隐通道传递信息机理的研究*)

刘志锋　鞠时光　王昌达　周从华
(江苏大学计算机科学与通信工程学院　江苏镇江 212013)

摘　要　可信计算机系统中一些隐蔽数据流避开了安全机制的监控，造成信息的泄漏。本文通过对这种隐蔽流泄漏信息的机理进行分析和抽象，提出了一个通道元模型。将每一类通道元看成一个有限状态机，以 Plotkin 的结构化操作语义等为基础，计算出状态机的状态变化序列。通过对不满足隐通道定义的状态变迁序列的归纳，得到了抽象机中安全状态转移的约束条件，找出两个通道元通过共享客体泄露信息的工作机理，从而开发出一种基于操作语义的隐通道标识方法。对电梯调度算法模型进行实验，可有效地标识出存在的隐通道。

关键词　隐通道，信息安全，安全模型

On the Principle of Information Transfer in Covert Channel

LIU Zhi-Feng　JU Shi-Guang　WANG Chang-Da　ZHOU Cong-Hua
(School of Computer Science and Telecommunication Engineering, Jiangsu University, Jiangsu Zhenjiang 212013)

Abstract　Some covert information flow evades the inspection of security mechanism in trusted computer system, which results in information leakage. The atomic channel model was established by analyzing and abstracting the principle of covert information flow. A finite state machine was used to describe an atomic channel model. Based on the structured operational semantics proposed by Plotkin we can compute sequences of states. By reasoning on state sequences against the definition of covert channel, restriction conditions of secure state translation were gained, and the principle of information leakage in two atomic channel models sharing an object was found. Consequently a covert channel identification method based on operational semantics was proposed. The experiment on elevator dispatch algorithm showed that our method could search for covert channels efficiently.

Keywords　Covert channel, Information security, Security model

由于隐通道在泄露机密信息方面所起的重要作用，1983年美国国防部在其发布的可信计算机系统评估准则(TCSEC)中，明确提出隐通道的问题，并规定在 B2 级及以上的高等级可信系统设计和开发过程中，必须进行隐通道分析。我国国标 GB/T18336、欧洲 ISO/IEC 15408 等均把隐通道分析的强度作为评估一个高等级可信系统的硬性指标。目前，各类可信系统中的隐通道问题不断地被发掘并加以分析，文[1]对该领域的现状已经做了一个全面的概述。隐通道概念[22]的内涵也逐步清晰，人们已在隐通道的搜索方面进行了较多的研究。

为了保证一个可信计算机系统的安全性，我们必须对其进行隐通道的搜索。一类是完全依赖顶级描述规范的隐通道标识方法。由 Denning 等提出的信息流分析法[2]，较多地用在寻找一个安全操作系统中的隐通道的场合。开发了诸如 MITRE 流分析器[3]、Gypsy 流分析器[4]等一些分析工具。

但这种完全依赖顶级描述规范来进行分析是不精确的，因为这类分析不能标识出所有可能出现在实现代码中的隐通道。在实践中，实现代码和形式化规范之间总是有差别的。这些代码包括性能监控、审计、调试以及其它和安全有关的代码，它们之中有可能导致潜在的隐通道。更关键的是，在实际情况中，很少有程序员愿意写详细而准确的系统顶级描述，有的系统甚至根本就没有这方面的工作，因此这类方法很难被软件开发团体所接受。

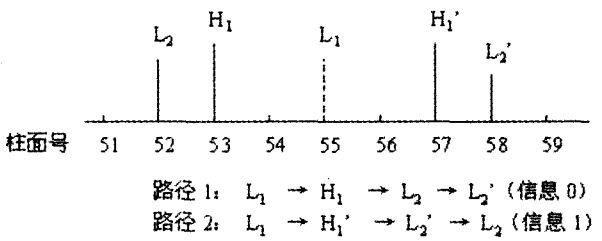
另一类是面向源代码的隐通道标识。如 1982 年 Goguen 等提出的无干扰分析法[5]、1983 年 Kemmerer 提出共享资源矩阵法[6]，以及 1990 年提出的隐蔽流树法[7]等。这些基于源代码的搜索方法，在代码量巨大、共享变量众多的场合，作业的空间、时间复杂度过高，即便在系统开发过程中遵循了良好的软件工程规范，依然很难实现。而且这种方法属系统实现后的分析，不便于重新设计系统或者部分修改设计以消除或限制隐通道。

上述这些研究结果，均是建立在对系统的顶级描述以及对源代码的语法或功能分析基础之上的。

我们考察文[13]给出的磁臂电梯调度算法模型。假设系统中有一个低安全级的进程 L 和一个高安全级的进程 H。进程 L 发出一个读磁盘柱面 55 的请求 L_1，然后立即放弃 CPU，并等待请求得到服务。接着高安全级进程 H 占有 CPU，它发出定位于柱面 53 的请求 H_1 或柱面 57 的请求 H'_1，在此之后立即放弃 CPU，并等待请求得到服务。当读柱面 55 的请求 L_1 得到服务后，低安全级进程立即同时发出两个异步定位请求，即定位于柱面 52 的请求 L_2 和柱面 58 的请求 L'_2。按照电梯算法，若高安全级进程请求的是定位于柱面 53 的请求 H_1，则请求服务的次序为图 1 所示的路径 1。请求 L_2 比请求 L'_2 先得到服务。反之，若高安全级进程请求的是定位于柱面 57 的请求 H'_1，则请求服务的次序为路径 2。请求 L'_2 比请求 L_2 先得到服务。根据请求 L_2 和请求 L'_2 得

*)本课题受到国家自然科学基金(编号：60573046)的资助。

到服务的次序，低安全级进程可以得知高安全级进程发出的定位请求柱面号，即高安全级进程 H 向低安全级进程 L 泄露了一个比特的信息“0”或“1”。



注：H_1 和 H_1' 为高进程 H 的两个不同请求；

L_1、L_2、L_2' 是低进程 L 的三个不同请求，55 为 L_1 请求的起始柱面号

图 1　磁臂隐通道(电梯调度算法)

在此例中，所有进程的操作都是合法的，系统中的安全控制机制并不能发现从高安全级进程向低安全级进程的信息传递。

发生这种问题的根本原因在于：低安全级进程可通过自己的运行状态和周围环境来感知高安全级进程的某个信息。而这个感知过程，用通过分析软件系统结构或源程序结构的方法是无从发现的。我们已证明隐通道的发生正是在不同的语境下，操作语义的歧义性所导致的[16]。

本文通过研究基于源代码的各类信息传导机制的形式化表示，以及它们的逻辑设计和代码设计的语义描述，讨论各类信息传导机制抽象机在不同语境下的操作语义，研究找出两个通道元抽象机通过共享客体泄露信息的工作机理。

1　可信系统的相关要素

正如在 TCSEC 等安全标准中所做的那样，我们将一个可信计算机系统分成若干个有序的安全等级 $L_1, L_2, \cdots, L_n$，其中 $L_i < L_{i+1}, 1 \leqslant i \leqslant n$。对每一级的可信系统用自然语言确定了一组安全约束条件。

定义 1(安全策略 λ)　我们把用自然语言对某一给定的可信系统所描述的安全约束条件的集合称为该系统的安全策略 λ。

每一安全策略对应着一个安全约束条件的集合。

$\lambda(L_i) = \langle \rho_1, \rho_2, \cdots, \rho_m \rangle$，

这里 L 表示系统的安全等级，ρ 表示安全约束条件。

公理 1　当两个不同等级的可信系统　满足 $L_i < L_{i+1}$ 时，它们对应的安全策略之间存在 $\lambda(L_i) \subset \lambda(L_{i+1})$ 的关系。

定义 2(安全模型 M)　在可信系统开发过程中，使用程序与算法对安全策略 λ 进行表达与实现。我们将这些程序与算法称为该可信系统的安全模型 M。

用一个安全模型来表达某个安全策略，由于受到实现人员的编程技术、所使用算法的优劣等因素的影响，通常 M 只能做到对 λ 的部分实现。我们用 Ω 函数代表模型所能覆盖的最大空间。即 Ω(λ)指一安全策略 λ 所确定的安全空间；Ω(M)指实现某一可信系统时，所使用的安全模型 M 所能覆盖的安全空间。一般情况下，$\Omega(M) \in \Omega(\lambda)$。显然，只有一个理想化的安全模型 M 才能对安全策略 λ 达到完全实现 $\Omega(\lambda) = \Omega(M)$。

定义 3(主体)　在一个信息传导过程中，引起信息传导的实体 S 称为主体，如一个用户或一个进程等等。主体 S 具有明确的安全等级 L(S)。

定义 4(客体)　在一个信息传导过程中，某个实体 O 用作为该信息传导的介质，我们将这个实体称为客体，如一个文件或一个记录等等。它具有明确的安全等级 L(O)。

在一个可信系统中，没有信息的流动，就不存在信息泄漏的问题。信息的流动往往存在着隐蔽流和形式流两种可能的缺陷[13]。形式流是一种无法通过系统安全机制检查的安全通道，即虽然信息流动符合安全策略 λ 的要求，但却不能通过安全模型 M 的检查；隐蔽流是安全系统中一些隐蔽数据流避开了安全模型 M 的监控，从而造成信息的泄漏。其主要原因是 Ω(M)未能实现对 Ω(λ)的彻底覆盖。

在软件系统的源代码中通常包含函数、数据结构和算法等用作为信息传导的介质：

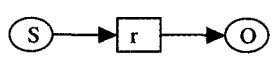
a. 函数。由于隐通道实现的是高安全级用户向低安全级用户的信息泄密，而用户进程/线程是通过调用系统函数来运行，即函数是所有用户进程/线程的入口，因此，若用户间想要实现信息传送，必然要借助某一或某些共享函数，所以函数是研究通道信息传导问题的最小逻辑单位。

b. 数据结构是信息逻辑上的载体，这有两层含义：一是数据结构本身是用来存放信息的，在 BLP 模型中，低安全级的用户进程无权访问高安全级信息的内容；二是数据结构的属性也可以作为信息载体，尽管低安全级用户进程无法读取数据结构中存放的数据内容，但是通过观察数据结构属性值的变化，来判断与它合作的高安全级用户所要传导的信息，这里变化的属性值就成为了信息载体，被用于隐通道信息传送。

c. 算法是程序的核心，系统功能实现的好坏程度取决于算法的优良与否。系统中信息传导机制同样依赖于算法。

2　信息传导模型

定义 5(通道元)　在某信息传导过程中，主体对客体的操作所形成的机制称为一个通道元。根据通道元中主体对客体的操作，我们可将通道元的操作分成如下 2 组：



（主体对客体的读操作）

（主体对客体的写操作）

我们将一个通道元，描述成有限状态机[21]

$T = (X, A, S, s_0, \delta)$，

这里 X 是抽象机操作的集合；A 是抽象机属性的集合；S 是抽象机状态的集合；s_0 是初始状态；δ 是状态变换，即 $\delta: S \times X \rightarrow S$。

定义 6(信息传导机制) 在某个信息通道中，主体 S_1 利用操作方法 β 在 t 时刻将信息 I 通过客体 O 传输到主体 S_2。我们将它们所构成的机制称为信息传导机制。正如图 2 所示。这里不妨假设主体 S_1 的安全级别高于 S_2。

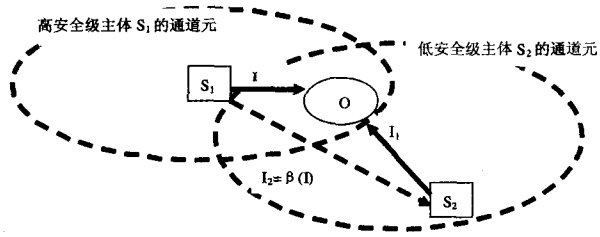


图 2 信息传导机制模型

我们可在时间属性上将信息传导机制传递信息分为 3 个有序的动作：

动作 1：在 t_1 时刻，主体 S_1 将信息 I 利用传导方法 $\beta_1(O)$ 送至客体 O；

动作 2：在 t_2 时刻，主体 S_2 从客体 O 利用传导方法 $\beta_2(O)$ 获得信息 I_1. 这里 $I_1 \square I$；

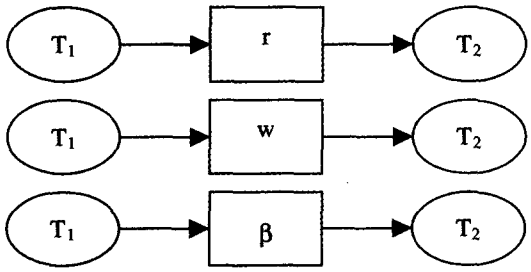
动作 3：在 t_3 时刻，主体 S_2 从主体 S_1 利用传导方法 β 获得信息 I_2。这里 β 是 β_1 和 β_2 的叉积，即 $\beta=\beta_1(O)\times\beta_2(O)$。我们把传导方法 β 称为感知。

感知是一个逻辑动作，并没有发生实在地物理动作。感知主要是计算和推理。所以，不存在通过系统安全机制检查的过程。这就意味着，即使 β_1、β_2 满足某一安全模型，也不能保证 β 满足系统的安全策略 λ。

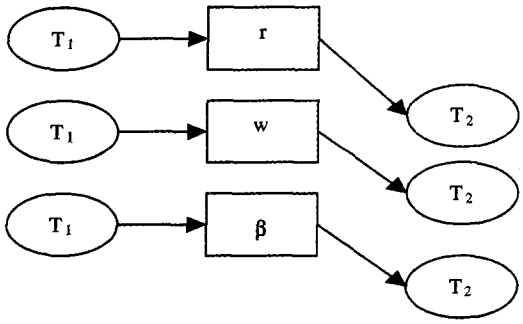
3 个动作不一定是连续发生的，但动作发生的时刻一定满足 $t_1<t_2\leqslant t_3$ 的关系。

根据 S_1 和 S_2 之间安全等级的高低，我们可将信息传导机制分成如下 9 组：

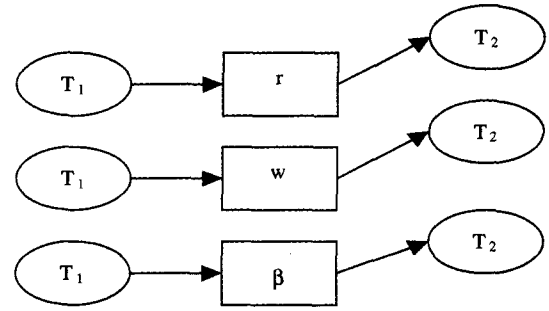
P_{1-3}：$L(S_1)=L(S_2)$，



P_{4-6}：$L(S_1)>L(S_2)$，



P_{7-9}：$L(S_1)<L(S_2)$，



根据 BLP 安全模型的向上写向下读规则，我们有 P_1，P_2，P_3，P_4，$P_8\square\Omega\{M=BLP\}$，即这 6 组信息传导机制满足 BLP 安全模型的规则，而 $P_7\square\Omega(\lambda)-\Omega\{M=BLP\}$，$P_5\square\Omega(\lambda)-\Omega\{M=BLP\}$，即这两类通道元违背 BLP 安全模型的规则，肯定不能被执行。即 P_7，P_5 不能被执行。而 P_6，P_9 是既违背安全模型，也违背安全策略。

3 基于操作语义的隐通道标识方法

本节主要讨论通道元抽象机在不同语境下的操作语义[20]，研究找出两个通道元抽象机通过共享客体 O 泄露信息的工作机理。

基于操作语义的隐通道标识方法由以下几个部分构成。

步骤一，以 Plotkin 的结构化操作语义等为基础，计算出每个通道元状态机的状态变化序列。

步骤二，通过研究抽象机状态变迁序列，找出通过共享客体属性，高安全级状态的操作能被低安全级的状态能够察觉的机理。这一策略可以归纳为：

- 找出已知语境中能够改变共享客体 O 的属性值的状态；
- 低安全级进程获取共享客体属性值的变化；
- 低安全级进程根据自己前后状态变化及共享客体属性值的变化，通过计算、推理感知出高安全级进程的信息。

步骤三，对于不满足隐通道定义的的状态变迁序列，通过归纳这个序列中状态转移需要满足的约束条件，得到安全状态转移的约束条件。

方法的第一和第二部分将在文章的第 4 小节中，通过磁臂电梯调度算法来详细说明。在方法的第三部分，我们通过研究划分安全状态变迁序列的功能等价类，并将每一个等价类中状态变迁序列所具有的共性，使用抽象机状态变迁约束条件的形式加以描述。然后将已得出的安全状态转移约束条件，按其功能分别归入不同的等价类，并从中选取所有的非空等价类，得到一组安全状态转移的基本约束条件。

我们已研究得出一个安全的信息传导机制，其安全状态转移由下述 6 个基本约束条件组成：

①在任何状态下，以高安全级的方式输出总是安全的。因为以高安全级的方式输出，意味着信息不会从高安全级流向低安全级，因而总是安全的。

② 在没有任何高安全级影响的状态下，系统以低安全级的方式输出是安全的。

③ 在任何状态下，对高安全级变量赋值是安全的。

④在低安全级的状态下，如果输入低安全级变量，输出也是低安全级变量，那么是安全的。

⑤在分支或循环结构中，各选择项与判断进入分支或循

环结构的变量需具有相同的安全级才是安全的。否则就能从分支或循环结构的选择上观察输入变量的变化，那样将导致信息的泄漏。

⑥在特定的情况下，尽管系统中存在从高安全级向低安全级的信息流动，但如果这种信息的泄漏对于外部观察者是不可见的，系统仍然是安全的。

4 实例

我们将本文前一部分所引入的磁臂调度进程抽象机描述为 $SM=(X,A,S,s_0,\delta)$，这里，

$X=\{$
send_request(m,priority)　m 表示请求位置，priority 安全级，
quit_cpu　放弃 cpu
wait_same_priority(n,priority)　等待同级进程执行
get_request(n,priority)　执行请求
wait_high_priority　等待高进程执行
$\}$

$\forall a\in A, a=\{$
num　进程数
request_list[]　记录请求列表每个元素是个二元组，如〈position, priority〉
exe_index[]　记录请求被先后执行的次序
is_request_exe[]　记录相应请求是否得到满足，t(true)，f(false)
e_direction　磁头的方向(共享变量)值为 up，down，⊥(方向未知)
e_position　磁头当前所处的位置
$\}$

$\forall s\in S, s=$(num, request_list[], exe_index[], s_request_exe[], e_direction, e_position)是对属性的一组赋值；

$s_0=(0,null,null,null,\perp,0)$；

δ:(s_i = (num, request_list[], exe_index[], s_request_exe[], e_direction, e_position))

(1)发送请求

$$\frac{s_i, send\_request(m,priority)}{s_i \xrightarrow{send\_request(m,priority)} s_{i+1}}$$

$$\left(\begin{array}{l} s_{i+1}=(num+1, request\_list[]\cup\{m,priority\}), \\ exe\_index[], is\_request\_exe[]\cup\{f\}, e\_direction, e\_position \end{array}\right)$$

(2)接受请求

$$\frac{s_i, get\_request(n,priority)}{s_i \xrightarrow{get\_reuquest(n,prioriry)} s_{i+1}}$$

$$\left(\begin{array}{l} s_{i+1}=(num-1, request\_list[]\backslash\{(n,priority)\}, \\ exe\_index[]\cup\{n\}, \{is\_request\_exe[]\backslash\{f\}\}\cup\{t\}, \\ e\_direction', n) \end{array}\right)$$

$$e\_direction'=\begin{cases} up: e\_position<n \\ down: e\_position>n \\ e\_deriction: e\_position=n \end{cases}$$

(3)放弃 CPU

$$\frac{s_i, quit\_cpu}{s_i \xrightarrow{quit\_cpu} s_{i+1}}$$

$$\left(\begin{array}{l} s_{i+1}=(num, request\_list[], \\ exe\_index[], is\_request\_exe[], \\ \perp, e\_postiion \end{array}\right)$$

(4)等待高进程的执行

$$\frac{s_i, wait\_high\_priority}{s_i \xrightarrow{wati\_high\_priority} s_{i+1}}$$

$$\left(\begin{array}{l} s_{i+1}=(num, request\_list[]\}, exe\_index[]\cup\{h\}, \\ is\_request\_exe[], \perp, \perp) \end{array}\right)$$

(5)等待同级进程的执行

$$\frac{s_i, wait\_same\_riority(n,priority)}{s_i \xrightarrow{wait\_same\_priority(n,priority)} s_{i+1}}$$

$$\left(\begin{array}{l} s_{i+1}=(num, request\_list[]\}, exe\_index[], \\ is\_request\_exe[], e\_direction, n) \end{array}\right)$$

根据上面所提的磁臂隐通道语义模型，我们来分析图 1 中高低进程的请求如何来导致隐通道的产生。

首先我们分析路径 1 高低进程的请求顺序：$L_1(55)\rightarrow H_1(53)\rightarrow L_2(52)\rightarrow L_2'(58)$，其中 L_1, L_2, L_3' 是低安全级进程请求，H_1 是高安全级进程请求。

根据电梯调度算法的规则，我们可以得到在磁臂调度过程中执行路径 1 时高进程的所有状态 $s_0^H, s_1^H, s_2^H, s_3^H$ 和低进程的所有状态 $s_0^L, s_1^L, s_2^L, s_3^L, s_4^L, s_5^L, s_6^L, s_7^L, s_8^L$（注：$s_0^H, s_0^L$ 分别是高低进程的初始状态，设 $s_0^H(0, null, null, null, \perp, 0)$，$s_0^L(0, null, null, null, \perp, 0)$）。

高进程的各个状态的推导过程如下：

$$\frac{s_0^H, send\_request(53,H)}{s_0^H \xrightarrow{send\_request(53,H)} s_1^H}$$

$(s_1^H(1,\{(53,H)\}, null, \{f\}, \perp, 0))$

$$\frac{s_1^H, quit\_cpu}{s_1^H \xrightarrow{quit\_cpu} s_2^H}$$

$(s_2^H(1,\{(53,H)\}, null, \{f\}, \perp, 0))$

$$\frac{s_2^H, get\_request(53,H)}{s_2^H \xrightarrow{get\_request(53,H)} s_3^H}$$

$(s_3^H(1, null, \{53\}, \{t\}, down, 53))$

低进程各个状态的推导过程如下：

$$\frac{s_0^L, send\_request(55,L)}{s_0^L \xrightarrow{send\_request(55,L)} s_1^L}$$

$(s_1^L(1,\{(55,L)\} null, \{f\}, \perp, 0))$

$$\frac{s_1^L, quit\_cpu}{s_1^L \xrightarrow{quit\_cpu} s_2^L}$$

$(s_2^L((1,\{55,L)\} null, \{f\}, \perp, 0))$

$$\frac{s_2^L, get\_request(55,L)}{s_2^L \xrightarrow{get\_request(55,L)} s_3^L}$$

$(s_3^L(0, null, \{55,\}, \{t\}, up, 55))$

$$\frac{s_3^L, send\_requetst(52,L),(58,L)}{s_3^L \xrightarrow{send\_request(52,L),(58,L)} s_4^L}$$

$(s_4^L(2,\{52,L)(58,l)\}, \{55\}, \{t,f,f\}, up, 55))$

$$\frac{s_4^L, wait\_high\_priority}{s_4^L \xrightarrow{wait\_high\_priority} s_5^L}$$

$(s_5^L(2,\{(52,L),(58,L)\}, \{55,h\}, \{t,f,f\}, \perp, \perp))$

$$\frac{s_5^L, get\_request(52,L)}{s_5^L \xrightarrow{get\_request(55,L)} s_6^L}$$

$(s_6^L(1,\{(58,L)\}, \{55,h,52\}, \{t,f,f\} down, 52))$

$$\frac{s_6^L, wait\_same\_priority(52,L)}{s_6^L \xrightarrow{get\_request(55,L)} s_7^L}$$

$(s_7^L(1,\{(58,L)\}, \{55,h,52\}, \{t,f,f\}, down, 52))$

$$\frac{s_7^L, get\_request(58,L)}{s_7^L \xrightarrow{get\_request(58,L)} s_8^L}$$

$(s_8^L(1,\{(58,L)\}\{55,h,52,58\}, \{t,t,t\}, down, 58))$

根据上面的高低进程抽象机的状态及其变迁关系，我们得到高进程抽象机状态变迁序列，$s_0^H s_1^H s_2^H s_3^H$ 和低进程抽象机状态变迁序列 $s_0^L s_1^L s_2^L s_3^L s_4^L s_5^L s_6^L s_7^L s_8^L$，用表格形式表示如下：

(下转第 142 页)

引擎。Raindrop因此使用了代数与自动机相结合的查询技术，并运用了数据库中的概念将系统分为四层：语义层、逻辑层、物理层和执行层，使得系统的优化机会更多，但是Raindrop中的代数特征使得系统相对其它流查询引擎占用更多的空间，不利于数据量很大的查询。TurboXPath[7]使用解析树取代传统的自动机作为查询的核心数据结构，有较高的查询效率。同时它还可以支持复杂的XPath式，除了文[8]的XPath特性外，还能支持反向轴，但是对XQuery的支持力度关注不够，因而不能支持复杂XQuery，如order子句。

结论 本文提出的TreeBuf算法是对文[8]中PBuf算法的继承，拥有PBuf的各种优点，也是对PBuf算法的改进。TreeBuf算法可以同时应用在XQuery查询和XPath查询上，并且通过前缀共享计算提高查询效率。

目前所实现的XQuery查询系统也存在一些不足，如对XQuery的支持力度仍然不够完善，不支持动态元素构造器、聚集函数等等。尽管在第5节说明了TreeBuf算法对XPath查询的改进，但是对于如何提取绝对XPath式的公共前缀，还没有成熟的算法。对系统支持力度的进一步扩展和寻找高效的前缀查找算法将作为下一步工作。

参考文献

1 Diao Yanlei. Altinel M, Franklin M J, et al. Path sharing and predicate evaluation for high-performance XML filtering. ACM Transations on Database Systems, 2003, 28(4): 467～516. http://yfilter.cs.berkeley.edu/code_release.htm
2 Chan Chee Yong, Felber P, Garofalakis M N, et al. Efficient Filtering of XML Documents with XPath Expressions. VLDB Journal, Special Issue on XML, 2002, 11(4): 354～379
3 Green T J, Miklau G, Sucia D. Processing XML streams with deterministic automata and stream indexes. ACM TODS, 2004, 29(4)
4 Gupta A, Suciu D. Stream Processing of XPath Queries with Predicates. In: SIGMOD, 2003. 419～430
5 Koch C, Scherzinger S, Schweikardt N, et al. FluXQuery: An Optimizing XQuery Processor for Streaming XML Data. In: Proc. VLDB, 2004. 228～239
6 Su Hong, Rundensteiner A, Mani M. Semantic Query Optimization in an Automata-Algebra Combined XQuery Engine over XML Streams. VLDB, 2004. 1293～1296
7 Josifovski V, Fontoura M, Barta A. Querying XML streams. VLDB, 2005, 14(2): 197～210
8 张昱，吴年．一种逐层提升缓冲的XML流查询自动机．小型微型计算机系统(已录用)
9 http://www.w3.org/TR/xquery/
10 Schmidt A, Waas F, Kersten M, et al. XMark: A Benchmark for XML Data Management. In: Proc. VLDB, 2002. 974～985
11 Zhang Xin, Dimitrova K, Wang Ling, et al. Rainbow: Multi-XQuery Optimization Using Materialized XML Views. In: Proc. ACM, SIGMOD, 2003. 671

(上接第95页)

时刻(t)	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
高进程状态	s_0^H	s_0^H	s_1^H	s_2^H	s_2^H	s_3^H			
高进程共享属性(direction,position)	⊥,0	⊥,0	⊥,0	⊥,0	⊥,0	down,53			
低进程状态	s_0^L	s_1^L	s_2^L	s_3^L	s_4^L	s_5^L	s_6^L	s_7^L	s_8^L
低进程共享属性(direction,position)	⊥,0	⊥,0	⊥,0	up,55	up,55	⊥,⊥	down,52	down,52	down,58

由此，根据上面表格我们可以清楚地看到：高进程状态 s_3^H 能够改变该共享客体磁臂 O 的属性；低进程状态 s_4^L，s_6^L 能够察觉这一改变，并且低进程状态 s_4^L，s_6^L 根据自身前后状态发生的变化能够察觉共享客体属性的变化。即低进程Low可以感知到磁臂在执行高进程High请求时是磁头的移动方向是down，根据电梯调度算法和高低进程之间的通信是按照事先约定好的编码方法进行通讯的，所以此时低进程可以感知到刚才高进程High请求的是53号柱面，从而泄露一个比特的信息。

隐通道传递信息的方法是感知型的，这是一种逻辑操作，无需通过安全模型的检验。隐通道的这一特点，决定了传统的隐通道的分析方法，即在语法分析基础上展开的基于系统顶级描述或源代码的分析，无法发现这种隐通道问题。本文提出的基于操作语义的隐通道分析方法，可有效地解决这类问题。

参考文献

1 McHugh J. Covert Channel Analysis: A Chapter of the Handbook for the Computer Security Certification of Trusted Systems [R]. Portland State University, December 1995
2 Denning D E, Denning P J. Certification of Programs for Secure Information Flow. Communications of the ACM[J], 1977, 20(7): 504～513
3 Bell D E, LaPadula L J. Secure computer system: Unified exposition and MULTICS interpretation: [TechRep MTR-2997]. The MITRE Corporation, 1976
4 McHugh J. An information flow tool for Gypsy. In: Computer Security Applications Conference[C], ACSAC 2001. Proceedings 17th Annual Dec. 2001. 191～201
5 Goguen J A, Meseguer J. Security Policies and Security Models. In: Proceedings of the IEEE Symposium on Security and Privacy [C], Oackland, California, 1982. 11～20
6 Kemmerer RA. Shared resource matrix methodology: An approach to identifying storage and timing channels. ACM Trans on Computer Systems[J], 1983. 256～277
7 Kemmerer R A. Covert Flow Trees: A Visual Approach to Analyzing Covert torage Channels. IEEE Transactions on Software Engineering[J], 1991, 17(11): 1166～1185
8 Venkatraman B R, Newman-Wolfe R E. Capacity Estimation and Auditability of Network Covert Channels. In: Security and Privacy, IEEE Symposium[C], May 1995. 186～198
9 Wang C D, Ju Shiguang. Research on the methods of search and elimination in covert channel. In: Grid and Cooperative Computing [C], LNCS, PT 1 3032: 1 2004. 988～99
10 Goldschlag D M. Several Secure Store and Forward Devices. In: Proc. of the Third ACM Conference on Computer and Communications Security[C], New Delhi, India, March 1996. 129～137
11 Kang M H, Moskowitz I S. A pump for rapid, reliable, secure communication. In: 1st ACM Conference on Computer and Communications Security [C], Fairfax, Virginia, November 1993. 119～129
12 Dogu T M, Ephremides A. Covert information transmission through the use of standard collision resolution algorithms. Lect Notes Comput Sc, 2000, 1768: 419～433
13 Karger P A, Wray J C. Storage Channels in Disk Arm Optimization. In: IEEE Symposium on Security and Privacy, 1991. 52～61
14 Ju Shiguang, Song Xiaoyu. On the Formal Characterization of Covert Channel. Lecture Notes in Computer Science, 2004, 3309: 155～160
15 Trusted Computer System Evaluation Criteria[R]. Department of Defense, U.S.A. December 1985
16 Hu Wei-Ming. Reducing timing channels with fuzzy time. In: IEEE Symposium on Research in Security and Privacy, Oakland, California, May 1991. 8～20
17 Melliar-Smith P M, Moser L E. Protection against covert storage and timing channels. In: Proceedings 4th IEEE Computer Security Foundations Workshop, June 1991. 209～214
18 Eckmann S T. Eliminating Formal Flows in Automated Information Flow Analysis. In: Proceedings of the IEEE Symposium on Security and Privacy, 1994. 30～38
19 Son S H, Mukkamala R, David R. Integrating security and real-time requirements using covert channel capacity. IEEE T KNOWL DATA EN, 2001, 13(5): 862～862
20 陆汝钤．计算机语言的形式语义．科学出版社，1992
21 冯玉琳，李京，黄涛．对象语义理论和行为约束推理．计算机学报，1993, 16(11)
22 卿斯汉．高安全等级安全操作系统的隐蔽通道分析．软件学报，2004, 15(12)