扩展不干扰模型(ENISM)及基于 CSP 的描述和验证方法

崔 隽 黄 皓 高晓春

[1)](南京大学软件新技术国家重点实验室 南京 210093)

[2)](南京大学计算机科学与技术系 南京 210093)

摘 要 在不干扰理论的基础上,提出扩展不干扰模型 ENISM 及其验证方法,用以描述和分析操作系统中的信息流策略.工作包括:(1)依据系统功能模块定义多个执行域,以即将执行的可能动作序列集合与可读取的数据存储值集合一同作为 ENISM 定义执行域安全状态的基础;(2)给出判定系统中不存在违反策略的执行轨迹和数据流动的条件 ENISM-CC;(3)基于通信顺序进程给出 ENISM-CC 的语义及操作系统模块设计的形式化描述和验证方法.

关键词 不干扰模型;通信顺序进程;形式化描述;形式化验证;完整性

中图法分类号 TP309 **DOI 号**: 10.3724/SP.J.1016.2010.00877

An Extended Non-Interference Security Model (ENISM) and Its CSP-Based Description and Verification Method

CUI Jun HUANG Hao GAO Xiao-Chun

(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)

(*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093)

Abstract Based on the theory of non-interference, this paper proposes an extended non-interference security model ENISM, for the purpose of specification and analysis of information flow policies in operating systems. This paper includes the following works: firstly, system modules would be recognized as domains, and the traces set which contains traces may be implemented after a system state and the data values set at the state are two most important analysis gist for defining the secure states in ENISM. Secondly, the sufficient conditions ENISM-CC are proposed on which unsafe traces and data flow is not existed. Thirdly, this paper gives out a formal description method for system design and describes the semantic ENISM-CC based on the Communicating Sequential Processes CSP.

Keywords non-interference model; communicating sequential processes; formal description; formal verification; integrity

1 引 言

用信息流模型来形式化地分析操作系统的安全特性,是目前操作系统安全验证的主要方法之一.信息流分析的方法使我们能够清楚地理解系统内部信息的流动方向,从独立于系统操作的另一个角度观察系统的实际行为.操作系统的很多安全属性都可

收稿日期:2008-04-23;最终修改稿收到日期:2009-07-10.本课题得到国家"八六三"高技术研究发展计划项目基金(2007AA01Z409)与国家自然科学基金(60473093)资助.**崔 隽**,男,1981 年生,博士研究生,研究方向为操作系统形式化验证、安全模型. E-mail: ctops@sina.com. **黄 皓**,男,1957 年生,博士,教授,博士生导师,主要研究领域包括系统软件、信息安全. **高晓春**,男,1979 年生,博士研究生,研究方向为形式化验证、安全模型.

以表述成一系列相关的信息流策略，比如机密性、完整性、隔离性、不可旁路性等. 通过信息流分析，我们可以证实一个操作系统的设计是否能够满足给定的信息流策略所表达的安全属性. 信息流分析方面的研究工作很多，Goguen 和 Meseguer 提出的不干扰模型[1]是最主要的研究成果之一. 不干扰模型通过观察信息域中信息值的改变来感知信息的流动，而不是简单依靠对读写操作的考察来推测信息流向，具有比其它信息流模型更好地发现确定性系统中各种存储隐蔽信道问题的能力. Haigh、Young[2]和 Rushby[3]等人在不干扰模型基础上在信道控制方面对其进一步探讨，提出了干扰的非传递性. 谢钧[4]等则将不干扰模型从确定性系统推广到非确定系统. 虽然不干扰模型在信息流策略的形式化描述和验证方面有很好的表现，但不干扰模型主要关注的仍是存储信息的改变，忽略了执行轨迹的变化. 尽管广义上讲，信息域执行轨迹的改变也是一种信息的表征，但是在具体的系统策略验证过程中很难直观地判断目标域未来执行的路径是否发生了改变. 仅依靠考察存储值的改变并不能完全反映出执行轨迹的变化，而执行轨迹的变化，特别是安全控制模块轨迹的改变或被旁路往往会产生极其严重的后果. 因此，为了更好地表达和分析系统的信息流策略，信息域的行为执行是否受到其它域的干扰也是信息流分析工作必须考察的对象.

Ryan[5]、Roscoe[6]和 Graham[7]都分别使用进程代数的方法从不同的视角描述了不干扰模型的语义，并阐述了高密级信息域的执行轨迹对低密级信息域的输出及可执行动作集合的影响，Roscoe 和 Goldsmith[8]在此基础上还针对非传递特性作了进一步探讨，并给出了判定干扰性存在的方法. 马建平[9]等则将不干扰关系定义为如果一个信息域 A 不干扰另一个信息域 B，则 A 的行为既不会改变 B 内存储变量的值，也不会影响 B 当前状态下可执行的动作. 上述文章表明，对执行序列的描述有助于分析信息域之间的不干扰属性，然而，本文认为还可以在 3 个方面做进一步探讨. 首先，用进程代数描述不干扰属性，主要描述的是行为本身，而忽视了输入数据对行为执行的影响，往往将具有不同输入值的同一输入事件视为同一事件来考虑，这就屏蔽或减弱了不同输入值对信息域的干扰作用. 其次，本文认为当前状态下可执行的序列集合比当前状态下可执行的事件集合更能准确地反映信息域内执行的改变，可执行的序列集合可反映出目标域未来的执行趋势，其不仅可以反映出未来每一个状态的可执行事件，更将各个状态联系起来更精确地考察目标域的执行改变. 最后，目标域的执行受到了干扰还是信息值受到了干扰并不总是一致的，应分别讨论. 在实际的信息系统中，两个合作的模块一定存在着合理的信息流动（如访问控制器的输入模块和决策模块），但这并不表明合作模块之间一定允许通过交互数据改变对方的执行轨迹（如输入模块只许给决策模块传递决策信息，而不允许改变决策模块的决策过程）. 因此，我们认为应针对目标操作系统的执行不干扰和存储不干扰分别制定安全策略.

基于以上考虑，本文提出更适于描述操作系统的信息流策略的扩展不干扰安全模型（ENISM）及验证系统设计是否满足该策略的验证方法. 我们将各个功能相对独立的系统模块均视作执行域. ENISM 在传统不干扰模型的基础上增加了未来可能执行的动作序列集合的改变作为分析目标域安全状态是否发生改变的依据之一，动作序列集合和信息存储值集合一同考察更能准确地反映出目标域的当前状态；ENISM 将输入数据值反映在输入动作中，将能够接收不同输入数据的同一输入动作看作同一动作的不同执行，以反映不同输入对目标域的影响；ENISM 分别考察目标域的执行不干扰条件和数据存储的不干扰条件，有助于从执行域之间的执行干扰和存储干扰两方面分析、描述和验证系统的信息流策略.

为了能够形式化描述一个实际的操作系统设计，并正确有效地描述、分析和验证系统的信息流策略，本文还通过通信顺序进程（CSP）[10-11]来实现系统设计的形式化描述和策略的语义表述. CSP 用一组通信进程来描述系统行为，能够准确地描述系统的执行轨迹，这正是描述行为不干扰属性所需要的. 本文给出基于 CSP 的形式化描述系统设计的方法，用 CSP 诠释 ENISM 中判定系统安全性条件的语义，并使用 CSP 验证工具 FDR2① 针对操作系统中的引用监视器设计的实例演示策略制定、设计描述、信息流分析、自动化验证的全过程.

本文第 2 节给出扩展不干扰模型 ENISM 的定义，提出执行不干扰和存储不干扰的概念，并在此基础上给出基于单步状态的安全展开条件（unwinding conditions），即设计中存在违背信息流策略的执行

① FDR2 用户手册. http://www.fsel.com/documentation/fdr2/html/index.html

轨迹或数据流动的充分条件(ENISM-CC)；第 3 节基于通信顺序进程 CSP 给出系统设计的形式化描述方法和 ENISM-CC 的语义，并给出形式化分析和验证信息流策略的方法；第 4 节以引用监视器为例，说明 ENISM 的有效性和形式化验证方法的可行性.

2　扩展不干扰模型

2.1　扩展不干扰模型的定义

本文在研究信息域不干扰理论的基础上，提出了扩展不干扰模型 ENISM. 该模型将执行域中即将执行的动作序列和执行域主体可观测的存储信息一同作为考察执行域完整性的对象. 因此，除保留传统不干扰模型所包含的执行域、状态、值域、状态转换函数、取值函数等元素外，还细化了执行域的定义，增加了对执行域上行为集合和数据存储集合的描述，并将输入数据作为输入动作的一个属性引入到定义中，以方便描述输入数据的不同对执行域可观测的存储值和执行序列的影响. 该模型还定义了执行域上某个执行状态之前和之后的执行序列，通过描述已执行和即将执行的动作序列来推测可能存在的干扰.

下面首先用状态转换机来描述一个确定系统，然后针对给定的不干扰策略给出系统信息流安全的定义以及单步展开条件.

定义 1.　系统 M 包括以下元素：

系统中所有的数据对象的集合 O；

系统中的数据对象可能取值的集合 V；

系统行为集合 A. 系统行为 $a_x \in A$，x 描述了动作 a 的输入值，以区分同一动作接受的不同输入. 如果 a 没有输入，则 x 取值 ε；

系统状态集合 S，系统的初始状态记为 s_0；

系统执行域集合 D，一个域 $u \in D$ 包含了可在这个域上执行的行为集合 u_I 和这个域上可观测的数据存储的集合 u_D. 且 D 上存在关系 $\bigcup$，有 $\forall u, v \in D, u \bigcup v \in D$；

行为可执行判断函数 $enabled: S \times A \to Boolean$，$enabled(s, a_x)$ 表示在状态 s 下动作 a_x 是否可执行；

单步状态转换函数 $step: S \times A \to S$. 如果 $enabled(s, a_x) = \text{true}$，则在状态 s 下执行动作 a_x 后系统由状态 s 转化为状态 $step(s, a_x)$；

行为执行域函数 $dom: A \to D$，$dom(a_x)$ 表示了行为 a_x 所在的执行域；

信息域取值函数 $values: S \times 2^O \to 2^V$. $\forall Objs \subset O$，$values(s, Objs)$ 表示在状态 s 下，数据存储集合 $Objs$ 中元素对应取值的集合；

系统的行为执行轨迹集合 T_s. T_s 记录了系统由状态 s 开始到任意时刻为止的所有可能行为序列的集合；$T_{s,u}$ 则表示了 T_s 在域 u 上的行为序列的集合. 且有 $\varepsilon \in T_{s,u}$；$a_x \circ \alpha \in T_{s,u}$，$\varepsilon$ 表示空串，$\circ$ 表示行为或行为序列的复合运算符，$a_x \circ \alpha$ 表示先执行行为 a_x，然后执行行为序列 α；

系统已执行的行为轨迹 RT_s. RT_s 记录了系统由初始状态 s_0 转化到状态 s 过程中的行为序列；$RT_{s,u}$ 则表示了 RT_s 在域 u 上的行为序列；

系统执行函数 $run: S \times A^* \to S$，$\alpha \in T_s$ 时，$run(s, \alpha)$ 有定义，并且满足 $run(s, \varepsilon) = s$，$run(s, a_x \circ \alpha) = run(step(s, a_x), \alpha)$.

为了能够描述同一行为的不同输入，我们定义集合 A 上的关系 R. 对于 A 中的任意两个行为，如果它们是执行的同一段代码，则认为它们满足关系 R，显然关系 R 满足自反、对称和传递性. 基于关系 R 可以建立等价类集合 A/R，记为 RA，有定义 $\{[a_x]\} = RA = \{c \mid c \in A \wedge cRa_x\}$. 为描述方便将 $[a_x]$ 简记为 a，下文中会用 a 来描述可接受不同输入的一类 R 等价行为.

系统的信息流策略可以定义为集合 $Du = \{u_I, u_D \mid \forall u \in D\}$ 上的关系 $\rightsquigarrow$ 及其补关系 $\not\rightsquigarrow = Du \times Du \setminus \rightsquigarrow$. 并分别称 $\rightsquigarrow$ 和 $\not\rightsquigarrow$ 为干扰关系和不干扰关系. 对于任意的 $u, v \in D$，$u_D \rightsquigarrow v_I$，$u_I \rightsquigarrow v_I$ 反映了域 v 的行为执行可能受到域 u 的执行或可观测数据值的影响，而 $u_D \rightsquigarrow v_D$，$u_I \rightsquigarrow v_D$ 反映了域 v 可观测的数据值可能受到域 u 的执行或可观测数据值的影响.

下面依据给定的信息流策略 $\rightsquigarrow$ 定义系统的信息流安全. 在定义系统的信息流安全之前，首先定义系统的执行安全性和存储安全性，以分别描述满足信息流策略的行为和信息流动.

定义 2.　执行安全性. 给定系统 M 以及安全策略 $\rightsquigarrow$，如果 $\forall v \in D, \alpha \in T_{s_0}$ 满足 $PurgeI(\alpha, v) \in T_{s_0} \wedge T_{run(s_0, \alpha), v} = T_{run(s_0, PurgeI(\alpha, v)), v}$，则称系统 M 满足执行安全性.

其中，提取函数 $PurgeI$ 从执行序列 $\alpha \in A^*$ 中去除所有不允许干扰 v_I 的行为，从而得到一个新的序列. 提取函数的定义如下：

$PurgeI(\varepsilon, v) = \varepsilon$；

$$PurgeI(a_x \circ \alpha, v) = \begin{cases} a_x \circ PurgeI(\alpha), & dom(a_x)_I \rightsquigarrow v_I \\ PurgeI(\alpha), & \text{其它} \end{cases}.$$

定义 2 说明，如果从任意可执行序列中去掉所有不允许干扰域 v 执行的行为后，余下的行为仍能构成一个可执行序列，且该序列的执行不影响域 v 的执行轨迹.

定义 3. 存储安全性. 给定系统 M 以及安全策略 $\rightsquigarrow$，$\forall v\in D,\alpha\in T_{s_0}$，满足 $PurgeD(\alpha,v)\in T_{s_0}\wedge values(run(s_0,\alpha),v)=values(run(s_0,PurgeD(\alpha,v)),v)$，则称系统 M 满足存储安全性.

$$SoursesD(\varepsilon,v)=\{v_D\},$$

$$SoursesD(a_x\circ\alpha,v)=\begin{cases}\{dom(a_x)_I\}\bigcup SoursesD(\alpha), & \exists u\in SoursesD(\alpha,v).\ dom(a_x)_I\rightsquigarrow v\wedge dom(a_x)_D\not\rightsquigarrow v\\ \{dom(a_x)_I\}\bigcup\{dom(a_x)_D\}\bigcup SoursesD(\alpha), & \exists u\in SoursesD(\alpha,v).\ dom(a_x)_I\rightsquigarrow v\wedge dom(a_x)_D\rightsquigarrow v.\\ SoursesD(\alpha), & \forall u\in SoursesD(\alpha,v).\ dom(a_x)_I\not\rightsquigarrow v\end{cases}$$

定义 3 说明，如果从任意可执行序列中去掉所有不允许干扰域 v 的可观测信息值存储的行为后，余下的行为仍能构成一个可执行序列，且该序列的执行不影响域 v 的可观测数据值. 从 $SoursesD$ 函数的定义可以看出，要保证提取之后的序列仍是可执行序列，不仅要保留允许改变域 v 的数据值的域行为，也要保留允许影响这些行为执行的其它域的行为.

由于对目标域执行轨迹集合的考察侧重于分析存储隐蔽通道，而对目标域数据值的考察则侧重于解决信道控制问题，所以从提取函数 $PurgeI$、$PurgeD$ 的定义可以看出，对目标域的行为集合的干扰策略支持传递性，而对目标域的数据集合的干扰策略则不支持传递性. 即对于任意的 $u,w\in Du$，$v\in D$，如果策略允许 $u\rightsquigarrow v_I$ 且 $v_I\rightsquigarrow w$，则策略也同样允许 $u\rightsquigarrow w$，但是即使策略允许 $u\rightsquigarrow v_D$ 且 $v_D\rightsquigarrow w$，策略也不一定允许 $u\rightsquigarrow w$.

定义 4. 对于给定的系统 M 以及安全策略 $\rightsquigarrow$，如果对于 M 中的任意安全域 $u\in D$，同时满足执行安全性和存储安全性，则称系统 M 满足安全策略 $\rightsquigarrow$.

定义 4 描述了如果系统 M 中不存在违背安全策略 $\rightsquigarrow$ 的信息流动和不被安全策略 $\rightsquigarrow$ 允许的执行轨迹，则可以说系统 M 是安全的. 因此，定义 2 和定义 3 给出的执行安全性和存储安全性条件就是系统 M 满足安全策略 $\rightsquigarrow$ 的条件.

为了便于系统验证，下文将给出并证明只涉及单步状态改变的展开条件.

定理 1. ENISM 单步展开定理(ENISM-CC). 对于给定的系统 M 和安全策略 $\rightsquigarrow$，对于 $\forall s,t\in S$，$v\in D,a_x\in A$，如果存在状态等价关系 $\overset{v_I}{\sim}$，$\overset{v_D}{\sim}$ 满足下列条件，则系统 M 满足安全策略 $\rightsquigarrow$.

其中，提取函数 $PurgeI$ 从执行序列 $\alpha\in A^*$ 中去除所有不允许干扰 v_I 的行为，从而得到一个新的序列. 提取函数的定义如下：

$$PurgeI(\varepsilon,v)=\varepsilon;$$

$$PurgeI(a_x\circ\alpha,v)=\begin{cases}a_x\circ PurgeI(\alpha), & dom(a_x)_I v_I\\ PurgeI(\alpha), & \text{其它}\end{cases},$$

其中，$SoursesD$ 函数计算由所有允许干扰 v_D 的执行域构成的集合

(i) $s\overset{v_I}{\sim}t\Rightarrow T_{s,v}=T_{t,v}$;

(ii) $s\overset{v_I}{\sim}t\wedge enabled(s,a_x)\wedge dom(a_x)_I\rightsquigarrow v_I\wedge RT_{s,TB}=RT_{t,TB}\Rightarrow enabled(t,a_x)\wedge step(s,a_x)\overset{v_I}{\sim}step(t,a_x)$;$(TB=\{\bigcup u\mid\forall u\in D,\ u_I\rightsquigarrow v_I\})$;

(iii) $(dom(a_x)_I\not\rightsquigarrow v_I\wedge enabled(s,a_x)\Rightarrow s\overset{v_I}{\sim}step(s,a_x)$;

(iv) $s\overset{v_D}{\sim}t\Rightarrow values(s,v)=values(t,v)$;

(v) $s\overset{v_D}{\sim}t\wedge enabled(s,a_x)\wedge enabled(t,a_x)\wedge(dom(a_x)_D\rightsquigarrow v_D\rightarrow s\overset{dom(a_x)_D}{\sim}t)\Rightarrow step(s,a_x)\overset{v_D}{\sim}step(t,a_x)$;

(vi) $(dom(a_x)_I\not\rightsquigarrow u_D\wedge enabled(s,a_x)\Rightarrow s\overset{v_D}{\sim}step(s,a_x)$;

$s\overset{v_I}{\sim}t$ 的实际意义表示：对于域 v 而言，在状态 s 和 t 下可执行轨迹是完全相同的，简记为执行等价. 同样 $s\overset{v_D}{\sim}t$ 的实际意义表示：对于域 v 而言，在状态 s 和 t 下可观察到的信息值是完全相同的，简记为存储等价. 条件(i)和(iv)表示：若两状态下 v 满足执行等价或存储等价，则在这两状态下可执行的轨迹集合或可观测的信息值集合相同；条件(ii)表示若两状态下 v 满足执行等价，且在这两个状态下，所有允许干扰 v_I 的执行域上已执行了相同的轨迹，则两状态下在上述域上可执行相同的行为 a_x，且执行 a_x 后 v 上仍满足执行等价；条件(v)表示若两状态下 v 满足存储等价，且可执行相同的行为 a_x，如果域 $dom(a_x)$ 在这两状态下也是存储等价的，则执行 a_x 后 v 上仍满足存储等价；条件(iii)(vi)表示，执行不

允许干扰 v_D 或 v_I 的行为后达到的状态，与执行前的状态在 v_D 或 v_I 上是等价的.

证明.　证明单步条件能够推导出执行安全性和存储安全性条件. 即

(i)(ii)(iii)(iv)(v)(vi)⇒

$(\forall v\in D,\alpha\in T_{s_0}\rightarrow((PurgeI(\alpha,v)\in T_{s_0}\wedge T_{run(s_0,\alpha),v}=T_{run(s_0,PurgeI(\alpha,v)),v}))\wedge(PurgeD(\alpha,v)\in T_{s_0}\wedge values(run(s_0,\alpha),v)=values(run(s_0,PurgeD(\alpha,v)),v)))$.

(1) 证明单步条件能推出执行安全性.

由 $s_0\overset{v_I}{\sim}s_0, RT_{s_0,TB}=RT_{s_0,TB}$ 知，如果能够证明

$\forall v\in D,s,t\in S.RT_{s,TB}=RT_{t,TB}\wedge s\overset{v_I}{\sim}t\wedge\alpha\in T_s\Rightarrow$

$PurgeI(\alpha,v)\in T_t\wedge T_{run(s,\alpha),v}=T_{run(t,PurgeI(\alpha,v)),v}$

则命题得证.　　证毕.

对 α 的长度进行归纳证明.

① 当 $\alpha=\varepsilon$ 时，命题显然成立.

② 假设 α 使命题成立，现证明 $a\circ\alpha$ 也同样使命题成立.

假设

$$RT_{s,TB}=RT_{t,TB}\wedge s\overset{v_I}{\sim}t\wedge a\circ\alpha\in T_s \tag{1}$$

对 a 分情况讨论：

a) 若 $dom(a)_I\nrightarrow v_I$，则

$$PurgeI(a\circ\alpha,v)=PurgeI(\alpha,v) \tag{2}$$

又由条件(iii)得

$$s\overset{v_I}{\sim}step(s,a) \tag{3}$$

所以

$$step(s,a)\overset{v_I}{\sim}t \tag{4}$$

又由 $dom(a)_I\nrightarrow v_I$ 知

$$RT_{step(s,a),TB}=RT_{s,TB}$$

则

$$RT_{step(s,a),TB}=RT_{t,TB},\alpha\in T_{step(s,a)} \tag{5}$$

于是由归纳假设和式(4)、(5)知

$$PurgeI(\alpha,v)\in T_t\wedge T_{run(step(s,a),a),v}=T_{run(t,PurgeI(\alpha,v)),v} \tag{6}$$

于是由式(2)和 run、$step$ 的定义知

$PurgeI(a\circ\alpha,v)\in T_t\wedge T_{run(s,a\circ a),v}=T_{run(t,PurgeI(a\circ\alpha,v)),v}$.

b) 若 $dom(a)_I\rightsquigarrow v_I$，则

$$PurgeI(a\circ\alpha,v)=a\circ PurgeI(\alpha,v) \tag{7}$$

由式(1)和条件(ii)得

$$enabled(t,a),step(s,a)\overset{v_I}{\sim}step(t,a) \tag{8}$$

又由式(1)和 RT 的定义得

$$RT_{step(s,a),TB}=RT_{step(t,a),TB}\wedge\alpha\in T_{step(s,a)} \tag{9}$$

则由归纳假设和式(8)、(9)得

$$PurgeI(\alpha,v)\in T_{step(t,a)}\wedge T_{run(step(s,a),a),v}=T_{run(step(t,a),PurgeI(\alpha,v)),v}$$

由 run、$step$ 的定义知

$$a\circ PurgeI(\alpha,v)\in T_t\wedge T_{run(s,a\circ\alpha),v}=T_{run(t,a\circ PurgeI(\alpha,v)),v}$$

由式(7)得

$$PurgeI(a\circ\alpha,v)\in T_t\wedge T_{run(s,a\circ\alpha),v}=T_{run(t,PurgeI(a\circ\alpha,v)),v} \tag{10}$$

综上所述，由①、②的结论可推出执行安全性条件.

(2) 证明单步条件能推出存储安全性

为证明方便，记

$$s\overset{SourcesD(\alpha,v)}{\sim}t\overset{\text{def}}{=}(\forall w\in SourcesD(\alpha,v).s\overset{w}{\sim}t),$$

则由

$$s_0\overset{SourcesD(\alpha,v)}{\sim}s_0,\ RT_{s_0,SD}=RT_{s_0,SD},$$

其中 $SD=\{\bigcup u\mid\forall u,w\in D,w_I\in SourcesD(\alpha,v),u_I\rightsquigarrow v_I\}$ 知，如果能够证明

$\forall v\in D,s,t\in S.RT_{s,SD}=RT_{t,SD}\wedge s\overset{SourcesD(\alpha,v)}{\sim}t\wedge\alpha\in T_s\Rightarrow$

$PurgeD(\alpha,v)\in T_t\wedge values(run(s,\alpha),v)=values(run(t,PurgeD(\alpha,v)),v)$,

则命题得证.　　证毕.

对 α 的长度进行归纳证明.

① 当 $\alpha=\varepsilon$ 时，命题显然成立.

② 假设 α 使命题成立，现证明 $a\circ\alpha$ 同样使命题成立.

假设

$$s\overset{SourcesD(a\circ\alpha,v)}{\sim}t\wedge a\circ\alpha\in T_s \tag{11}$$

对 a 分情况讨论：

a) 若 $dom(a)_I\notin SourcesD(a\circ\alpha,v)$，则

$$PurgeD(a\circ\alpha,v)=PurgeD(\alpha,v) \tag{12}$$

又由条件(iii)(vi)及 $SourcesD$ 函数的定义得

$$s\overset{SourcesD(\alpha,v)}{\sim}step(s,a) \tag{13}$$

所以

$$step(s,a)\overset{SourcesD(\alpha,v)}{\sim}t \tag{14}$$

又由假设条件(11)知

$$\alpha\in T_{step(s,a)} \tag{15}$$

于是由归纳假设和式(14)、(15)知

$$PurgeD(\alpha,v)\in T_t\wedge values(run(step(s,a),\alpha),v)=values(run(t,PurgeD(\alpha,v)),v),$$

于是由式(12)和 run、$step$ 的定义知

$$PurgeD(a\circ\alpha,v)\in T_t\wedge values(run(s,a\circ\alpha),v)=$$

$values(run(t,PurgeD(a\circ\alpha,v)),v)$.

b) 若 $dom(a)_I\in SourcesD(a\circ\alpha,v)$，则

$$PurgeD(a\circ\alpha,v)=a\circ PurgeD(\alpha,v) \quad (16)$$

由条件(ii)、(v)及假设 $RT_{s,SD}=RT_{t,SD}$ 得

$$enabled(t,a),step(s,a)\overset{SourcesD(\alpha,v)}{\sim}step(t,a) \quad (17)$$

又由假设条件(11)知

$$\alpha\in T_{step(s,a)} \quad (18)$$

由归纳假设和式(17)、(18)知

$PurgeD(\alpha,v)\in T_{step(t,a)}\wedge values(run(step(s,a),\alpha),v)=values(run(step(t,a),PurgeD(\alpha,v)),v)$

由 run、$step$ 的定义知

$a\circ PurgeD(\alpha,v)\in T_t\wedge values(run(s,a\circ\alpha),v)=values(run(t,a\circ PurgeD(\alpha,v)),v)$.

由式(16)得

$PurgeD(a\circ\alpha,v)\in T_t\wedge values(run(s,a\circ\alpha),v)=values(run(t,PurgeD(a\circ\alpha,v)),v)$.

综上所述，由①、②的结论可推出存储安全性. 则由定义 4 知命题得证. 证毕.

ENISM 不仅可用于描述和分析操作系统的信息流策略，更可以用于类似的具有模块化结构的复杂系统的信息流分析和描述. 但是，根据不同系统的特点，描述和验证的方法会有所差别，下文将主要针对操作系统的描述和信息流策略的验证给出具体的形式化方法.

3 基于 CSP 的系统描述和信息流策略验证方法

Rushby 在文献[3]中通过考察信息域上的主体对信息值的观察(observe)或修改(alter)能力来描述信息的流动，并通过检查是否存在恶意主体拥有不被允许的能力来查找违反安全策略的信息流动. 但是，信息流动不仅仅反映在数据值的传递上，还反映在一个域的行为执行和信息改变对另一个域上的行为执行的影响上，如影响到上节单步展开条件中描述的 $T_{s,u}$. 而行为序列集合 $T_{s,u}$ 可能包含多个甚至无限多个连续行为，验证行为序列集合的改变无法使用 Rushby 的方法来实现. 本文通过通信顺序进程(CSP)[10]来形式化描述操作系统的设计和安全状态. CSP 用一组通信进程来描述系统行为，能够描述系统的执行轨迹，这正是描述行为不干扰属性所需要的. 目前，已有许多支持 CSP 的自动化验证工具，可辅助我们进行自动化的系统验证. 本节即将给出基于 CSP 的形式化描述系统设计的方法，并用 CSP 诠释系统 M 满足安全策略 $\rightsquigarrow$ 的条件，为进一步利用 CSP 验证工具 FDR2 进行形式化验证做准备.

为便于理解，先将下文中用到的通信顺序进程(CSP)的部分算子列出如下：

P	一个进程
αP	P 的子母表
$a\rightarrow P$	前缀算子，if a then P
$(a\rightarrow P\mid b\rightarrow Q)$	选择算子，if a then P else if b then Q
P/s	后继算子，P after s
$P\parallel Q$	并发算子，P in parallel with Q
$P\backslash C$	屏蔽算子，P with C hiding
$P\Gamma C$	约束算子，s restricted to A
$P\Box Q$	不确定算子，P or Q
$traces(P)$	P 可能执行的迹的集合
$failures(P)$	P 当前不接受的动作集合

虽然，随着硬件技术和操作系统实现技术的发展，操作系统具有了更复杂的功能，支持对更多资源设备的管理. 但是，如果我们不考虑信息的载体和获取信息的方式，总可以把信息抽象为对象，把信息的接收、读取、传送抽象为读、写操作. 于是，在描述一个操作系统设计时，除使用标准的 CSP 算子和定义外，增加定义以下描述规则.

定义 5. 相关定义和描述规则如下.

(1) 对应于 ENISM 中的执行域 u，进程 U 描述域 u 上的执行轨迹集合，并将进程的集合定义为 D_P；将数据存储或外部输入输出设备描述为 $d\in u_D$，读入或写出的系统内变量描述为 $v.U$；将非读写操作 $c\in u_I$，描述为 $c.U$；将读写操作描述为 $d.read.v.U$ 或 $d.write.v.U$；规定每一个读写操作只能描述对一个数据存储或外部输入输出设备的读或写. 系统临时变量之间的值传递也必须描述为通过某个外部存储传递.

(2) 对于给定系统 M 上的任意进程 U 和任意迹 $tr\in Traces(M)$，定义 $interferedSet(tr,U)\subset\alpha M$ 为执行迹 tr 在执行 U 上操作时读取的数据所能影响的系统变量集合. 定义 $Acts(tr)\subset A$ 为 tr 中操作的集合，则计算 $interferedSet(tr,U)$：① $\forall d.read.v.U\in Acts(tr)$，$v.U\in interferedSe(tr,U)$；② $\forall d\in\alpha M$，若 $V,W\in D_P$，$d.write.v.V\in\alpha V$，$d.write.w.W\in\alpha W$，$v.V\in interferedSet(tr,U)$，则 $w.W\in interferedSet(tr,U)$. 由此可计算 U 的所有输入数据所能影响到的系统 M 的变量集合为 $\{a\mid a\in interferedSet(tr,U),\ tr\in Traces(M)\}$，记为 βU.

(3)将选择语句中的分支选择条件描述为一组操作,分别作为各分支的起始标记.如

$tr0\rightarrow$if $cond_1(v)$ then tr_1 else if $cond_2(v)$ then tr_2 else if $cond_n(v)\cdots$ else tr_n,可描述为 $tr_0\rightarrow((v.U.ct_1.m_1\rightarrow tr_1)\mid(v.U.ct_1.m_2\rightarrow tr_2)\mid\cdots\mid(v.U.ct_1.m_n\rightarrow tr_n))$,其中 $ct_i(i=1,\cdots,p)$,$m_i(i=1,2,\cdots,n)$为不同的自然数,ct_i用于区分系统中不同的选择语句,m_i用于区分选择语句中的不同分支,p描述系统中选择语句的数量,n描述每一个选择语句的分支数.上例中 $v.U.ct_1.m_2$表示这是描述过程中遇到的第 1 个选择语句的第 2 个分支.定义由任一选择语句中的分支起始标记构成的集合为$E_{cti}(i=1,2,\cdots,p)$,由$E_{cti}(i=1,2,\cdots,p)$构成的集合为$E=\{E_{cti}\mid i=1,2,\cdots,p\}$,上例中$E_{cti}=\{v.U.ct_i.m_1,v.U.ct_i.m_2,\cdots,v.U.ct_i.m_n\}$.

(4)函数 $GetChoiceActs:T\times D_P\rightarrow 2^E$,$GetChoiceActs(tr,U)=\{E_{cti}\mid\exists x.ct_i.m_j\in E_{cti},x\in interferedSet(tr,U)\}$,则函数 $GetChoiceActs$ 描述了 tr 执行 U 的操作时读取的数据所能影响到的选择语句集合.

(5)对于 E 的任意子集$e\subset E$,设$e=\{e_1,e_2,\cdots,e_n\}$,则定义函数 $Cartesian(e)=e_1\times e_2\times\cdots\times e_n$,描述 e 中所有元素的笛卡儿积.由于 e_i描述的是某个选择语句的所有分支起始标记的集合,$Cartesian(e)$则反映了 e 中涉及的多个选择语句的所有选择结果的组合.对于其中的任意一种选择结果 $\alpha\in Cartesian(e)$,设 $\alpha=\{\alpha_1,\alpha_2,\cdots,\alpha_n\}$,定义进程 $Choice(\alpha)=(\alpha_1\mid\alpha_2\mid\cdots\mid\alpha_n)\rightarrow Choice(\alpha)$,$Choice(\alpha)$可将分支选择结果反映在与其同步的进程中.

(6)对于任意执行迹 $tr\in Traces(M)$,如果存在某个状态 s,是由初始状态s_0执行tr后达到的.则一定可以由tr的输入数据和执行轨迹确定 $GetChoiceActs(tr,M)$中的一部分选择语句的分支选择结果,即存在 $GetChoiceActs(tr,M)$的一个子集 e_s和 e_s上的一种选择组合 $\alpha_{s,tr,M}$,有 $\alpha_{s,tr,M}\in Cartesian(e_s)$,下文称 $\alpha_{s,tr,M}$为系统 M 由初始状态 s_0执行轨迹 tr 到状态 s 时的分支选择向量.

(7)$\forall d.read.v\in\alpha M$,记状态 s 下 d 的取值为 $getData(d,s)$,v 的取值为 $getParam(v,s)$.

(8)系统描述中遇到不确定选择 $P\square Q$,均表达为确定性选择关系$(a\rightarrow P\mid b\rightarrow Q)$,在不影响语义的前提下确保系统描述中不存在非确定性.

下面基于 CSP 和上述描述规则给出一种等价关系定义.

定义 6.　对于给定的系统 M 和安全策略$\rightsquigarrow$,任意状态 s,t,M 中的任意执行域 $w\in D$ 上进程的描述 W.如果由初始状态 s_0执行到状态 s 的执行轨迹为 $trs\in Traces(M)$,产生的分支选择向量为 $\boldsymbol{\alpha}_{s,trs,M}$,执行到状态 t 的执行轨迹为 $trt\in Traces(M)$,产生的分支选择向量为 $\boldsymbol{\alpha}_{t,trt,M}$,则有

$$s\overset{w_D}{\sim}t\ \text{iff}(\forall d.read.v\in\alpha M,getData(d,s)=getData(d,t),getParam(v,s)=getParam(v,t))$$

$$s\overset{w_I}{\sim}t\ \text{iff}(((M/trs)\parallel Choice(\boldsymbol{\alpha}_{s,trs,M}))\Gamma\alpha W=((M/trt)\parallel Choice(\boldsymbol{\alpha}_{t,trt,M}))\Gamma\alpha W).$$

定义 6 基于 CSP 给出了一种状态等价的定义.基于进程 W 的存储等价要求 W 可读取的数据存储和 W 中的变量值在两个状态下都必须是相等的;执行等价则由于引入了数据对执行的影响,不再简单地理解为$(M/trs)\Gamma\alpha W=(M/trt)\Gamma\alpha W$,还要考虑输入对分支选择的影响,即引入了 $Choice()$来体现分支选择的结果.只有 trs、trt 中的行为及输入值所影响的分支选择结果都不能改变域 W 上行为的执行时,才能认为状态 s,t 是执行等价的.

基于上述等价关系和 CSP 描述规则,我们给出目标系统满足不干扰策略的判定条件.

定理 2.　对于确定性系统 M 和安全策略$\rightsquigarrow$,如果对于 M 中的任意执行域 $w\in D$,存在定义 6 定义的等价关系$\overset{w_D}{\sim}$,$\overset{w_I}{\sim}$,并满足以下条件,则称系统 M 满足安全策略$\rightsquigarrow$.

(1)执行安全性条件

$\forall w\in D,u=\{\cup z\mid z_I\not\rightsquigarrow w_I\},v=\{\cup z\mid z_I\rightsquigarrow w_I\}$.

设 W,U,V 分别描述域 w,u,v 上的进程,$M=U\parallel V\parallel W$,则满足

$Assert(M\backslash\alpha U\backslash\beta U):[deterministic[FD]]=\text{true}$

(表明 $M\backslash\alpha U\backslash\beta U$ 中不存在非确定性).

(2)存储安全性条件

$\forall u\in D,u_I\not\rightsquigarrow w_D$,设 U,W 分别描述域 u,w 上的进程,且满足

$$\forall d.write\in\alpha U\Rightarrow d.read\notin\alpha W.$$

执行安全性条件要求 $M\backslash\alpha U\backslash\beta U$ 是确定的,即通过 $\alpha U,\beta U$ 反映 U 的执行和输入对其它进程的干扰,而存储安全性条件则主要关心两个域之间是否存在直接的信息传递.

证明.

(1)证明执行安全性条件,即证明满足定理 1 的(i)(ii)(iii).

① 由定义5和定义6知

$$((M/trs)\parallel Choice(\boldsymbol{\alpha}_{s,trs,M}))\Gamma\alpha W$$

即为 $T_{s,w}$,因此定理1(i)成立.

② 证明定理1的(ii)成立

由定义1知 $RT_{s,u}$ 是一组行为的执行序列,且有

$$RT_{s,\{v,w\}}=RT_{s,\{v,w\}}\Rightarrow trs\backslash\alpha U=trt\backslash\alpha U \quad (19)$$

由定理1(ii)的条件 $enabled(s,a_x)\wedge dom(a_x)_I\rightsquigarrow w_I$ 知,存在 $a\in\alpha M-\alpha U$,$(trs\hat{}\langle a\rangle)\in traces(M)$.

由 $M\backslash\alpha U\backslash\beta U$ 的确定性知

$$(trs\hat{}\langle a\rangle)\backslash\alpha U\backslash\beta U\in traces(M\backslash\alpha U\backslash\beta U) \quad (20)$$

假设

$$(trt,\{a\})\in failures(M) \quad (21)$$

如果 $a\notin\beta U$,则同式(20)有

$$(trt\backslash\alpha U\backslash\beta U,\{a\})\in failures(M\backslash\alpha U\backslash\beta U)$$

即由式(19)得

$$(trs\backslash\alpha U\backslash\beta U,\{a\})\in failures(M\backslash\alpha U\backslash\beta U) \quad (22)$$

根据式(20)和(22)知这与 $M\backslash\alpha U\backslash\beta U$ 的确定性矛盾.

如果 $a\in\beta U$,则 a 所在的选择语句必至少还存在另一个分支选择行为 $b\in\beta U$,满足 $(trt\hat{}\langle b\rangle)\in traces(M\parallel Choice(Choice(\boldsymbol{\alpha}_{t,trt,M})))$,因为 a 不是 U 进程上的分支选择,所以 a,b 所在分支在 U 进程以外存在分歧.不妨设存在行为 $c\in\alpha M-\alpha U-\beta U$,行为序列 trx 满足

$$trx\in traces(M\backslash(trs\hat{}\langle a\rangle));$$

$$trx\in traces(M\backslash(trt\hat{}\langle b\rangle));$$

$$(trx\hat{}\langle c\rangle)\in traces(M\backslash(trs\hat{}\langle a\rangle));$$

$$(trx,\{c\})\in failures(M\backslash(trt\hat{}\langle b\rangle));$$

即 $(trs\hat{}\langle a\rangle\hat{}trx\hat{}\langle c\rangle)\in traces(M)$;

$(trs\hat{}\langle b\rangle\hat{}trx,\{c\})\in failures(M)$;

于是有

$$(trs\hat{}\langle a\rangle\hat{}trx\hat{}\langle c\rangle\backslash\alpha U\backslash\beta U)\in traces(M\backslash\alpha U\backslash\beta U)$$

$$(trs\hat{}\langle b\rangle\hat{}trx\backslash\alpha U\backslash\beta U,\{c\})\in failures(M\backslash\alpha U\backslash\beta U);$$

由式(19)和 $a,b\in\beta U$ 知

$$(trs\hat{}trx\hat{}\langle c\rangle\backslash\alpha U\backslash\beta U)\in traces(M\backslash\alpha U\backslash\beta U),$$

$$(trs\hat{}trx\backslash\alpha U\backslash\beta U,\{c\})\in failures(M\backslash\alpha U\backslash\beta U);$$

这与 $M\backslash\alpha U\backslash\beta U$ 的确定性矛盾.即式(21)假设不成立,有

$$enabled(t,a_x) \quad (23)$$

最后用反证法证明 $step(s,a_x)\overset{w_I}{\sim}step(t,a_x)$ 即假设

$$((M/(trs\hat{}\langle a\rangle))\parallel Choice(\boldsymbol{\alpha}_{step(s,a),trs\hat{}\langle a\rangle,M}))\Gamma\alpha W\neq((M/(trt\hat{}\langle a\rangle))\parallel Choice(\boldsymbol{\alpha}_{step(t,a),trt\hat{}\langle a\rangle,M}))\Gamma\alpha W \quad (24)$$

则一定有

$$((M/(trs\hat{}\langle a\rangle))\parallel Choice(\boldsymbol{\alpha}_{step(s,a),trs\hat{}\langle a\rangle,M}))\backslash\alpha U\backslash\beta U\neq((M/(trt\hat{}\langle a\rangle))\parallel Choice(\boldsymbol{\alpha}_{step(t,a),trt\hat{}\langle a\rangle,M}))\backslash\alpha U\backslash\beta U,$$

即存在不相等的执行序列,不妨设存在

$$trx\in traces(((M/(trs\hat{}\langle a\rangle))\parallel Choice(\boldsymbol{\alpha}_{step(s,a),trs\hat{}\langle a\rangle,M}))\backslash\alpha U\backslash\beta U),$$

$$c\in\alpha M-\alpha U-\beta U,$$

满足

$$(trx\hat{}\langle c\rangle)\in traces(((M/(trs\hat{}\langle a\rangle))\parallel Choice(\boldsymbol{\alpha}_{step(s,a),trs\hat{}\langle a\rangle,M}))\backslash\alpha U\backslash\beta U)$$

$$(trx,\{c\})\in failures(((M/(trs\hat{}\langle a\rangle))\parallel Choice(\boldsymbol{\alpha}_{step(s,a),trs\hat{}\langle a\rangle,M}))\backslash\alpha U\backslash\beta U),$$

同式(23)的证明过程可知,这与 $M\backslash\alpha U\backslash\beta U$ 的确定性矛盾.即式(24)的假设不成立,即

$$step(s,a_x)\overset{w_I}{\sim}step(t,a_x) \quad (25)$$

综合式(23)、(25)可得,系统 M 满足定理1的(ii).

③ 证明此条件满足定理1的(iii).

下面同样用反证法证明 $s\overset{w_I}{\sim}step(s,a_x)$,假设

$$((M/trs)\parallel Choice(\boldsymbol{\alpha}_{s,trs,M}))\Gamma\alpha W\neq((M/(trs\hat{}\langle a\rangle))\parallel Choice(\boldsymbol{\alpha}_{step(s,a),trs\hat{}\langle a\rangle,M}))\Gamma\alpha W \quad (26)$$

则同②的证明过程可得到该假设同样与 $M\backslash\alpha U\backslash\beta U$ 的确定性矛盾,即假设错误,$s\overset{w_I}{\sim}step(s,a_x)$ 得证.即满足定理1的(iii).

综上所述,系统 M 满足定理1的条件(i)(ii)(iii),即系统 M 满足安全策略 $\rightsquigarrow$.

(2)证明存储安全性条件,即证明满足定理1的(iv)(v)(vi)

① 由定义5和定义6知 $\forall d.read.v\in\alpha W$,$getData(d,s)$,$getParam(v,s)$ 即为 $values(s,v)$,因此定理1(iv)成立.

② 由定理1(v)知,对于任意两个状态 s,t,若 $s\overset{w_D}{\sim}t\wedge enabled(s,a_x)\wedge enabled(t,a_x)\wedge(dom(a_x)_D\rightsquigarrow w_D\rightarrow s\overset{dom(a_x)_D}{\sim}t)$,如果 a_x 不是写操作,显然满足(v);否则不妨设 $a_x=d.write.v.*$.

a)如果 a_x 在不干扰 w 的域中执行,即 $d.write.v.*\in\alpha U$,根据数据完整性条件 $\forall d.write\in\alpha U\Rightarrow d.read\notin\alpha W$ 知,$d.read\notin\alpha W$,即 d 的改变不会被域 w 所观察到,仍有

$$values(step(s,a_x),w)=values(step(t,a_x),w),$$

即满足(v).

b) 如果a_x在干扰w的域中执行，即$d.write.v.* \notin \alpha U$，则有 $dom(a_x)_D \rightsquigarrow w_D$. 根据定理 1(v)的条件知 $s \overset{dom(a_x)_D}{\sim} t$，即$getParam(v,s)=getParam(v,t)$，于是如果有 $d.read.v'.* \in \alpha W$，则在执行动作 $d.write.v.*$ 后，域 w 上有

$getData(d,step(s,a_x))=getData(d,step(t,a_x))$，而对于域 w 上可观察的其它存储或变量不会因动作 $d.write.v.*$ 的执行而改变，因此满足(v).

③ 如果 $dom(a_x)_I \not\rightsquigarrow w_D$，即 $d.write.v.* \in \alpha U$，根据数据完整性条件 $\forall d.write \in \alpha U \Rightarrow d.read \notin \alpha W$ 知，$d.read \notin \alpha W$，即 d 的改变不会被域 w 所观察到，仍有

$\forall d'.read.v'.* \in \alpha W$，

$getData(d',s)=getData(d',step(s,a_x))$，

$getParam(v',s)=getParam(v',step(s,a_x))$，

即 $values(s,w)=values(step(s,a_x),w)$，满足定理 1 的(vi).

综上所述，系统 M 满足定理 1 的条件(iv)(v)(vi).

因此，根据(1)、(2)的证明结论知，系统 M 满足安全策略 $\rightsquigarrow$.　　证毕.

下节主要通过操作系统中的一个实例来说明如何通过上述方案来描述系统设计以及分析和验证系统是否满足给定的信息流策略.

4　实例研究：引用监控器设计的安全分析与策略验证

本文所研究的扩展不干扰模型(ENISM)和基于通信顺序进程(CSP)的描述和验证方法是作者参与的“八六三”课题“分布式可信计算系统研究”中的研究成果之一，被用在对操作系统的安全服务器、引用监视器以及操作系统微内核等关键模块的正确性、完整性、不可旁路性的验证上. 本节以引用监控器的设计为例，验证与引用监控器的完整性和不可旁路性相关的信息流策略，即借助形式化验证工具 FDR2 来验证引用监控器是否满足定理 2 的两个条件. 通过该实例说明 ENISM 的正确性、有效性和可用性. 验证的过程主要分为以下几个步骤：

(1) 系统功能分析和安全域划分.

该实例是一个有关安全操作系统中的引用监控器的实例. 任何用户要访问文件系统都必须经过引用监控器来进行访问控制. 如图 1 所示，安全服务器用于存放主客体的安全标识、修改安全策略以及协助引用监控器进行访问控制决策. 引用监控器是访问控制的执行单元，安全服务器才是真正的决策单元，模块交互图如下.

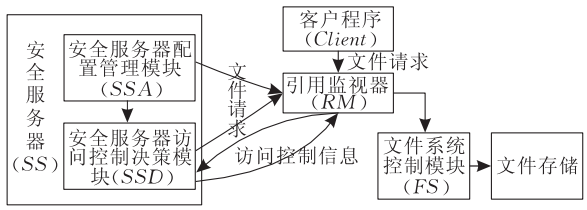

图 1　模块交互图

安全服务器被分为配置管理模块和决策模块. 配置管理模块负责对策略文件的维护和修改，需要访问文件系统读取策略文件. 决策模块负责对应用程序的访问请求进行权限判断，也需要通过文件系统读取策略文件. 应用程序同样需要通过文件系统访问资源. 文件系统的入口是引用监控器，所有请求都必须首先经过引用监控器，引用监控器通过向安全服务器决策模块请求决策来决定是否执行文件操作.

(2) 用 CSP 描述各模块的执行过程和数据资源

下面我们用 CSP 来描述各模块的执行过程(以下为 FDR2 中源码的部分截图)：

(a) 引用监控器的描述

引用监视器(RM)接收请求访问文件系统的进程信息 $Pinfo$、要访问的文件信息，如果是安全服务器(SS)的请求(包括配置模块(SSA)和决策模块(SSD))，则通知文件系统 FS 执行，否则向 SSD 发送 $permQuery$，询问请求进程是否有访问该文件的权限. 如果 SSD 返回 Yes，则通知 FS 执行，否则向请求者返回拒绝信息 $reject$.

```
RM=accqueryPinfo.read.Pinfo.RMn->accquerycont.read.accFile.RMn->
( ( xPinfo.RMn.Admin->accessFS.write.accFile.RMn ->accFSResult.read.FileResult.RMn
      ->accqueryResult.write.FileResult.RMn->RM )
  [] ( xPinfo.RMn.ClientInfo->querydb.read.permQuery.RMn
      ->checkquery.write.permQuery.RMn->checkresult.read.result.RMn
      ->( (xresult.RMn.Yes->accessFS.write.accFile.RMn->accFSResult.read.FileResult.RMn
          ->accqueryResult.write.FileResult.RMn->RM )
        [](xresult.RMn.No->rejectdb.read.reject.RMn->accqueryResult.write.reject.RMn->RM)) ) )
```

(b)安全服务器的描述

安全服务器(*SS*)分为配置模块(*SSA*),决策模块(*SSD*),写策略模块(*SSDc*).

$$SS=SSA|||SSD|||SSDc.$$

SSA 接受管理员的命令向文件系统 *FS* 请求更新策略文件,请求被 *RM* 截获.更新完毕,会发送消息 *exchangeAD* 通知 *SSDc* 修改内存中的策略信息.

SSD 接受来自 *RM* 的权限决策请求 *permQuery*,并依据 *permQuery* 中的用户信息和请求操作查询策略库,得到决策结果,并将决策结果返回 *RM*.

```
SSA=inSSA.read.accFile.SSAn->Pinfodb.read.Pinfo.SSAn->accqueryPinfo.write.Pinfo.SSAn
    ->accquerycont.write.accFile.SSAn->accqueryResult.read.result.SSAn->
    ((xresult.SSAn.Yes->exchangeAD.write.result.SSAn->exchangeDA.read.policyInfo.SSAn->SSA)
    [](xresult.SSAn.No->showError.SSAn->SSA))

SSD=checkquery.read.permQuery.SSDn->policyFlash.read.policyCon.SSDn
                ->checkresult.write.f_permQuery_policyCon.SSDn->SSD

SSDc=exchangeAD.read.result.SSDn->inSSD.read.accFile.SSDn->
   accessFS.write.accFile.SSDn->accFSResult.read.policyInfo.SSDn->
   ((xpolicyInfo.SSDn.none->exchangeDA.write.policyInfo.SSDn->SSDc)
   [](xpolicyInfo.SSDn.other->policyFlash.write.policyInfo.SSDn->exchangeDA.write.policyInfo.SSDn->SSDc))
```

(c)客户程序的描述

客户程序 *Client* 接收用户输入,并向 *FS* 请求文件访问,文件请求被 *RM* 截获并处理.客户程序等待处理结果 *accqueryResult*.

```
Client=inClient.read.accFile.Clientn->Pinfodb.read.Pinfo.Clientn
    ->accqueryPinfo.write.Pinfo.Clientn->accquerycont.write.accFile.Clientn
    ->accqueryResult.read.result.Clientn->Client
```

(d)文件系统的描述

文件系统 *FS* 接收 *RM* 的指令 *accessFS*,执行文件操作,并向 *RM* 返回结果.

```
FS=accessFS.read.accFile.FSn->doAccFile.FSn->accFSResult.write.FileResult.FSn->FS
```

(e)消息传递进程的描述

实际系统中是由内核实现消息传递的,这里引入 *msgPass* 进程实现该功能.用于接收和转发其它进程之间的同步消息,以实现进程间同步.*msgPass* 描述的片断如下:

```
msgPass=((checkresult.write.f_permQuery_policyCon.SSDn->checkresult.read.result.RMn->msgPass)
    [](accqueryPinfo.write.Pinfo.SSDn->accqueryPinfo.read.Pinfo.RMn->msgPass)
    [](accquerycont.write.accFile.SSDn->accquerycont.read.accFile.RMn->msgPass)
    [](exchangeDA.write.policyInfo.SSDn->exchangeDA.read.policyInfo.SSAn->msgPass))
    []((accqueryPinfo.write.Pinfo.SSAn->accqueryPinfo.read.Pinfo.RMn->msgPass)
    [](accquerycont.write.accFile.SSAn->accquerycont.read.accFile.RMn->msgPass))
```

上述各模块并发执行,并通过 *msgPass* 同步,构成完整的系统 *M*,其描述如下:

$$M=(SSA|||SSD|||SSDc|||Client|||RM|||RM\ |||\ RM\ |||FS|||FS|||FS)[|aM|]msgPass.$$

上述 CSP 描述构成了系统的完整描述,以.csp 文件的形式提交给形式化验证工具 FDR2 进一步分析.

(3)提出安全目标,制定安全策略,安全策略反映为不干扰关系.

由于篇幅的限制,我们仅对引用监控器最敏感的不可旁路的安全目标进行验证.由于安全服务器配置模块 *SSA* 和用户模块 *Client* 都提供了对用户的接口,根据用户请求访问文件系统 *FS*,它们是 *RM* 主要监控的对象.因此,本例验证的目标是:

SSA 和 *Client* 不可以绕过 *RM* 直接干扰文件系统 *FS*.

验证目标描述成不干扰关系是:

$SSA_I \nrightarrow RM_I$; $Client_I \nrightarrow RM_I$; $SSA_I \rightsquigarrow RM_D$; $Client_I \rightsquigarrow RM_D$; $SSA_I \nrightarrow FS_I$; $Client_I \nrightarrow FS_I$; $SSA_I \nrightarrow FS_D$; $Client_I \nrightarrow FS_D$; $RM_I \rightsquigarrow FS_I$; $RM_I \rightsquigarrow FS_D$; $SSD_I \rightsquigarrow RM_I$; $SSD_I \rightsquigarrow RM_D$; $SSA_I \nrightarrow SSD_I$; $Client_I \nrightarrow SSD_I$; $SSA_I \nrightarrow SSD_D$; $Client_I \nrightarrow SSD_D$; $SSDc_I \nrightarrow RM_I$; $SSDc_I \rightsquigarrow RM_D$; $SSDc_I \nrightarrow FS_I$; $SSDc_I \nrightarrow FS_D$; $SSA_I \rightsquigarrow SSDc_I$; $SSA_I \rightsquigarrow SSDc_D$.

上述不干扰关系表明 *SSA*、*Client*、*SSDc* 只能向 *RM* 传递数据而不能改变其执行路径;*SSA*、*Client*、*SSDc* 既不能直接向 *SSD*、*FS* 传递数据也不能改变其执行路径;*RM* 可以与 *FS* 有直接的交互,甚至调用 *FS* 执行.

(4)根据定理 2(2)验证以下对数据存储的不干扰关系:

$SSA_I \nrightarrow FS_D$; $Client_I \nrightarrow FS_D$; $SSA_I \nrightarrow SSD_D$; $Client_I \nrightarrow SSD_D$; $SSDc_I \nrightarrow FS_D$.

验证过程如下：

SSA 写数据集合为{$accquerycont.write.accFile.SSAn$, $accqueryPinfo.write.Pinfo.SSAn$, $exchangeAD.write.result.SSAn$}；影响到的读数据集合为{$accquerycont.read.accFile.RMn$, $accqueryPinfo.read.Pinfo.RMn$, $exchangeAD.read.result.SSDn$}；

$Client$ 写数据集合为{$accquerycont.write.accFile.Clientn$, $accqueryPinfo.write.Pinfo.Clientn$}；影响到的读数据集合为{$accquerycont.read.accFile.RMn$, $accqueryPinfo.read.Pinfo.RMn$}；

$SSDc$ 写数据集合为{$accessFS.write.accFile.SSDn$, $exchangeDA.write.policyinfo.SSDn$, $policyFlash.write.policyInfo.SSDn$}；影响到的读数据集合为{$accessFS.read.accFile.FSn$, $exchangeDA.read.policyinfo.SSAn$}

显然满足 $SSA_I \nrightarrow FS_D$；$Client_I \nrightarrow FS_D$；$SSA_I \nrightarrow SSD_D$；$Client_I \nrightarrow SSD_D$，但是不满足 $SSDc_I \nrightarrow FS_D$，因为在 $SSDc$ 的读数据集合中存在 $accessFS.read.accFile.FSn$，即 $SSDc$ 绕过 RM 直接将数据写到 FS 中. 修改 $SSDc$ 如下.

```
SSDc=exchangeAD.read.result.SSDn->Pinfodb.read.Pinfo.SSDn->accqueryPinfo.write.Pinfo.SSDn
  ->inSSD.read.accFile.SSDn->accquerycont.write.accFile.SSDn
  ->accqueryResult.read.policyInfo.SSDn->
  ((xpolicyInfo.SSDn.none->exchangeDA.write.policyInfo.SSDn->SSDc)
  []((xpolicyInfo.SSDn.other->policyFlash.write.policyInfo.SSDn->exchangeDA.write.policyInfo.SSDn->SSDc))
```

修改后的 $SSDc$ 写数据集合为{$accquerycont.write.accFile.SSDn$, $accqueryPinfo.write.Pinfo.SSDn$, $exchangeDA.write.policyinfo.SSDn$, $policyFlash.write.policyInfo.SSDn$}；影响到的读数据集合为{$accquerycont.read.accFile.RMn$, $accqueryPinfo.read.Pinfo.RMn$, $exchangeDA.read.policyinfo.SSAn$}，即满足 $SSDc_I \nrightarrow FS_D$.

(5) 根据定理 2(1)借助 FDR2 来验证以下执行不干扰关系：

将上节得到的.CSP 源文件输入 FDR2 中，根据定理 2(1)可知要证明系统满足

$SSA_I \nrightarrow RM_I$；$Client_I \nrightarrow RM_I$；$SSA_I \nrightarrow FS_I$；$Client_I \nrightarrow FS_I$；$SSA_I \nrightarrow SSD_I$；$Client_I \nrightarrow SSD_I$；$SSDc_I \nrightarrow RM_I$；$SSDc_I \nrightarrow FS_I$

即要判定以下表达式的确定性(βU 在 FDR2 中写作 bU)：

checkM=M\aSSA\aClientaSSDc\bSSA\bClient\bSSDc

结果如图 2 所示.

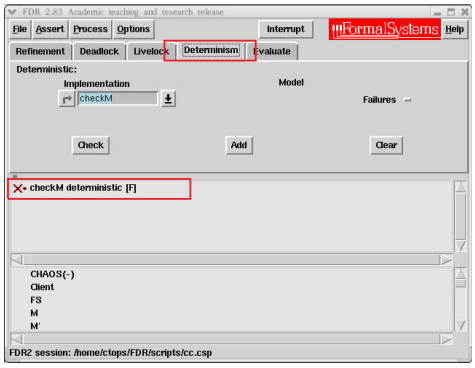

图 2　FDR2 验证结果图

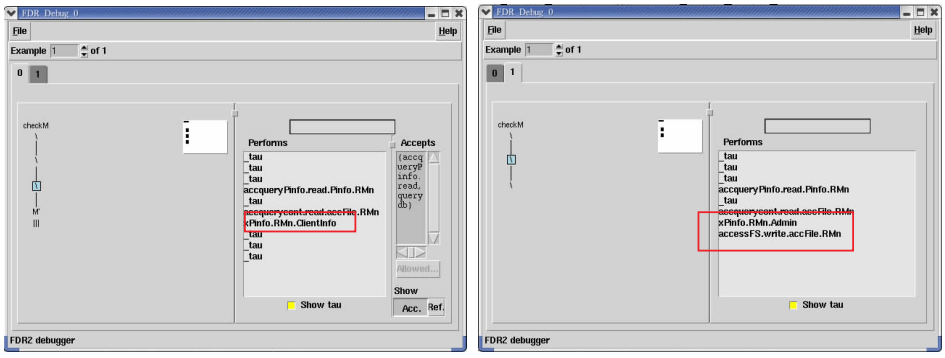

图 3　不确定轨迹分析

图 2 表明，表达式 $checkM$ 不满足确定性，图 3 显示了错误的原因. 图 3 的两幅截图显示了两条执行路径，第 2 幅中 RM 由于接收到 $pinfo$ 传递来的参数值为 $Admin$，则可以执行 $access.write.accFile.RMn$，而第一幅图则由于接收的 $pinfo$ 的参数值为 $Client$，不允许执行 $access.write.accFile.RMn$，从而产生不确定. 因此，产生不确定的原因是由于 SSA，$Clien$，$SSDc$ 之一传递给 RM 的数据值 $pinfo$ 可能干扰了 RM 的执行，即不满足 $SSA_I \nrightarrow RM_I$；$Client_I \nrightarrow RM_I$；$SSDc_I \nrightarrow RM_I$ 之一. 实际的意义是：

当 *SSA*,*Clien*,*SSDc* 之一被攻破而以管理员的角色访问 *RM* 时,*RM* 不会对该进程的访问请求做实质性的检查,即不会将请求送到 *SSD* 中验证,因此,该进程可以绕过 *RM* 执行任意文件访问操作.

修改设计的过程中,可以在 *RM* 中不区分执行进程是否以管理员的角色运行,对所有要求访问 *FS* 的请求都送到 *SSD* 中进行权限检查.修改后 *RM* 源码和结果图如图 4 所示.

```
RM=accqueryPinfo.read.Pinfo.RMn->accquerycont.read.accFile.RMn->querydb.read.permQuery.RMn
    ->checkquery.write.permQuery.RMn->checkresult.read.result.RMn
    ->( (xresult.RMn.Yes->accessFS.write.accFile.RMn->accFSResult.read.FileResult.RMn
        ->accqueryResult.write.FileResult.RMn->RM )
    [](xresult.RMn.No->rejectdb.read.reject.RMn->accqueryResult.write.reject.RMn->RM))
```

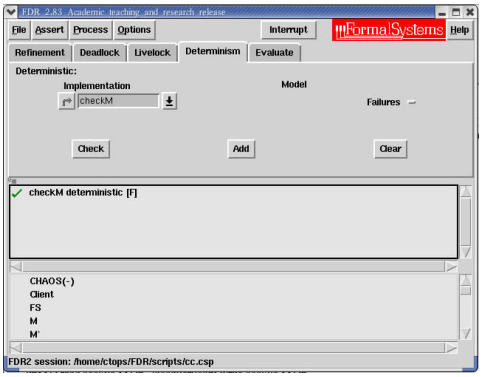

图 4 修改后的执行结果

显然,修改后的系统设计可证明解决了引用监视器 *RM* 被旁路的问题.

5 总结与展望

本文提出的扩展不干扰模型 ENISM 可用于分析操作系统中可能存在的违背信息流策略的执行轨迹和数据流动,从而分析系统中的存储隐蔽通道,解决信道控制问题,验证信息流策略.该模型视操作系统的不同功能模块为执行域,强调将未来可能执行的行为序列集合与数据存储值集合一同作为分析执行域安全状态改变的依据.并通过分析单步动作的执行对系统执行和信息流动的影响总结出安全的状态转换条件.本文还基于通信顺序进程 CSP 给出了形式化的系统设计描述方法和 ENISM 策略表达及验证方法,并通过一个实例展示了 ENISM 及相应策略验证方法的正确性和有效性.

虽然目前已存在多种形式化验证工具,但是对可验证的系统规模还存在一定的限制,而且系统描述也需要人工的参与.因此,在实际的策略验证过程中,如何对复杂的系统实现合理的分解,以提高验证的效率;如何改进 ENISM 模型,使验证条件更简单更便于验证,都是需进一步研究的重点.

参 考 文 献

[1] Goguen J, Mesegue J. Security policies and security models//Proceedings of the 1982 Symposium on Security and Privacy. Los Alamitos, 1982: 11-20

[2] Haigh J, Young W. Extending the non-interference model of MLS for SAT//Proceedings of the 1986 Symposium on Security and Privacy. Oakland, CA, 1986: 232-239

[3] Rushby J. Noninterference, transitivity, and channel-control security policies. Stanford Research Institute, Menlo Park: Technical Report CSL-92-02, 1992

[4] Xie Jun, Huang Hao. A noninterference model for nondeterministic systems. Journal of Software, 2006, 17(7): 1601-1608(in Chinese)
(谢钧, 黄皓. 一个非确定系统的不干扰模型. 软件学报, 2006, 17(7): 1601-1608)

[5] Ryan P Y A, Schneider S A. Process algebra and non-interference. Journal of Computer Security, 2001, 9(1/2): 75-103

[6] Roscoe A W, Woodcock J C P, Wulf L. Non-interference through determinism//Proceedings of the European Symposium on Research in Computer Security (ESORICS). LNCS 875. 1994: 33-54

[7] Graham C J. Some laws of non-interference//Proceedings of the Computer Security Foundations Workshop. Franconia, USA, 1992: 22-33

[8] Roscoe A W, Goldsmith M H. What Is Intransitive Noninterference? Mordano, Italy: Computer Security Foundations Workshop, 1999: 228-238

[9] Ma Jian-Ping, Yu Xiang-Xuan, Hong Fan, Zhang Jiang-Ling. A computer noninterference model. Chinese Journal of Computers, 1997, 20(11): 1034-1037(in Chinese)
(马建平, 余祥宣, 洪帆, 张江陵. 一个完整的无干扰模型. 计算机学报, 1997, 20(11): 1034-1037)

[10] Hoare C A R. Communicating Sequential Processes. New Jersey, USA: Prentice-Hall, 1985

[11] Roscoe A W. The Theory and Practice of Concurrency. New Jersey, USA: Prentice-Hall, 1997

CUI Jun, born in 1981, Ph. D. candidate. His current research interests include secure operating system, formal verification and security model.

HUANG Hao, born in 1957, Ph. D., professor, Ph. D. supervisor. His current research interests include information security and network security.

GAO Xiao-Chun, born in 1977, Ph. D. candidate. His current research interests include security model and formal verification.

Background

This work is mainly supported by the National High Technology Research and Development Program (863 Program) of China under grant No. 2007AA01Z409 and the National Natural Science Foundation of China under grant No. 60473093. They both aim to develop a secure and trusted microkernel operating system in which novel security technologies and policy models are being researched, system design will be formally specified and some of the security properties will be verified formally or informally. In past years, the group has done a lot of related works including a multi-policy views security model, a reference monitor, a trusted boot revolution, a microkernel trusted operating system prototype (NUTOS), and its partial formal specifications and verification. And this paper reports the authors' works on how to formally specify and verify information flow policies which reflect some of the security properties in NUTOS.

This paper aims to resolve how to use information flow models to formally specify and verify information security properties in operating systems. In this paper, the authors explain that traditional noninterference models don't distinguish data flow and control flow, so they could not be used to specify some of the security properties, such as a reference monitor which collects information from other modules for checking if there exists an attack, but it is not allowed for any collected information to interfere control flows in reference monitor. So, reference monitor requires different data flow and control flow policies, and this paper aims to resolve this problem by extending traditional noninterference theory. And for the purpose of distinguishing and analyzing data flow and control flow, in this paper the authors relate control flow with traces' sets in which each trace is a sequence of operations. The authors also introduce Communicating Sequential Processes CSP and its formal checking tool FDR2 for specifying and verifying information flow policies in operating systems, and based on CSP, propose and prove a theorem for determining whether an operating system design satisfies a given information flow policies.