exec 执行入口守护程序，因此主体进程的当前执行的域是惟一的.

• b : 当前访问集，它被表示为 $\{(s, d, o, t, y) \vee (s, s_1, d, d_1, u)|s, s_1 \in Subjects, o \in Objects$, 使得 $d, d_1 \in active_domain(s)$, $t \in Types$, $y \in \text{DTAM}, u \in \text{DCM}\}$, 其含义是：在当前状态下，主动实体 s 活动在域 d 中，被激活的客体 o 具有型 t, s 能以 y 的形式访问 o, 或者 s 可以实现从域 d 到域 d_1 的通信.

1.5 模型不变量

1. 在任何状态下，每个客体有且仅有一个类型. 即：在任何状态下，

$$dom(object_type) = Objects.$$

2. 在任何状态下，每个主体至少隶属于一个域. 即：

$$dom(subject_authorized_domains) = Subjects.$$

3. 在任何状态下，每个主体的当前可执行域必须是该主体获得授权的域. 即：

$$\forall\, s \in Subjects, \quad active_domain(s) \in subject_authorized_domains(s).$$

4. 在任何状态下，实体实施受限信息流动的当前使用的可信管道必须是实体所在角色获得使用授权的可信管道，而且不同可信管道的授权良构事务集合是不相交的. 即：

$$\forall\, s \in Subjects, \quad current_T(s) \in \{role_authorized_T(r)|(r, s) \in subject_role\}.$$

$$\forall\, p_1, p_2 \in T(p_1 \neq p_2) \Longrightarrow T_\text{WFT}(p_1) \cap T_\text{WFT}(p_2) = \emptyset.$$

5. 可信管道的输入、输出型是惟一的，而且可信管道具有原子性，也就是

说，可信管道不能包含子可信管道. 即：

$$(t_k, observe, d_k, alter, t_{(k-1)}, observe, d_{k-1}, \cdots, observe, d_1, alter, t_0) \text{ 是可信管道}$$
$$\Longrightarrow \forall\, i \leqslant j < k, \quad (t_j, observe, d_j, alter, t_{(j-1)}, observe, d_{j-1}, \cdots,$$
$$observe, d_{i+1}, alter, t_i) \text{ 不是可信管道.}$$
$$\forall\, (p_1 \in T)(t_{11}, t_{12}, t_{21}, t_{22} \in Types)(t_{11} p_1 t_{12} \wedge t_{21} p_1 t_{22})$$
$$\Longrightarrow t_{11} = t_{21} \wedge t_{21} = t_{22}.$$

6. $\forall\, s \in Subjects,\ subject_user(s) = u \wedge subject_role(s) = r \Longrightarrow r \in user_roles(u)$.

7. $\forall\, s \in Subjects,\ d \in subject_authorized_domains(s) \wedge subject_role(s) = r \Longrightarrow role_domains(d) = r$.

8. 如果存在 $t_k, t_0 \in Types$, 满足 $clingto(t_k, t_0)$, 那么它们必须满足如下的条件之一：

• $(\forall\, d \in Domains)((d, t_0, observe) \in \text{DTT} \Longrightarrow (d, t_k, observe) \in \text{DTT}) \wedge ((d, t_k, alter) \in \text{DTT} \Longrightarrow (d, t_0, alter) \in \text{DTT})$. 或者

• 存在可信管道 p, 使得 t_k 是它的输入, 而 t_0 是输出. 这里 $alter$ 代表 $\{w, l, c, a\}$ 中的一种操作.

该不变量的含义是：如果进程能访问某个类型，那么或者访问是DTT直接许可的，就是说，访问在DTT确定的范围内，或者经由可信管道实现.

9. 对于给定的 $x, y \in Domains$, 满足： $\neg control(x, y)$, 那么对任意 $z \in Domains$, 只要 $control(x, z)$, 就必然有 $\neg control(z, y)$. 即： $(x, y \in Domains)(\neg control(x, y))(\forall\, z \in Domains) control(x, z) \Longrightarrow \neg control(z, y)$. 也就是说，不可控制关系不可因状态的变化而变化，因为不可控制关系决定了某些域之间的不可支配性.

对于给定的 $t_1, t_2 \in Types$, 满足：$\neg clingto(t_1, t_2)$, 那么对任意 $t \in Types$, 只要 $clingto(t_1, t)$, 就必然有 $\neg clingto(t, t_2)$. 即：$(t_1, t_2 \in Types)(\neg clingto(t_1, t_2))(\forall\, t \in Types) clingto(t_1, t) \Longrightarrow \neg clingto(t, t_2)$. 也就是说，非依赖关系不可因状态的变化而变化，因为非依赖关系限定了某些型的绝对隔离.

10. 如果存在 $d_1, d_2, \cdots, d_k \in Domains$, 满足 $d_i \neq d_j,\ i \neq j$, 而且 $(s_{11}, d_1, o_{11}, t_1, observe) \wedge (s_{12}, d_2, o_{12}, t_1,\ alter) \in b,\ (s_{21}, d_2, o_{22}, t_2, observe) \wedge (s_{31}, d_3, o_{32}, t_2, alter) \in b, \cdots, (s_{(k-1)1}, d_{(k-1)}, o_{(k-1)1}, t_{(k-1)}, observe) \wedge (s_{(k-1)2}, d_k, o_{(k-1)2}, t_{(k-1)}, alter) \in b,\ \forall\, t_0 \in Type(d_1, alter),\ \forall\, t_k \in Type(d_k, observe)$, 那么它们必须满足如下条件之一：

• $(\forall\, d \in Domains)((d, t_0, observe) \in \text{DTT} \Longrightarrow (d, t_k, observe) \in \text{DTT}) \wedge ((d, t_k, alter) \in \text{DTT} \Longrightarrow (d, t_0, alter) \in \text{DTT})$. 或者

• 存在可信管道 p, 使得 t_k 是它的输入, 而 t_0 是输出. 这里 $alter$ 代表 $\{w, l, c, a\}$ 中的一种操作.

不变量 8 与不变量 10 的不同之处在于：不变量 8 处理的是系统可能存在的信息流，而不变量 10 处理的是当前可能存在的信息流；因此系统满足不变量 8, 就自动满足不变量 10; 但反之则不然. 本文的可信管道的概念与文献 [22] 中的一致，它也与 Linux 系统中提出的管道概念一致，但与文献 [23] 中的管道概念不同，在那里是 $seq(Domains\times$ DTAM$\times Types\times$ DTAM$\times Domains)$, 因此管道的输入、输出是域；下面我们来证明，在引入特征型的概念后，这两个概念达到的效果是相同的. 而且本文的不变量 8 蕴涵类似文献 [20] 中的不变量：如果存在 $d_1, d_2 \in Domains$, 满足 $d_1 \neq d_2$, 而且 $Type(d_1, observe) \cap Type(d_2, alter) \neq \emptyset$, 那么它们必须满足如下条件之一：

- $Type(d_2, observe) \subseteq Type(d_1, observe)$, 或者
- $\forall\, t \in Type(d_1, observe) \cap Type(d_2, alter)$, 存在文献 [23] 中的可信管道 p, 使得 d_2 是可信管道的输入域，d_1 是可信管道的输出域.

定理 1 假设每个域都存在特征型，那么文献 [23] 中的管道概念都可由本文定义的可信管道的概念实现；而且本文的不变量 8 蕴涵上述不变量.

证 不失一般性，我们只考虑管道长度为 2 的情况.

设 $(d_1, alter, t, observe, d_2)$ 是文献 [23] 中的管道，t_1 是 d_1 的特征型，t_2 是 d_2 的特征型. 那么显然，$(t_1, observe, d_1, alter, t, observe, d_2, alter, t_2)$ 是本文的管道，而且进程通过 $(d_1, alter, t, read, d_2)$ 实现信息传送时，它直接从 d_1 写信息到型 t 中，然后再经过 d_2 从型 t 中读取信息. 这个过程也可用本文的管道实现，过程如下：由于 t_1 是 d_1 的特征型，因此进程可以从 d_1 写信息到型 t_1 中，而且还可以从 t_1 中读取，再写入型 t, 经过 d_2 从型 t 中读取信息，再写入型 t_2 中，因为 t_2 是 d_2 的特征型，因此进程可以从 t_2 中读取信息，从而实现文献 [23] 中管道所能传递的信息.

从上面的讨论易看出，不变量 8 蕴涵上述不变量的第二部分，因此只需证明第一部分.

对于 $\forall\, t_0 \in Type(d_2, observe)$, 有 $(t_0, observe, d_2, alter, t, observe, d_1, alter, t_1)$, 从而 $clingto(t_0, t_1)$, 又因为 t_1 是 d_1 的特征型，所以 $(d_1, t_1, observe) \in$ DTT, 由不变量 8 有：$(d_1, t_0, observe) \in$ DTT, 由于 t_0 是任意的, 所以有：$Type(d_2, observe) \subseteq Type(d_1, observe)$.

定义 1 如果存在 $d_1, d_2, \cdots, d_k \in Domains,\ t_1, t_2 \cdots, t_{(k-1)} \in Types$, 满足 $d_i \neq d_j,\ i \neq j$, 而且

$$t_1 \in Type(d_2, alter) \cap Type(d_1, observe),$$

$$t_2 \in Type(d_3, alter) \cap Type(d_2, observe),\ \cdots,$$

$$t_{(k-1)} \in Type(d_{(k)}, alter) \cap Type(d_{k-1}, observe),$$

$$t_0 \in Type(d_1, alter) t_k \in Type(d_k, observe),$$

那么我们称

$$(t_k, observe, d_k, alter, t_{(k-1)}, observe, d_{k-1}, \cdots, observe, d_1, alter, t_0)$$

是 $clingto(t_k, t_0)$ 的一个实现. 称这个实现是原子的，如果该实现或者是可信管道，或者实现中的任何一段都不是可信管道，也就是说，

$$\forall\, i \leqslant j, \quad (t_j, observe, d_j, alter, t_{(j-1)}, observe, d_{j-1}, \cdots, observe, d_{i+1}, alter, t_i)$$

都不是可信管道.

定理 2　如果

$$(t_k, observe, d_k, alter, t_{(k-1)}, observe, d_{k-1}, \cdots, observe, d_1, alter, t_0)$$

是 $clingto(t_k, t_0)$ 的一个原子实现，而且不是可信管道，那么

$$\forall\, 1 \leqslant i \leqslant k, \quad Type(d_i, alter) \subseteq Type(d_i, observe).$$

证　由不变量 8 直接可证.

1.6　模型规则及限制性条件

1. $get_access(d, t, y)$

请求类型为 $R^{(1)}$, 含义是：在域 d 中的主体 s 请求以 y 方式访问类型为 t 的客体 o. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) $(d, t, y) \in$ DTT.

如果条件被满足，决策输出 YES. 下一个状态变化如下：

- $b^* = b \cup \{(s, d, o, t, y)\}$
- 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

2. $release_access(d, t, y)$

请求类型为 $R^{(2)}$, 含义是：在域 d 中的主体 s 请求释放以 y 方式访问类型为 t 的客体 o. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，决策输出 YES. 下一个状态变化如下：

- $b^* = b \backslash \{(s, d, o, t, y)\}$
- 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

3. $execute_\mathrm{T}(d, s, p, t, o, y)$

请求类型为 $R^{(3)}$, 含义是：在域 d 中的主体 s 请求执行可信管道 p 访问型为 t 的对象 o. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) $(d, t, y) \notin$ DTT.

(b) 设管道 p 的输入和输出型为 t_{I} 和t_{O}, 如果 $y = observe$, 则 $t = t_{\mathrm{I}} \wedge (d, t_{\mathrm{O}}, observe) \in$ DTT; 如果 $y = alter$, 则 $t = t_{\mathrm{O}} \wedge (d, t_{\mathrm{I}}, alter) \in$ DTT.

(c) $subject_role(s) = r \wedge p \in role_authorized_\mathrm{T}(r)$.

(d) 如果 $y = creat(delete)$, 那么 $\forall\ s_1 \in Subjects, \forall\ d_1 \in Domains, \forall\ y \in$ DTAM, $(s_1, d_1, o, t, y) \notin b$.

如果条件被满足，决策输出 YES. 下一个状态变化如下：

• $b^* = b\backslash\{(s,d,o_1,t_1,y_1)|(s,d,o_1,t_1,y_1) \in b \wedge o_1 \neq o\} \cup \{(s,d,o,t,y)\}$

• $current_\text{T(s)}^* = p$

• 如果 $y = creat$, 那么 $Objects^* = Objects \cup \{o\}$; 如果 $y = delete$, 那么 $Objects^* = Objects\backslash\{o\}$

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

4. $exit_\text{T}(d,s,p,t,o,y)$

请求类型为 $R^{(4)}$, 含义是：在域 d 中的主体 s 请求结束执行 p 访问型为 t 的对象 o. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，决策输出 YES. 下一个状态变化如下：

• $b^* = b\backslash\{(s,d,o,t,y)\}$

• $current_\text{T(s)}^* = \text{Nil}$ 即主体 s 当前没有可执行的 p

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

5. $create_object(d,t)$

请求类型为 $R^{(5)}$, 含义是：在域 d 中的主体 s 请求创建类型为 t 的新客体 o_{new}. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) $(d,t,creat) \in$ DTT

如果条件被满足，决策输出 YES. 下一个状态变化如下：

• $b^* = b \cup \{(s,d,o_{\text{new}},t,create)\}$

• $Objects^* = Objects \cup \{o_{\text{new}}\}$

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

6. $delete_object(d,t)$

请求类型为 $R^{(6)}$, 含义是：在域 d 中的主体 s 请求删除类型为 t 的客体 o. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) $(d,t,delete) \in$ DTT;

(b) $\forall\, d_1 \in Domains$, $\forall\, y \in$ DTAM, $(s_1,d_1,o,t,y) \notin b$.

如果条件被满足，决策输出 YES. 下一个状态变化如下：

• $b^* = b \cup \{(s,d,o,t,delete)\}$

• $Objects^* = Objects\backslash\{o\}$

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

7. $request_send_signal(s_1,s_k,d_1,d_k)$

请求类型为 $R^{(7)}$, 含义是：在域 d_k 中的进程 s_k 请求向域 d_1 中的进程 s_1 发信号或跟踪该进程，如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件之一：

(a) 如果存在 $d_1, d_2, \cdots, d_k \in Domains$, 满足 $invoke(d_i, d_{i+1})$, $i = 1, \cdots, k-1$, 使得 $d_i \in subject_authorized_domains(s_1)$, s_k 是 s_1 的子进程，而且信号仅是通知父进程该进程的结束．或者

(b) $d_1 = d_k$, 或者

(c) $(d_k, d_1, signal) \in$ DDT.

如果条件之一被满足，决策输出 YES. 下一个状态变化如下：

• $b^* = b \cup \{(s_k, s_1, d_k, d_1, signal)\}$

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

$request_transition(s_1, s_k, d_1, d_k)$

请求类型为 $\hat{R^{(7)}}$, 含义是：在域 d_1 中的进程 s_1 请求向域 d_k 中的进程 s_k 转换．如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) 如果存在 $d_1, d_2, \cdots, d_k \in Domains$, 满足 $invoke(d_i, d_{i+1})$, $i = 1, \cdots, k-1$, 使得 $d_i \in subject_authorized_domains(s_1)$.

如果条件被满足，决策输出 YES. 下一个状态变化如下：

• $b^* = b \cup \{(s_1, s_k, d_1, d_k, auto \vee exec)\}$

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

8. $creat_type(d_1, r_1, t)$

请求类型为 $R^{(8)}$, 含义是：在域 d_1 中的具有角色 r_1 的主体 s_1 请求生成类型 t. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) r_1 是 SecAdmin, d_1 是 $Admin_d$, $t \notin Types$.

如果条件满足，决策输出 YES. 下一个状态变化如下：

• $Types^* = Types \cup \{t\}$

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

9. $add_\mathrm{T}(d, r, p, r_1)$

请求类型为 $R^{(9)}$, 含义是：在域 d 中的具有角色 r 的主体 s_1 请求添加 p 到角色 r_1 的授权使用的可信管道集．如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) r 是 SecAdmin, d 是 $Admin_d$;

(b) $p \notin role_authorized_\mathrm{T}(r_1)$.

如果条件 (a) 和 (b) 被满足，决策输出 YES. 下一个状态变化如下：

• $role_authorized_\mathrm{T}(r_1)^* = role_authorized_\mathrm{T}(r_1) \cup \{p\}$

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

10. $add_\mathrm{DTT}(d, r, (d_1, t, y))$

请求类型为 $R^{(10)}$, 含义是：在域 d 中的具有角色 r 的主体 s_1 请求生成并添加 (d_1,t,y) 到 DTT. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) r 是 SecAdmin, d 是 $Admin_d$, $(d_1,t,y)\notin$ DTT;

(b) $(\forall\, t_1,t_k\in Types)\neg clingto(t_1,t_k)\wedge(d_1,t,y)\in \mathrm{DTT}\Longrightarrow\neg clingto(t_1,t_k)$.

如果条件满足，决策输出 YES. 下一个状态变化如下：

- $\neg clingto^*=\neg clingto\cup\{\neg clingto(t,t_b)|\exists t_a, clingto(t_a,t)\wedge\neg clingto(t_a,t_b)\}$.
- $\mathrm{DTT}^*=\mathrm{DTT}\cup\{(d_1,t,y)\}$
- 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

11. $change_type(d,r,t,t_1,o)$

请求类型为 $R^{(11)}$, 含义是：在域 d 中的具有角色 r 的主体 s_1 请求把 o 的类型从 t 变成 t_1. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) r 是 SecAdmin, d 是 $Admin_d$;

(b) $\forall\, d_1\in Domains$, $s\in Subjects$, $\forall\, y\in$ DTAM, $(s,d_1,o,t,y)\notin b$.

如果条件 (a) 和 (b) 被满足，决策输出 YES. 下一个状态变化如下：

- $object_type^*=object_type\cup\{(o,t_1)\}\backslash\{(o,t)\}$
- 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

12. $delete_$DTT$(d,r,(d_1,t,y))$

请求类型为 $R^{(12)}$, 含义是：在域 d 中的具有角色 r 的主体 s_1 请求从 DTT 删除 (d_1,t,y). 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验如下条件：

(a) r 是 SecAdmin, d 是 $Admin_d$;

(b) $\forall\, s\in Subjects$, $o\in Objects$, $(s,d_1,o,t,y)\notin b$, $(d_1,t,y)\in$DTT;

(c) $\forall\, p\in T$, $(d_1,t,y)\notin p$.

如果条件被满足，决策输出 YES.

下一个状态变化如下：

- $\mathrm{DTT}^*=\mathrm{DTT}\backslash\{(d_1,t,y)\}$
- 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

13. $delete_$T(d,r,p)

请求类型为 $R^{(13)}$, 含义是：在域 d 中的具有角色 r 的主体 s_1 请求删除 p. 如果访问类型不恰当，则下一状态保持原状，决策输出 UNDEFINED; 否则，检验条件：

(a) r 是 SecAdmin, d 是 $Admin_d$;

(b) $\forall\, s\in Subjects$, $current_T(s)\neq p$.

如果条件 (a) 和 (b) 被满足，决策输出 YES. 下一个状态变化如下：

• $\forall\, r \in Roles,\ role_authorized_\mathrm{T}^*(r) = role_authorized_\mathrm{T}(r)\backslash\{p\}$

• 其他变量的赋值不变

如果上述条件不成立，决策输出 NO; 下一状态保持原状.

1.7 计算模型及基本安全定理

该模型的状态 V 为

(Subjects, Objects, Roles, Types, Domains, Users, WFT, $\neg control$, $\neg clingto$, $role_domains$, DDT, DTT, T, b, $T_$WFT, $current_$T, $role_authorized_$T, $subject_authorized_domains$, $object_type$, $subject_role$, $user_roles$, $subject_user$, $active_domains$).

状态转移函数为 ρ: $V \times R \longrightarrow V \times D$.

定义 2　设 $\omega = \{\rho_1, \rho_2, \cdots, \rho_s\}$ 是相对于请求集 R, 和决策集 D 的一组规则集，关系 $W(\omega) \subseteq R \times D \times V \times V$ 定义为：$(R_k, D_m, V^*, V) \subseteq W(\omega)$ 当且仅当 $D_m \neq undefined$, 并且存在惟一的 i, $1 \leqslant i \leqslant s$, 使得 $(D_m, V^*) = \rho_i(R_k, V)$.

定义 3　系统 $\sum(R, D, W, z_0) \subseteq X \times Y \times Z$(其中 X, Y, Z 分别代表 R, D, V 的有限序列的集合) 定义为：$(x, y, z) \in \sum(R, D, W, z_0)$ 当且仅当对每一个 $s \in Time$, $(x_s, y_s, z_s, z_{s-1}) \in W$, 其中 z_0 是初始状态. 系统 $\sum(R, D, W, z_0)$ 称为是安全的，如果系统的每一个状态均满足系统给出的 10 个不变量.

定义 4　我们称关系 W 保持某个不变量, 如果对每一个 $s \in Time$, $(x_s, y_s, z_s, z_{s-1}) \in W$, z_{s-1} 满足该不变量，那么 z_s 也满足该不变量. 称规则 ρ_i 保持某个不变量，如果 $(D_m, V^*) = \rho_i(R_k, V)$, V 满足该不变量，则 V^* 也满足该不变量. 由关系 W 的定义不难看出：若 $\omega = \{\rho_1, \rho_2, \cdots, \rho_n\}$ 中的每个规则都保持某个不变量，则 W 保持这个不变量.

定理 3　假设 $\omega = \{\rho_1, \rho_2, \cdots, \rho_n\}$ 中的每个规则都是保持 10 个不变量的，且 z_0 满足 10 个不变量，则系统 $\sum(R, D, W, z_0)$ 满足 10 个不变量.

证　对状态序列的长度用归纳法.

定理 4　本模型中定义的 13 条迁移规则保持 10 个不变量.

证　要证明这个定理仅需根据每条规则的条件，逐一验证它们满足 10 个不变量就可. 由于验证方法是类似的, 因此限于篇幅, 本文只给出比较复杂的规则 7 中第一个操作满足 10 个不变量的验证. 因为操作 $request_send_signal(s_1, s_k, d_1, d_k)$ 仅仅影响到状态变量 b (注意：如果进程 s_k 请求向域 d_1 中的进程 s_1 发送 $kill$ 命令，那么在执行该命令之前，系统必须检查消灭一个进程的条件，本模型不考虑这种情况，把它留给特权模型), 因此与 b 无关的不变量自然保持不变.

下面我们就来对 10 个不变量逐一进行验证.

• 对不变量 1 的验证：由于操作 $request_send_signal(s_1, s_k, d_1, d_k)$ 不生成对象，所以对关系 $object_type$ 没有影响，从而操作前后该关系保持不变.

• 对不变量 2 的验证：由于操作 $request_send_signal(s_1, s_k, d_1, d_k)$ 不生成主体，所以操作前后对授权关系没有任何影响.

• 对不变量 3~7 的验证：由于本模型不考虑消灭一个进程的情况，因此操

作 $request_send_signal\ (s_1, s_k, d_1, d_k)$ 不生成主体也不消灭主体，因此对可信管道、角色及用户没有影响.

• 对不变量 8 的验证：注意在本模型中实现 $clingto(t_k, t_0)$ 的方式只有两种：一是经过域与型之间的交替传递；二是通过可信管道传递. 对于第二种情况，由于操作 $request_send_signal(s_1, s_k, d_1, d_k)$ 不影响可信管道，因此操作后不变量保持成立. 对于第一种情况，我们用数学归纳法来验证，在系统配置之初，必须是满足我们的不变量的. 假设操作发生后，$clingto(t_k, t_0)$ 成立，那么在操作没有发生之前，只能有两种子情况出现，第一种子情况是：$clingto(t_k, t_0)$ 成立，由于操作之前的状态是满足不变量 8 的 (归纳假设), 因此操作后的状态满足不变量 8; 第二种子情况是：$clingto(t_k, t_0)$ 不成立，此时，必然存在两个型 t^* 及 t^{**}, 满足：操作发生前 $clingto(t_k, t^*)$ 及 $clingto(t^{**}, t_0)$ 成立，但 $clingto(t^*, t^{**})$ 不成立；操作发生后，它们都成立，而且无论操作发生前后都有 $(d_k, t^{**}, w) \in \mathrm{DTT}$, $(d_1, t^*, observe) \in \mathrm{DTT}$, 由归纳假设，我们须证：$(\forall\ d \in Domains)((d, t^{**}, observe) \in \mathrm{DTT} \Longrightarrow (d, t^*, observe) \in \mathrm{DTT}) \wedge ((d, t^*, alter) \in \mathrm{DTT} \Longrightarrow (d, t^{**}, alter) \in \mathrm{DTT})$. 为证此，我们注意到，我们已经假设系统已发生有效迁移，因此它必然满足规则 7 的条件，由此可见，如果假设规则 7 的条件 (a) 或 (c) 出现，那么由于它们都不足以推出 $clingto(t^*, t^{**})$ 成立，矛盾. 所以只有条件 (b) 可能出现，此时 $(d_{middle}, t^{**}, w) \in \mathrm{DTT}$, $(d_{middle}, t^*, observe) \in \mathrm{DTT}$ 成立，这里 $d_{middle} = d_1 = d_k$. 因此，这自然蕴涵了我们要证明的结论.

• 对不变量 9 的验证：仍然使用归纳法，假设 $\exists\ z, y$, 在操作发生之前满足 $\neg control(z, y)$, 操作发生后满足 $control(z, y)$, 那么出现这种情况的原因必然是：在操作发生之前 $control(z, d_k)$, $\neg control(d_k, d_1)$ 及 $control(d_1, y)$ 成立，但由操作的条件知：要发生操作必然有 $control(d_k, d_1)$, 矛盾. 所以，操作后不变量 9 成立.

• 对不变量 10 的验证：类似不变量 8 可验证.

2 模型的应用

本文的模型已成功用于中国科学院软件研究所信息安全技术工程研究中心的 B2 级安全操作系统的开发中，并在原型系统安胜 OS V4.0 中有效实现，测试表明，该模型获得较好的完整性保护效果. 在文献 [24], 我们定义了如下非管理用户域和管理用户域：

• 非管理用户域：

– daemon_d—— 与系统守护进程 (包括 init) 相关的域；系统的初始域.

– login_d—— 与 DTE login 程序相关的域.

– user_d—— 与普通用户的会话相关的域.

– operate_d—— 与系统操作用户的会话相关的域.

• 管理用户域：

– admin_d—— 与管理用户的会话相关的域.

我们把系统所有的文件和对象分为26种类型：base_t, bin_t, dev_t, conf_t, user_t, lib_t, disk_t, root_t, mnt_t, package_t, proc_t, sbin_t, tmp_t, boot_t, log_t, varrunt_t, shell_t, oshell_t, login_xt, getty_xt, tty_t, wdev_t, init_t, build_t, devnull_t, uwdev_t. 并根据我们提出的不变量，给这些型和域配置了DDT和DTT表，对这些型的解释及相关的表可以参考我们的技术文档[24].

除了上述的域和型外，我们还给出了非控制关系、非依赖关系、可信管道及角色等，并验证这些配置满足我们的不变量. 详细的过程可在上述提到的技术文档中找到. 值得一提的是，在我们的模型实现中我们仅用可信程序实现了简单的可信管道，复杂的可信管道实现技术有待进一步研究.

3　对模型的分析及相关的工作

Nyanchama在他的博士学位论文[1)]的第五章中提出了一个用于刻画基于角色的系统中信息流的抽象层次较高的一般框架，该框架很容易用DTE技术进行描述，所以那里刻画的阻止非法信息流的不变量也容易对应地进行翻版，文中的constraint 5.5, 类似本文定理2.1中的不变量；但是，由于Nyanchama考虑的架构比较泛，提到的一些性质对操作系统控制安全信息流是无用的，constraint 5.4用DTE技术来描述就是：$\forall\ s \in Subjects,\ (s, d_1, o_1, t_1, read) \in b \wedge (s, d_2, o_2, t_2, alter) \in b \Longrightarrow (t_1 \in Type(d_2, read)) \vee (\exists\ p \in \mathrm{T},\ d_1, d_2 \in p)$.

但这个不变量是没有意义的，因为进程进入域须执行系统调用exec, 因此同一个进程不能生存于不同的域；从而不变量的前提不成立. 另一方面，文中仅处理了“明流”, 并没有处理“暗流”; 本文的不变量8就是处理“暗流”的不变量. 另外，在操作系统中存在生存周期不同的客体，对于生存周期短的客体，如果要求满足不变量8, 那么这样的要求太强，实际应用中是不合理的，因此我们仅要求它们满足不变量10.

在文献[25]中，抽象地提出了一种对模型的限制性条件，序相关及序无关历史限制，对这种限制，Jaeger认为是他在文献[25]所描述的限制中最复杂的限制，他没有给出太多的刻画；我们认为本模型中的不变量8就是序相关历史限制的典型实例. Haigh在文献[23]中，曾提出类似本模型的不变量8中第二部分的安全条件，并在互不干扰理论框架下证明不变量8中第二部分的安全条件足以保证在互不干扰意义下无非法信息的流动. 但Haigh并不关心这些信息流的含义，实际上信息流的非法与否与信息流的含义紧密相关，就我们所知，是本模型因考虑恶意代码的传染问题而第一次把不变量8用于完整性保护. 类似于Haigh在文献[23]中的证明，我们能证明：不变量8足以保证在互不干扰意义下没有完整性Trojan horse的非法传染；这里的非法是指违背完整性策略的行为，当然，为形式地证明这个结论必须把非法传染用互不干扰理论清楚表达. 这个结论表明了我们模型不变量8的合理性.

1) 见573页脚注

本模型是为实现操作系统的结构化保护而提出，考虑到工程实现的复杂性及操作系统的功能模块相对稳定，我们不考虑域的动态变更，文献 [16] 考虑了域的动态配置，在那里 Tidswell 仅探讨了动态配置所涉及到的局部性质，因此把它们用于一个完整的模型中，还需进一步的研究. 另外，本模型中引入了角色，一方面是受文献 [26] 的影响，另一方面我们认为实现职责隔离的有效手段就是基于角色的方法；我们的不变量 6, 7 就是借鉴文献 [26] 中的协调关系 $subject_user_role_consistent$ 及 $subject_domain_role_consistent$, 但那里仅是一种局部的想法，并不系统地考虑它与其他变量的关系. 本模型虽然使用了良构事务，但我们的方法与 Clark-Wilson 模型中的用法不同，我们的良构事务仅分配给角色，而不管实施角色的主体；与我们不同, Clark-Wilson 模型中需要维护一张主体使用良构事务的授权表 (Fischer-Hubner 的隐私策略模型也要维护这样的表). 我们这样做的理由是：良构事务太多有可能导致系统不实用；另外，操作系统的进程动态性太强，维护这样的表几乎不可能；再者，主体进入域是经过 IVP 的，维护这样的表就有点重复授权之嫌.

4 结束语

本文提出一个基于 DTE 技术的完整性保护模型，我们深入研究了完整性策略的现状及其内涵；之后，提出为完成完整性保护进行配置 DTE 的一些基本规则，这些规则类似于文献 [19] 中的规则，但它们之间仍然有本质的不同. 这些规则主要是一些保障规则 (certification rule), 对于执行规则 (enforcement rule) 主要体现在我们的状态机模型中，在该模型中我们精确描述不变量及迁移规则，接着比较了与其他模型的异同.

本模型虽然提出了配置 DTE 的一些基本规则，但在具体的系统设计中仍然需要进一步细化. 例如：当把角色加入模型时，如何把角色的特权与域结合起来？这将是我们另一篇文章的主题. 正如前面提到的，本模型没有考虑域的动态变化，但著名的 SELinux 中有此设置，我们将在以后的工作中研究它. SELinux 中设置了 43 个域，大约 780 个类型及 133775 个规则；面对如此复杂的设置，我们将用这里提出的模型分析它的安全性，这个工作目前正在进行中.

参 考 文 献

1 Jueneman R R. Integrity controls for military and commercial applications. Fourth Aerospace Computer Security Applications Conference. Florida: IEEE Computer Society Press, 1988. 298∼322

2 Ruthberg Z G, Polk W T. Report of the Invitational Workshop on Data Integrity. National Institute of Standards and Technology, September 1989. NIST Special Publication 500-168

3 Mayfield T, Boore J M, Welke S R. Integrity-oriented control objectives: proposed revisions to the trusted computer systems evaluation criteria (TCSEC, DoD5200.28.STD), IDA document D-967, prepared for National Security Agency (U. S.), available at http://www.mirrors.wiretapped.net/security/info/reference/ncsc-publications/C-TR-111-91.pdf

4 Information Technology-Security Techniques-Evaluation Criteria for IT Security — Part 2: Security Functional Requirements. ISO/IEC 15408-2, 1999, International Standards Organization
5 Abrams M D, Joyce M V. Trusted system concepts. Computers and Security, 1995, 14(1): 45∼56
6 Bell D E. Modeling the "multipolicy machine", in Proceedings of the 1994 ACM SIGSAC on New Security Paradigms Workshop, August 1994, Little Compton, RI USA. 2∼9
7 Mayfield T, Roskos J E, Welke S R, et al. Integrity in Automated Information Systems. NCSC Technical Report. National Computer Security Center, 1991
8 Sandhu R S. On five definitions of data integrity. In: Proceedings of the IFIP WG11.3 Workshop on Database Security, Alabama: Lake Guntersville, 1993
9 LaPadula L J. Rule-set modelling of a trusted computer system. In: Abrams M D, Jajodia S, Podell H J, eds. Information Security: An Integrated Collection of Essays. Los Alamitos: IEEE Computer Society Press, 1995
10 Kargar P A, Austel V, Toll D. A new mandatory security policy combining secrecy and integrity. IBM Research Report, RC21717, 2000
11 Fraser T. LOMAC: Low water-mark integrity protection for COTS environments. NAI Labs Report 0775, 2000
12 Ott A. The rule set based access control (RSBAC) Linux kernel security extension. In: 8th International Linux Kongress, Enschede, November, 2001
13 Smith R E. Sidewinder: Defense in depth using type enforcement. Secure Computing Corporation Report, 2000
14 Badger L, Sterne D F, Sherman D L, et al. Practical domain and type enforcement for UNIX. In: Proceedings of IEEE Symposium on Security and Privacy, 1995
15 Walker K M, et al. Confining root programs with domain and type enforcement (DTE). In: Proceedings of 1996 Usenix Security Symposium, San Jose, CA, 1996
16 Tidswell J E, Potter J M. An approach to dynamic domain and type enforcement. In: Proceedings of the 2nd Australasian Conference on Information Security and Privacy, July 1997
17 Boebert W E, Kain R Y, Young W D. Secure computing: The secure ada target approach. In: Turn R, ed. Advance in Computer System Security, Vol 3. Norwood: Artech House, Inc, 1988
18 Cohen F. Computer viruses: Theory and experiments. In: Turn R, ed. Advance in Computer System Security, Vol 3. Artech House, Inc, 1988
19 Lee T M P. Using mandatory integrity to enforce "commercial" security. In: Proceedings of IEEE Symposium on Security and Privacy, 1988. 140∼146
20 Thomsen D J. Role-based application design and enforcement. In: Jajodia S, Landwehr C E, eds. Database Security, IV: Status and Prospects, Amsterdam: North-Holland, 1991. 151∼168
21 Lipner S. Non-discretionary control for commercial applications. In: Proceedings of IEEE 1982 Symposium on Research in Security and Privacy, 1982. 2∼10
22 Fraser T, Badger L. Ensuing continuity during dynamic security policy reconfiguration in DTE. In: Proceedings of 1998 IEEE Symposium on Research in Security and Privacy. Oakland: IEEE Computer Society Press, 1998. 15∼26
23 Haigh J T, Young W D. Extending the noninterference version of MLS for SAT. In: Proceedings of 1986 IEEE Symposium on Research in Security and Privacy. Oakland: IEEE Computer Society Press, 1986. 232∼239
24 季庆光，唐柳英，沈晴霓．安胜OS V4.0安全策略形式模型设计及其在系统中的应用．信息安全技术工程研究中心，"结构化保护级"安全操作系统技术文档，2003年3月
25 Jaeger T, Tidswell J E. Practical safety in flexible access control models. ACM Transactions on Information and System Security, 2001, 4(2): 158∼190
26 Hoffman J. Implementing RBAC on a type enforced system. In: Proc 13th Annual Computer Security Applications Conference, 1997. 158∼163