信息流和隐蔽通道

信息安全原理第六组

信息安全原理
作者：程礼棋、张金羽、陈宇扬、李想、杨凌霄、程梦烨、王晋琰、王嘉怡、王子腾、朱一丁

摘　要：本文将重点放在信息安全中的信息流和隐蔽通道这个方面，主要分为两个部分分别对信息流和隐蔽通道进行整体的简介和总结。对信息流主要采用识别检测分析和基于信息流的建模分析等，从保护信息流的层面深入展开探讨。对于信息流的保护离不开流中的特殊信道-隐蔽通道，第二部分主要是发现和防止隐蔽通道的方法分析和总结，主要从 SRM(共享资源矩阵机制）出发研究隐蔽通道的防止方法，并实现了基于 Linux 进程标识符下的信息流传输，深入了解了隐蔽通道的存在机制和利用安全漏洞的危害。

关键词: 信息流分析；隐蔽通道；信息流安全；SRM；信息访问建模；进程控制符；Linux;
中图法分类号: TP　　**文献标识码:** A

Information Flow and Convert Channel

Group 6 , Principle of Information Security, Cheng Liqi

(1. *Principle of Information Security, Zhejiang University, Zip Code 310058, China*)
Cheng Liqi, Zhang Jinyu, Chen Yuyang, Li Xiang, Yang Lingxiao, Cheng Mengye, Wang Jinyan, Wang Jiayi, Wang Ziteng, Zhu Yiding

Abstract: This article focuses on the aspect of information flow and convert channels in information security. It is mainly divided into two parts to give an overall introduction and summary of the information flow and convert channels. The information flow mainly adopts identification detection analysis and modeling analysis based on information flow, etc., and carries out in-depth discussion from the aspect of protecting information flow. The protection of the information flow cannot be separated from the special channel in the flow-the convert channel. The second part mainly analyzes and summarizes the methods of discovering and preventing the convert channel, mainly from the SRM (shared resource matrix mechanism) to study the convert channel prevention method. And realized the information flow transmission based on the Linux process identifier, in-depth understanding of the existence mechanism of convert channels and the hazards of using security vulnerabilities.

Key words: Information flow analysis; Covert channels; Information flow security; SRM; Information access modeling; pid; Linux;

1 信息流

1.1 信息流简介

何谓信息流？信息流（Information Flow）理论，也称为通道理论（Channel Theory），是一种通用的规则理论。能应用到自然界的生物、物理系统和人工世界的计算系统内在的分布式信息的交流中。基本概念之一是信息射（Infomorphisms），是形成信息通道的重要因素。由于信息流理论能弥补 Shannon 信息论难以支持语义互操作的不足，因此逐渐受到重视。

信息从实体 A 转移至实体 B 的过程被称为信息流，用 A →B 表示。系统中信息流有两种:合法信息流和非法信息流。合法信息流是符合系统安全策略的所有信息流,而不合法的信息流都属于非法信息流。（隐通道）

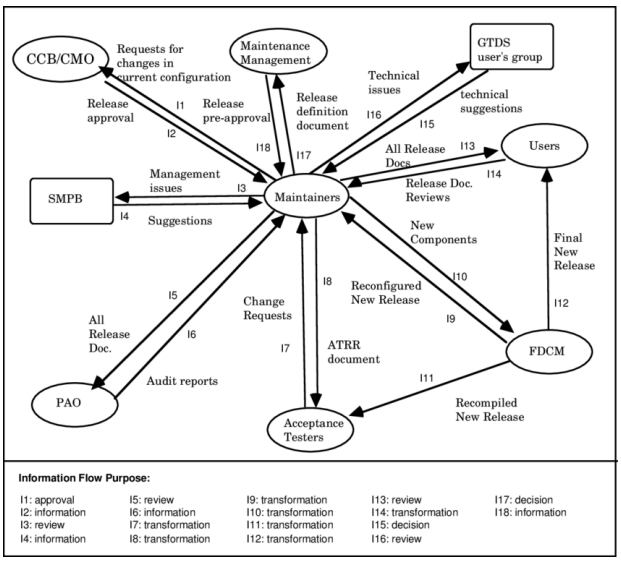
主体是引起信息流发生的活动实体,即系统中的用户和进程 s 表示。

客体是主体操作的对象,用 O 表示,如:操作系统中的文件、缓存等信息交互的媒介。

安全标签用于表示主体和客体的安全等级。安全标签指的是绝密(TS)、机密(S)、秘密(C)和普通(U)。

操作是主体对其它主体或客体所实施的动作。操作是信息流产生的根本原因,操作类型有:读(read)、写(write)、生成(create)和删除(delete),用 op 表示操作的集合,有 op={read,write,create,delete}。

图 1　信息流实例



1.1.1 信息流的信道

信息流通过流来传播，不可避免的使用流的信道，流的信道的保护和安全策略是整个信息流安全原理的防护重点。其中隐蔽通道是之后信息流防护的重点和难点。

1.正规信道（Legitimate Channels）通过设计者指定的信息传送渠道(如进程参数)在程序或进程间传送信息的。

可以理解为编程语言中的临时变量。

2. 存储信道（Storage Channels）指由多个进程或进程共享的客体，例如通过共享文件和或程序中的全局变量传递信息。

3. 隐蔽通道（Convert Channels）指不被设计者或用户所知道的泄露系统内部信息的信道。例如：通过观察某个进程的操作规律，判断这个进程在干什么，这种信道也不完全面向信息交换；例如：进程在系统夹在过程中产生的某些效果。

一般来说，正规信道的安全防护最简单。存储信道的安全防护要困难得多，因为每一个客体（可能是文件、变元和状态比特）都需要保护。

1.2 信息流原理

信息流在计算机中表现为整体或部分数据的转移。在系统中，信息可并发地通过网络或硬件通道从源头端输送至接收端。系统的安全策略限定了对信息流的传送，其主要的设计目的在于保障数据的保密性或者数据的完整性。对于前者，系统注重于防止信息流向未授权的用户，而对于后者，系统中信息可被允许流向保密性更低的进程。

系统安全策略的制定便依据与信息流原理：

第一原理：信息流来自分布式系统中的规则.

第二原理：信息流的重要之处是同时包括了类型（type）和个性（particular）.

第三原理：分布式系统中某些组件的信息依靠连接之间的规则携带了其他组件的信息.

第四原理：一个给定的分布式系统的规则与利用信息通道对其进行分析有关.

1.3 信息流安全防护

信息流的安全原则规定了信息可以流动的路径，当信息流发生时，传输过程中的信息流可以用端与通道的组合形成一组有向图，系统通过规定图的结构可以限制信息的流通途径和方式。在系统中，信息流的产生不是由于数据被改动，而是由于操作改动数据的控制信号的产生和传递，这也就意味着单纯关注显式存在的数据变化不足以监测系统内部全部的信息流，我们还需要检测隐式传送的数据。

为提高传输过程的安全性，我们可以在源头端采取加密的访问控制机制，防止错误释放信息，但仅加密是不够的，如果有程序合法访问信息，则这些信息有可能会被程序二次传播，造成泄漏。因此，静态分析程序的安全级别十分重要，如 L 表示低安全性，H 表示高安全，那么 H->L 则意味着高泄漏可能，应当对程序的访问提高警惕。同时，为防止恶意程序污染信息，可以用同样的方式设置信息的安全级别，避免污染信息扩散至安全信息区，以提高信息流安全程度。信息流在计算机中表现为整体或部分数据的转移。在系统中，信息可并发地通过网络或硬件通道从源头端输送至接收端。系统的安全。

1.3.1 信息流安全原则

信息流的安全原则规定了信息可以流动的路径，当信息流发生时，传输过程中的信息流可以用端与通道的组合形成一组有向图，系统通过规定图的结构可以限制信息的流通途径和方式。

信息流安全原则主要针对数据的两个方面：

数据的保密性：防止信息流向未授权的用户.

数据的完整性：允许信息流向保密性更低的进程.

1.3.2 信息流安全策略

信息流的传输路径形成一组有向图，安全系统通过规定图的结构可以限制信息的流通途径和策略。

信息流安全策略主要从权限限制和安全级别考虑：

控制访问机制：在信息源头端采取加密的访问控制机制，防止错误释放信息.

静态分析安全级别:

程序安全级别：监测低安全程序对高安全程序的访问
信息安全级别：避免污染信息扩散至安全信息区

1.4 信息流基本模型

1.4.1 信息流的格模型

信息流的格模型是 Denning 建立的， 用来描述信息流的信道与策略。

相关概念：
如果（L,≤）是一个偏序集合，L 中每一对元素 x 和 y 都有最大下界与最小上界，则二元组（L,≤）是格。
有限格:L 中元素的个数是有限的格

在之后的信息流识别中做具体的介绍和分析。

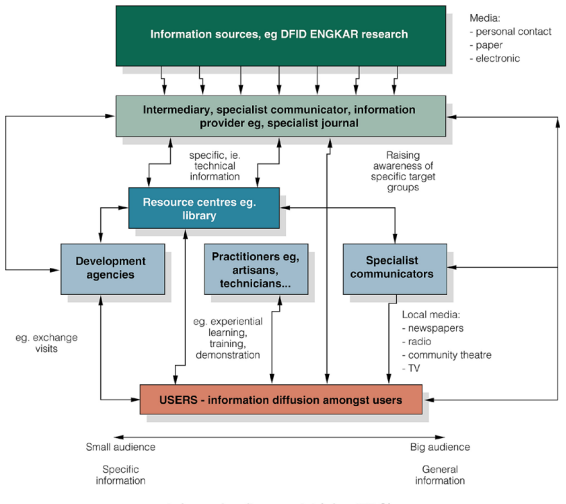
1.4.2 信息流的状态机模型

通过描述和限制能影响系统安全的各种变量与规则，达到保持系统和维持安全状态的目的。

一个状态机是一个七元组<U, S, C, O, out, do, s0>，基于状态机模型的系统， 如果经过一系列的转换规则后，最终输出能和输入一样是安全的， 就表明系统一直处于安全的状态。

信息流的格模型是通过格结构的流动策略、状态和状态转换来形式化建模的， 因此从某种意义上来说， 信息流的格模型也是一种状态机模型。

图 2 信息流模型



Information flow model (after ITDG)

1.5 基于信息流的信息访问建模

1.5.1 信息访问建模定义及目的

【定义】：信息访问建模是一种用来定义信息常规表示方式的方法。使用不同的应用程序对所管理的数据进行重用，变更以及分享。

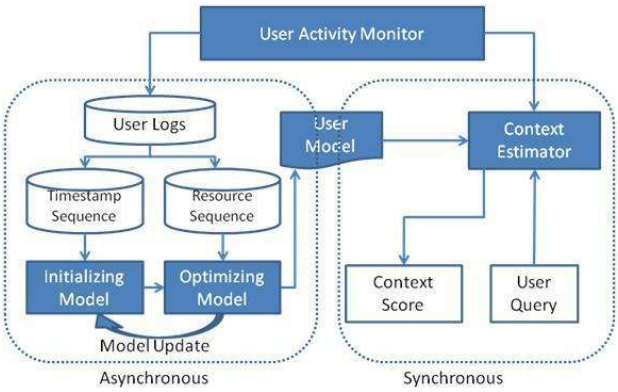
【目的】：信息访问建模在于对对象间相关性的描述。 提供了必要的通用语言来表示对象的特性以及一些功能，以便进行更有效的交流。

通过使用信息模型，我们可以使用不同的应用程序对

所管理的数据进行重用，变更以及分享。使用信息模型的意义不仅仅存在于对象的建模，同时也在于对对象间相关性的描述。 除此之外，建模的对象描述了系统中不同的实体以及他们的行为以及他们之间（系统间）数据流动的方式。 这些将帮助我们更好的理解系统。对于开发者以及厂商来说，信息模型提供了必要的通用语言来表示对象的特性以及一些功能，以便进行更有效的交流。

下图就是基于一个上下文搜索系统信息的模型。

图 3　上下文搜索系统的信息访问模型



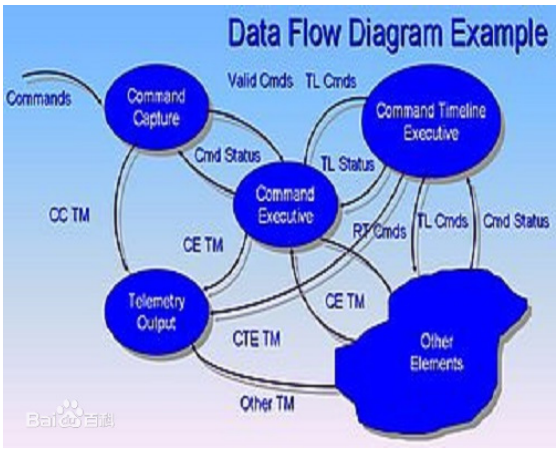
1.5.2　信息访问建模方式

信息访问建模目前各种常见的核心方法如下：

（1）结构化建模方法。

结构化建模方法是以过程为中心的技术，可用于分析一个现有的系统以及定义新系统的业务需求。结构化建模方法所绘制的模型称为数据流图（DFD）。

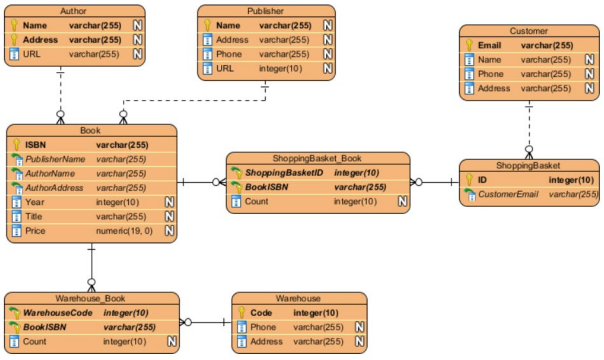
图 4　数据流图（DFD）



（2）信息工程建模方法（或数据库建模方法）。

信息工程建模方法是一种以数据为中心，但过程敏感的技术，它强调在分析和研究过程需求之前，首先研究和分析数据需求。这种建模方法所创建的模型被称为实体联系图（ERD）。

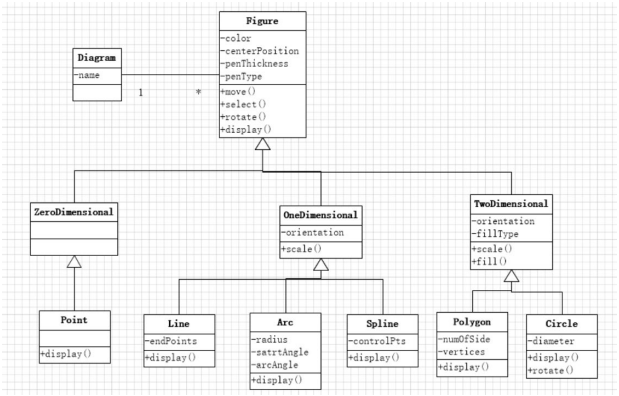
图 5　实体联系图（ERD）



（3）面向对象建模方法。

面向对象建模方法将“数据”和“过程”集成到被称为“对象”的结构中，消除了数据和过程的人为分离现象。面向对象建模方法所创建的模型被称为对象模型。随着面向对象技术的不断发展和应用，形成了面向对象的建模标准，即 UML（统一建模语言）。UML 定义了这些模型图以对象的形式共建一个信息系统或应用系统。

图 6　面向对象建模（UML）



1.5.3　信息访问建模功能

【企业建设现实功能：】

物流企业在规模化发展和网络化管理过程中，必然要面临各种各样的业务管理瓶颈和业务发展难题。信息化建模的有效实施将有助于突破企业发展和管理的瓶颈，促进企业发展战略的贯彻落实。

尽管一些企业已经进行了程度不同的信息化建设，但由于系统建设往往受全套解决方案的指导思想的制约，最终的信息系统较为封闭，形成了一个个“信息孤岛”，这样的信息系统对于将信息共享视为关键的物流业来说，互联互通困难，造成信息流不畅。

【存储、维护、处理功能】

信息模型是从信息的角度对企业进行描述。企业信息系统用于存储/维护/处理与企业相关的所有信息，而信息是集成的基础，是联系各个功能元素的纽带，因此建立企业信息模型是非常重要的，它为信息共享提供了帮助。通过对系统决策过程的建模，可以了解系统的决策制定原则和机理，了解系统的组织机构和人员配置。组织模型描述组织结构树、团队、能力、角色和权限等。资源模型描述企业的各种资源实体、资源类型、资源池、资源分类树、资源活动矩阵等。产品模型描述产品类型和产品结构等信息，也包括产品和其它企业要素之间的关系。

【最终目标】

信息建模的目标是用面向对象的方法刻划企业数据和信息的需求，并同时确定企业的关键信息，明确其主要内容，以形成企业信息系统集成的依据。

2 如何识别信息流

2.1 信息流控制问题

信息流控制的问题是信息流安全的关键问题。要实现对数据与隐私的安全保护,加密、访问控制和信息流控制是最有效的方法:对整个信息流的控制是从安全角度进行考虑,从信息流概念出发对信息流控制问题进行阐述和解释。

2.1.1 信息流及信息流控制概念

信息流是指信息在系统内部和系统之间的传播和流动,是 Denning 首先提出的,如果 A 信息影响了 B 信息的值,那么就存在从 A 到 B 的信息流。

信息流控制(Isarithmic Flow Control)是指以相应的信息流策略控制信息的流向。信息流控制将信息时代的用户串联起来。经济学上甚至提出过“控制信息流以控制支付流,控制支付流以控制资金流”,信息时代下信息流在当今经济中起到很大的作用,对其控制和保护的需求不断提高。为了解决分区内核中分区间通信存在的信息流控制问题,提出了一种兼顾通信效率和安全性的模型:信息流控制方案(Flow Control Scheme)。

2.1.2 信息流控制策略

信息流控制策略是规定客体能够存储的信息的安全类和客体安全类之间的关系,其中包括不同安全类客体之间信息的流动关系。如果系统的访问控制机制是完善的,但缺乏适当的信息流策略或因缺乏实现信息流策略的适当机制也会造成信息的泄露。

信息流控制策略一般包括数据机密性策略和完整性策略。

机密性策略是防止信息流向未授权获取该信息的主体

完整性策略是防止信息流向完整性高的主体或数据

2.1.3 信息流控制机制实现的核心思想

核心思想:将标签(污点)附着在数据上,标签随着数据在整个系统中传播(数据派生出的对象也将会继承原有数据标签),并使用这些标签来限制程序间的数据流向。

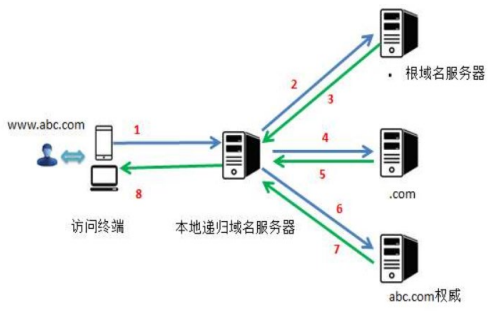
机密性标签可以保护敏感数据不被非法或恶意用户读取;而完整性标签可以保护重要信息或存储单元免受不可信或恶意用户的破坏。

2.1.4 基于该思想的实例

双向 DNS 检查

客户端向域名请求一个 IP,域名提供逆向 IP 解析服务返回给客户,DNS 正向分析和反向分析同时进行,从 IP 获取域名,再从域名获取 IP,再检验两个 IP,确保 IP 地址的一致性,保证了信息流(该域名 IP)传输的正确性,从而避免了 DNS 欺骗等诱导信息流出现安全漏洞的恶意行为。

图 7 双向 DNS 检查



数字签名(以 MD5 文件传输标识为例)

如何确保网络传输文件下载的一致性、正确性?这就需要用到一种文件的标识,当今制作这种文件标识-数字签名的加密算法是 MD5 算法,MD5 算法广泛应用在传输文件的数字签名应用上,这种定长单向散列算法,速度快,不易破解,冲突少,是信息流控制中最适合的保护方法。通过附加 MD5 运算出来的附加数字签名添加到文件中,对整个文件内容进行保护和验证,从而控制了信息流的流向和传输一致性。

2.1.5 安全性与精确性

设 F 是一个信息流系统中所有可能信息流的集合;

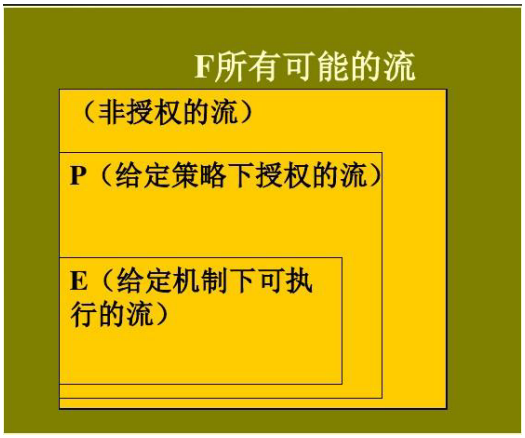
E 是在给定的流控制机制下可执行的 F 的子集;

P 是给定策略下授权的流;

如果 P 包含 E,即所有可执行的流都是授权的,那么系统是安全的。

如果 E=P.即所有授权的流都是可执行的,那么安全系统是精确的。

图 8 双向 DNS 检查



2.2 信息流控制机制综述

信息流和控制机制的设计和实施方法:

一是编译时刻与验证信息流的安全性;

二是执行时刻验证与检查信息流的安全性;

三是利用程序正确性证明技术去验证程序的安全性,这种方法建立在“格模型与 Hoar 功能的正确性推理系统”上的信息流演绎系统,可以给出一个更为精确的安全性实施机制,但这一部分内容要求的理论基础比较深。

2.2.1 编译时刻检查与验证信息流的安全性

基于编译器的机制检验程序中的信息流是否经过授权。这类机制要判断程序中的信息流是否“可能”违反给定的信息流策略。这种判定不是精确的,安全的信息流路径也可能会标志为违反策略;但这种判定是安全的,即未经授权的信息流路径一定会被检测出来。

【定义】如果一个语句集合中的信息流不违反某信息流策略,则称此语句集合关于某信息流策略是有安全保证的。

【类型声明】

使用语言结构来表示可流入变量的信息的类型集合:

x: integer class{ A, B }

表示 x 是一个整数变量,且来自安全类型 A 和 B 的数据可以流入 x。类型是静态分配而不是动态分配的。把安全类型视为格,这表示 x 的类型必须至少是类型 A 和 B 的最小上限,即 lub(A,B)≤ x。

两种不同的类型, Low 和 High 分别代表了格中的最大下限与最小上限。所有的常量都属于 Low 类型。

信息可以通过参数输入或输出子程序,可将参数分为三种类型:输入参数(数据可以通过它输入子程序)、输出参数(数据可以通过它输出子程序)和输入/输出参数(数据既可以通过它输入也可以通过它输出)。

输入参数的类型就是实际参数的类型：

is:type class{is}

设 r1,…,rp 为输入和输入/输出变量的集合,它们的信息流向输出变量 os。类型声明必须体现出这一点：

os:type class{r1,…,rp}

输入/输出参数和输出参数一样，除了初始值（作为输入）影响允许的安全等级外。同样，r1,…,rp 如上定义：

ios:type class{r1,…,rp,ios}

【程序语句】

一个程序包含若干类型的语句，典型的语句有：**赋值语句；复合语句；条件语句；迭代语句；goto 语句；过程调用；函数调用；输入/输出语句**。除函数调用通过过程调用模仿，输入/输出语句使用赋值语句模仿，分别独立地考虑其余语句的安全性。

下面以赋值语句、复合语句和迭代语句为例展开说明：

1.赋值语句

一个赋值语句有如下形式：

y:=f(x1,…,xn)

其中 y 和 x1,…,xn 是变量，而 f 是这些变量的函数。信息从每一个 xi 流向 y。因此要使得信息流是安全的，就要求 lub{x1,…,xn}≤y

例如,语句 x:=y+z,则使得信息流为安全的必要条件是 lub{y,z}≤x

2.复合语句

复合语句有以下形式：

begin

S1;

…

Sn;

end;

其中每一个 Si 是一条语句。如果每条语句的信息流都是安全的，则复合语句中的信息流也就是安全的。因此，使得复合语句中信息流为安全的必要条件是：

S1 安全

…

Sn 安全

3.迭代语句

迭代语句迭代语句有如下形式:

while f(x1, ..., xn) do .

S;

其中 x,…,xn 是变量,f 是这些变量的(布尔)函数。除了循环执行外,这就是一个条件语句，所以对条件语句中信息流为安全的条件在此也适用。

要处理循环，首先注意到循环的次数致使信息通过对 S 中的变量赋值而流动。循环的次数由变量 x1,…,xn 的值控制,所以信息从这些变量流向 S 中的赋值对象,但这可通过验证条件语句的信息流要求而检测出来。

然而,如果程序跳不出这个迭代语句,则在此循环后的语句将不会被执行。在这种情况下,信息就通过语句的“不执行”而从变量 x1,…,xn 中流出。因此,安全信息流要求循环必须终止。

这样,安全信息流的**必要条件**就是:

迭代语句终止

S 是安全的

lub{x1,…,xn}≤glb{y|y 是 S 中的一个赋值对象}

【可靠性】

Denning 和 Denning,Andrews 和 Reitman,还有其他一些学者将他们关于安全的论点建立在一种直觉之上,即对某些安全策略,安全信息流的组合可以产生另一条安全的信息流。然而,他们从来没有形式化地证明这一直觉。Volpano,Irvine 和 Smith 将上述信息流分析的语义表达为类型的集合,并将验证特定信息流的发生等价为验证类型的正确使用。在此情况下,验证有效信息流等价于验证变量与表达式的类型符合安全策略所规定的语义。

【总结】

设 x 和 y 是程序的两个变量,且 x 的标号支配 y 的标号。如果在程序执行过程中,x 的值不能影响 y 的值,则一个信息流规则集合是可靠的。Volpano,Irvine 和 Smith 使用基于语言的技巧证明,给定一个等价于以上讨论的验证规则的类型系统,无类型错误的所有程序拥有以上描述的不干涉属性。因此，Den-ning 的信息流验证规则及 Andrews 和 Reitman 的信息流验证规则都是可靠的。

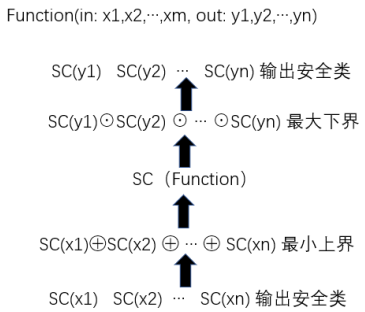
2.2.2 执行时刻验证与检查信息流的安全性

【编译时检查的一些缺陷】

由于编译时检查过于严格，会让一些有效的语句无法被通过检查在运行时检查信息流的安全性。运行时检查对于包含显式信息流的语句，可以达到这个目的。

【例子】

图 9 执行时刻验证实例



比如的赋值语句，y=f(x1,…,xn)中，只需要 y 的安全性比 x1-xn 的任意一个安全性还高（即 SC(y)>=SC(x1)-SC(xn)的 least upper bound，SC 表示安全性）即可通过检查。然而如果是在编译时刻检查，即使满足条件语句也可能会不通过

【缺陷】

然而存在着隐式（implicit）的信息流，会使得这样的检查方式让一些不和规定的语句通过检查。

if (x==1) then y=a;

信息从 x 流向 y，如果 x!=1 运行时并不会发现信息有从 x 流向 y，因为 y 所在的语句直接被跳过了，所以即使 y 的安全性低于 x 的安全性,语句仍然会会通过检查。但是，实际上有信息从 x 传到 y，即 x!=1。

【解决方案】

Fenton 开发了一种称为数据标记机的抽象机器来研究隐式信息流在执行时的处理。

图 10　数据标识机指令

指令	执行效果（pc是程序计数器，SC表示安全性）
x=x+1；	If(SC(p)<=SC(x)) x=x+1; else skip;

If(x==0) goto n
Else x=x-1;
	If(x==0) {
	push(pc,SC(pc)); SC(pc)=SC(pc)⊕SC(x); pc=n;
	}
	Else {
	if(SC(pc)<=SC(x) x=x-1; else skip;
	}

If' (x==0) goto n
Else x=x-1;
//该语句不保存PC进栈，如果分之发生，则PC的安全类型比x高,所以增加从x到PC的信息不回改变PC的安全类型
	If(x==0) {
	if(SC(x)<=SC(pc) pc=n; else skip;
	}
	Else {
	if(SC(pc)<=SC(x) x=x-1; else skip;
	}

Return;	Pop(pc,SC(pc))
Halt;	If empty stack than Halt;

数据标记机（Data Mark）：

是一台扩充的 Minsky 机，它包含了每一类寄存器（存储器单元）的安全类所作的标记

这种机器标记了变量以及 PC（program counter）

PC：用于存放下一条指令所在单元的地址的地方，相当于一个指针，可以遍历整个程序。

标记了 PC 的好处在于，PC 可以让隐式信息流作为显示信息流来处理，因为程序的分支仅仅是对 PC 的赋值。

因此每当程序进入一个分支后，PC 的安全性就会改变，一旦从分支退出，回到之前的“主干”后，PC 的安全性就不再受分支的影响。

让我们回到最开始的隐式信息流的例子

if (x==1) then y=a; 等价于

if (x==1) then {push(PC); PC=lub(PC,x);if PC<=y then y=a else skip;}

这里下划线表示某一变量的安全性。

我们可以把条件语句后面的执行句看成一个分支，当进入分支时，先把 PC 代表的安全级别压入程序栈，并让 PC 的安全性等于 x 与 PC 中较大者，这时候将 PC 和 y 的安全性相比较，如果 y 的安全性较高，允许执行，否则直接跳过该语句。因此实际上信息流仅在 y 的安全性比 x 的安全性高时才允许执行。当整个分支结束后，把 PC pop 出来。

调用某个语句，就将 PC 的 Push 进程序栈。返回，就将 PC 从程序栈 Pop 出来。同时还有停机指令 halt，要让程序栈先清空再执行停机，防止用户在停机后通过查看程序栈而获取信息。

【总结】

数据标记机标记了变量以及 PC，显示了他们的安全性，从而使得隐式的信息流可以转换成显示的信息流，PC 在这中间承担了一个“中间变量”的功能。

2.3 信息流检测方法

构造一个既满足安全性又满足精确性的机制是困难的。更多的对信息流提供保护支持的是进行安全检测，这就导致信息流检测方法在信息流识别中特别重要。

2.3.1 语法信息流方法

【Denning 的信息流格模型】

是语法信息流方法中最著名的一个，也是最初的系统分析隐蔽通道的方法。

【分析步骤】

① 将信息流语义附加在每个语句之后。例如，当 b 不为常数时，赋值语句 a:=b 产生由 b 到 a 的信息流，用 a←b 表示，并称之为“明流”或者显式信息流；类似地条件语句产生暗流或者隐式信息流。例如,if x=a then y:=b else z:=c 产生的暗流是 y←x 和 z←x.这时，同时存在明流 y←b 和 z←c。

② 定义安全信息流策略，例如“如果信息从变量 b 流向变量 a,则 a 的安全级必须支配变量 b 的安全级”。

③将流策略应用于形式化顶层规范或源代码，生成信息流公式。例如 a:=b 的流公式为 a≥b ，其中 x 表示变量 x 的安全级。

④证明流公式的正确性。如果无法证明某个流公式的正确性，则需要进一步对语句进行语义分析，并判断该信息流：(a) 是非法流还是伪非法流；(b) 是否能够产生真实隐蔽通道，而不只是潜在隐蔽通道。

2.3.2 语义信息流分析法

1990 年,Tsai, Gligor 和 Chandersekaran 对语法信息流方法作了重大改进，增加了语义分析。

【改进】Tsai 等人提出了一种标识隐蔽存储通道的新方法,基于：(1) 分析编程语言的语义、代码和内核中使用的数据结构，发现变量的可见性/可修改性；(2) 解决内核变量的别名问题,确定内核变量的间接可修改性；(3)对源代码进行信息流分析,确定内核变 量的间接可见性。他们的语义信息流方法在 Secure Xenix 系统的应用中获得成功。

【分析步骤】① 选择用于隐蔽通道分析的内核原语；② 确定内核变量的可见性/可修改性；(i) 通过语义分析,确定内核变量的直接可见性/可修 改性；(ii) 对每个原语生成一个“函数调用依赖关系”集合 FCD；(iii)通过信息流分析,确定内核变量的间接可见性；(iv) 在每个原语中解决变量别名问题；(v) 标识在原 语间共享的用户进程可见/可修改的变量,消除局部变量； ③ 分析共享变量,并标识隐蔽存储通道。

【特点】

1、以源代码信息流分析为基础，不需要系统有严格的顶层形式化描述和一致性证明

2、以源代码信息流分析为基础，能找出更多、更全的隐蔽存储信道

3、对源代码信息流经过约化，只关心共享变量有关的信息流，大大减少了信息流分析的工作量；

4、对发送方和接收方进程可能存在的函数序列有一套明确的分类和计算方法,需要计算到函数序列的哪一级可以随意控制。

2.3.3 改进的语法信息流检测方法

格是建立在群概念上的一种数据结构,一个格是一个集合 S 和关系 R 的组合，满足如下条件：R 是自反的、反对称和传递的；对任意 s,t∈S,存在最大下界；对任意 s,t∈S,存在最小上界。

典型数据模型：**数据标记机**

在程序中引入 PC 程序计数器来处理隐式信息流问题，即 pc 只是对程序分支的一种计算赋值。push (x,x)表示将变量 x 及安全类型 x 压入栈中，而 pop(x,x)表示从程序栈中弹出栈顶数据及安全类型。程序计数器的引入能从程序执行的层次 上控制信息的流动，也能解决了程序执行过程中隐式信息流的发生。

数据标记机的保护特性是可以在实际系统中实现的。可用具有标记存储器的单累加器机器为例来说明。在存储器地址的标记区存放数据客体的安全级，一个可变的标记 SC(acc)表示在累加器中信息的安全类。和标记机一样，用堆栈存储（pc,SC（pc））,其中 SC(pc)仍表示程序计数器的安全类。

3 隐蔽通道（Convert Channel）

3.1 隐蔽通道简介

3.1.1 隐蔽通道定义及解释

【定义】

隐蔽通道是一种计算机攻击，它利用现有媒介的结构，通过现有的信息通道或网络传输对象来传递信息，从而以小部分传输数据。这使得通过隐蔽通道的传输几乎无法被管理员和用户检测到。

隐蔽通道已被用于从高安全度的系统中偷窃数据。

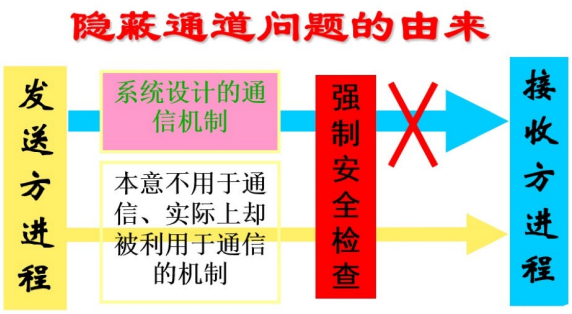
【具体解释】

通过使用填充时的空间或网络数据包传输的其他部分中的某些可用空间来创建隐蔽通道。隐蔽通道会使用任何可以将数据添加到数据流而不影响正在传输的数据主体的方法。这允许隐蔽接收器不创建任何类型的数据跟踪而从系统中提取数据。单个数据包可能仅包含秘密数据流的一或两个比特，这使得检测非常困难。

创建隐蔽通道需要一些巧妙的编程，并且通信源头文件系统的访问权限至关重要。这意味着只能通过病毒感染或具有对系统的管理或其他授权访问权限的编程工作来启动隐蔽通道。

隐蔽通道分析是检测隐蔽通道的几种方法之一。系统性能下降可以用来显示秘密通道的使用，但是随着计算机的发展，与处理的数据量相比，下降并不明显。这使得检测更加困难。防范隐蔽通道攻击的主要方法是检查在源计算机上运行的源代码，并监视相关系统对资源的使用。

图 11 隐蔽通道



3.1.2 隐蔽通道的分类

在现实的系统实践中，可以按照不同的特性对隐蔽通道进行分类:

【按照利用的存放信息的方式】,

可以分为存储隐蔽通道 (storage covert channel) 和时间隐蔽通道 (timing covert channel) ;

存储隐蔽通道：对它的利用“包括一个进程对一个存储变量的直接的或者间接的修改，以及另一个进程对这些存储变量的直接的或者间接的读取”。

时间隐蔽通道：它的利用“包括一个进程调制它自己关于一个系统资源的使用规律 (比如 CPU 使用时间)，这样的操作使得第二个进程对于某个操作响应的时间发生变化”。

【按照信息通道的特性,】

可以分为有噪声隐蔽通道 (noisy covert channel) 和无噪声隐蔽通道 (noiseless covert channel) ;

【按照共享变量的利用方式,】

可以分为聚集的隐蔽通道 (aggregated covert channel) 和非聚集隐蔽通道 (non-aggregated covert channel) 。

其中和隐蔽通道发现密切相关的是第一种分类方式，也就是存储 / 时间隐蔽通道。

一般地，定义一个存储隐蔽通道是这样的一种隐蔽通道，当对它的利用“包括一个进程对一个存储变量的直接的或者间接的修改，以及另一个进程对这些存储变量的直接的或者间接的读取” 。而对时间隐蔽通道则是指的它的利用“包括一个进程调制它自己关于一个系统资源的使用规律(比如 CPU 使用时间)，这样的操作使得第二个进程对于某个操作响应的时间发生变化”。

3.2 两种隐蔽通道存在的条件：

3.2.1 存储隐蔽通道

1. 发送和接收进程必须对某个共享资源的同一个属性有访问权。

2. 一定有某种方式使得发送进程能够迫使这个属性去改变。

3. 一定有某种方式使得接收进程能够探测到这个属性的改变。

4. 一定存在某种机制使得发送/接收双方能够同步发送事件，这个可以是另外一个带宽较小的通道。

3.2.2 时间隐蔽通道

1. 发送和接收进程必须对某个共享资源的同一个属性有访问权。

2. 发送和接收进程必须有一个统一的时间参考， 比如一个实时钟。

3. 发送进程必须能够调制接收者的响应时间来表示一个属性的改变。

4. 一定存在某种机制使得发送/接收双方能够同步发送事件。

3.2.3 隐蔽通道存在条件总结

可以看出隐蔽通道存在的一个共同的前提条件是资源的共享， 这个是一个实际有用的系统中所必须具有的一个特性，所以也就成为隐蔽通道存在的一个基础。而对于两种隐蔽通道的区别之处也就是对时钟的参考， 很多时候， 由于计算机中的时间概念可以被看作是一个具有相对位置关系的事件序列， 进而时间的获取也可以来源于对存储变量的改变的观察， 所以如同上面指出的， 它们之间的界限是模糊的。换句话说， 一个通道是存储隐蔽通道还是时间隐蔽通道取决于人们对时间概念的看待。事实上， 系统中任意一个隐蔽通道都可以分解成两个不同的可以互相区别的事件序列的组合。但是一般系统当中，有些机制是被固定看作时间的 （如时钟发生器， 某个外在的有固定频率的 I/O 事件） ， 而另一些则更倾向于看作存储变量 （如文件锁， 磁盘臂的位置等等） ， 这样也就有了很多一般意义上的存储隐蔽通道和时间隐蔽通道。

3.3 隐蔽通道实例

3.3.1 存储隐蔽通道实例

1）共享资源耗尽通道

假设系统中一个有限的资源，如系统的虚拟内存 M，被 S 和 R 共享使用，TCB 原语 alloc（size）用于用户向系统请求分配大小为 size 的内存，如果当前空闲虚拟内存量 empty（M）≥size 那么分配成功并返回内存指针，否则返回为 null。一种情形，在 S 运行的时候，它通过频繁申请大小为 size′ 的小块内存直到 empty（M）<size′，然后 S 等待推出运行。另一种情形，S 释放所有它占用的内存块，然后推出执行。等到 R 进入运行，R 进行一次 alloc（size′）的操作，如果这个操作的返回值为 null，那么，R 就知道，上一次 S 的运行是采用的情形一，否则就是采用的情形二。如果事先 S 和 R 定义第一种情形代表“0”，第二种情形代表“1”，那么每次 S 和 R 的运行就完成了一位的信息传递。

2）客体存在通道

一个由 S 产生或者消除的客体，它的存在与否能通过某种方式被 R 所感知。典型的例子如文件存在通道。在 MAC 的访问控制机制中，假设一个目录与 R 同级别，那么如果该目录为空就能够被 R 所删除，否则删除失败。S 通过在该目录中创建文件使得目录非空，或者删除文件使得目录为空。这样，S 的不同操作就能被 R 感知，进而每次 S 和 R 的执行可以用来传送一位的信息。

3.4 如何防止隐通道例

想要防止隐蔽通道关键是找到它，信息流的传播一定会通过流的信道，与正规信道相比，隐蔽通道的隐藏性很高，难以被发现，但其实其所使用的攻击手段并不会太直接，由于隐蔽通道本身对自己也有一定的限制，所以找到隐蔽通道是防止隐蔽通道的关键。

3.4.1 非相干性分析

非相关性模型从本质上形式化了这样一个想法：在一个安全系统中，一个用户不能意识到任何不由它所支配的用户（对于多级安全系统，就是安全级别比它高的用户）的任何操作。它将一个安全系统的 TCB 抽象成一个状态机（state machine），假设 A 和 B 是这样一个抽象 TCB 的两个用户，如果 w 是对这个状态机从初始状态的输入，并且这个输入的最后一个操作是来自 B 的，那么定义 B（w）为 B 由最后一个输入获得的输出。同时 w/A 定义为将 w 中来自 A 的输入的删除（裁减）以后得到的子串。这样，定义 A 非相干于 B，当且仅当对于所有可能的以 B 的输入结尾的输入串 w，B（w）= B（w/A）。如果一个多级系统中不存在隐蔽通道，那么任何一个用户都应该非相干于级别比它低的用户。

3.4.2 共享资源矩阵(Shared Resource Matrix，SRM)

有一定局限性，但仍然是当前最实用的隐蔽通道发现方法，主要包含以下步骤：

①根据具体的系统描述（形式化的、非形式化的、源码）分析所有的 TCB 原语，构造一个基本的共享资源矩阵。矩阵的绘制方法是：用户可见的 TCB 原语标识矩阵的行，用户可见或者可改变的 TCB 的变量标识矩阵的列，以 R 或者 M 作为矩阵的元素分别来代表一个 TCB 变量是否能被一个 TCB 原语所读取或者改变。

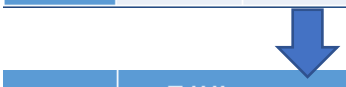
②构造这个基本共享资源矩阵的传递闭包。构造的方法按照这样的原则：假设一个 TCB 原语 P1 对 TCB 变量 V 的有读取操作，P2 对 V 有改变的操作。那么，P1 对所有 P2 用以改变 V 所读取的变量具有间接读的操作。对于共享资源矩阵来说就是，对于任何一个 R 值的成员检查是否在同一行有 M 成员，如果有的话，将 M 所在列的所有 R 成员添加到原先“R”所在列的对应行上面去。重复这样一个过程，直到不再有新的 R 元素被添加。

③分析上一步所得到的矩阵。如果某个进程能够利用 TCB 原语读一个变量，而另一个的进程能够通过 TCB 原语写同一个变量的话，那么就存在一个的信息通道。对于这个矩阵的分析结果中可能存在的隐蔽通道在一些情况下是不能成为隐蔽通道的：(1) 发送方和接收方是同一个进程；(2) 这个通道不能用来传送有用的信息；(3) 这个通道本身按照安全策略来说是合法的。剩下的通道被称为是潜在隐蔽通道。

对于上一步所发现的潜在隐蔽通道，分析这个通道所涉及到的矩阵的元素，进而试图构造这个隐蔽通道实际使用的情形。在有些情况下，一个通道的实际利用情形并不存在，所以它也就不能被实际利用，进而成为一个真正的隐蔽通道。

SRM 实例分析：

变量	TCB函数接口			
	S_OR	S_OW	S_CR	S_UL
F_b			RW	W
f.r	W			R
f.w		W	W	R



变量	TCB函数接口			
	S_OR	S_OW	S_CR	S_UL
F_b			RW	WR
f.r	W		R	R
f.w		W	RW	R

STEP1-构建共享资源矩阵和传递闭包

变量	TCB函数接口			
	S_OR	S_OW	S_CR	S_UL
F_b			RW	WR
f.r	W		R	R
f.w		W	RW	R

STEP2-分析矩阵

3.4.3 隐蔽流树(Covert flow trees，CFT)

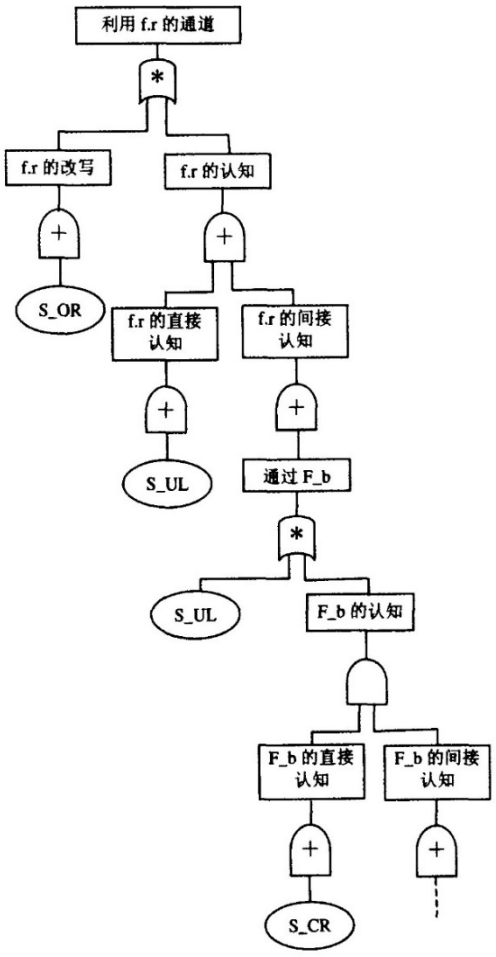
隐蔽流树分析的第一步是根据 TCB 的说明或者源码对每个 TCB 原语构造 3 个列表：引用列表、改写列表和返回列表，分别对应了这个原语所读取的变量、所能改写的变量以及返回结果给用户的时候引用的变量。这一步类似于 SRM 方法中构造基本的资源共享矩阵。在具体构造一个 CFT 的时候，选择一个 TCB 的变量作为这个 CFT 分析的焦点，以之

作为根节点。然后， 根节点分做两个分支， 左边的分支代表发送者对这个变量的改写动作 (modification) 可能使用的 TCB 原语， 右边分支代表接收者对这个变量的改变的认知 (recognition) 所使用的 TCB 原语。紧接着的工作是继续扩展右边的认知分支。扩展的原则是:如果这个变量出现在某个原语的返回列表中， 那么将那个原语添加成为一个“直接认知” (direct recognition) 的分支，如果它不出现在任何一个原语的返回列表中，那么“失败”的页节点作为“直接认知”。如果这个变量出现在一个原语的引用列表中， 并且这个原语的改写列表非空， 那么这个原语被作为一个“间接认知” (interred-recognition) 的分支被添加。然后，添加这个原语中改写列表中的每一个变量作为一个新的认知右边分支，这个原语作为左分支。对于每个新的这样的认知分支， 不断递归地进行以上的操作， 直到所有的叶节点都是“直接认知”或者“失败”。

变量	TCB 函数接口			
	S_OR	S_OW	S_CR	S_UL
F_b			RW	WR
f.r	W		R	R
f.w		W	RW	R

通过穿越构造完成的 CFT，可以发现潜在的隐蔽通道。穿越的操作将 CFT 中所有原语的节点分成两个列表， 一个包含了改写这个变量的 TCB 原语， 另一个包含了直接或者间接认知这个变量的原语。这分别来自两个列表中的任意两个原语的组合构成了潜在的隐蔽通道。下面以上一个例子中的 TCB 原语和变量来说明 CFT 的构造，如下图。

图 12 CFT 构造



3.4.4 信息流分析

基于语言的信息流分析 (language based information flow) 是一个拥有庞大分支的用于对安全属性建模或者证明的方法体系。它的通用性使得它同样可以应用于隐蔽通道的发现。

这样的方法通过对每条系统的 TCB 描述或者源码语句添加信息流语义， 然后通过这些语义分析系统中时候是否存在非法的信息流。举例来说对于 Pascal 语句假设 B、D、E 为常数， a、c 为变量 if a=B then c:=D else c:=E 就导致了一个从 a 到 c 的信息流动。在具体分析一个多级别系统的时候， 分析者根据系统的安全策略， 将每个变量标签以一个安全级别。假设我们用 level (x) 来表示的安全级别。那么在以上的一个信息流中， 如果 level (a) >level (c) 那么这个信息流就是不合法的。当在系统分析的时候遇到这样非法的信息流的时候， 就有可能存在一个隐蔽通道。造成这个非法信息流的原因 (是那些 TCB 原语造成的) 需要进一步的系统分析才能达到。至于这个隐蔽通道是否能被实际利用需要看这个通道时候能找到对应的实际利用的情况。

3.4.5 四种方法的特点分析

特性 方法	作用的对象	虚报率	自动化程度	对实际利用的指导作用	增量性
非相干性分析	FTLS	无	存在自动化分析工具，但不完善	指导意义很小，需要结合其它方法	是
共享资源矩阵	FTLS、非形式化 TCB 说明、源码	较低	对于源码的分析不存在自动化工具	有一定的指导意义	否
隐蔽流树	同上	较低	同上	比较直观地反映通道的利用情况	否
信息流分析	FTLS、源码	较高或者不稳定	存在自动化分析工具，但不完善	指导意义不大	是

3.4.6 发现隐蔽通道后的处理

【消除】删除隐蔽通道使用的共享变量，或防止其受到修改，以确保无法通过该变量传递信息。

【带宽限制】引入噪音或增加延时以增加通过隐蔽通道传递信息的难度或减少信息传递带宽。

【审计】并不消除隐蔽通道，而是对其施以监视，以期威慑试图通过隐蔽通道传递信息的用户。

3.5 附：LINUX下实现隐蔽通道信息传输

3.5.1 实验步骤

第一步，先以 root 进入系统，通过证书认证获得特权。

第二步，root 用户创建 2 个用户，分别取名为 high 用户和 low 用户。并设置：1）low 用户的默认安全级别是 SYS_PUBLIC，主目录是/home/low 2）high 用户的安全级别是 SYS_ADMIN，主目录是/home/high。

第三步，root 用户对 High 和 low 这两个用户的主目录安全级别分别设置为默认安全级别 1）/home/high 的安全级别设置为 SYS_ADMIN；2）/home/low 的安全级别设置为 SYS_PUBLIC.

第四步，分别从两个不同终端以 high 和 low 用户分别登录系统。

第五步，high 用户和 Low 用户分别将其主目录的自主发囊文控制权限打开，1）high 用户执行：chmod 0777 /home/high;2)low 用户执行：chmod 0777 /home/low

第六步,测试一下目前系统的 MAC 策略是否正确实施,即：1）是否不可以上读，即 low 用户不能直接读取 high 用户创建的高安全级别文件 in。

2）是否可以下读，但不可下写：high 用户可以读取 low 用户建立的文件 out，但不能在 low 用户主目录下创建文件，更不能直接拷贝其高安全级别文件 in 内容给 low 用户的文件 out

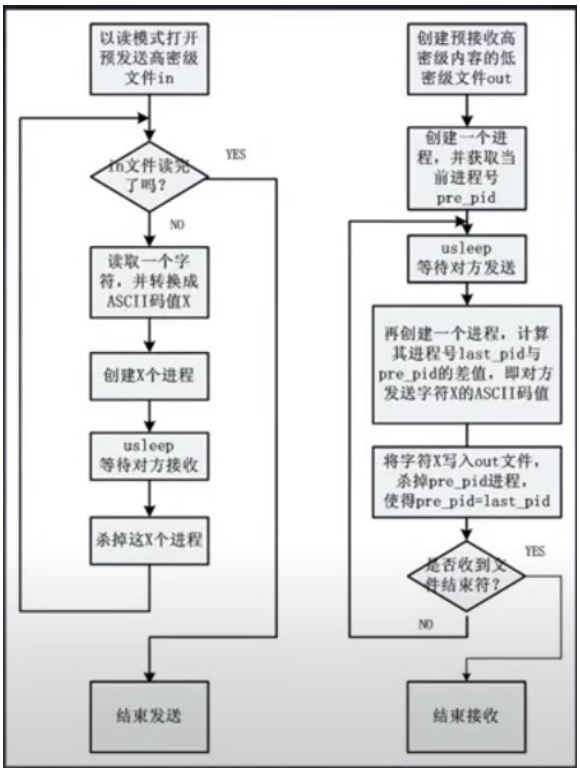
第七步，high 用户在其主目录下，利用已经标识的隐蔽存储通道，构建秘密信息 in 的发送程序 sender.c，并通过 gcc 编译链接生成 sender.exe 可执行文件

第八步，low 用户在其主目录下，利用已经标识的隐蔽存储通道，构建秘密信息 in 的接受程序 receiver.c,并通过 gcc 编译链接生成 receiver.exe 可执行文件。

第九步，通常的，先后有 Low 用户启动运行 receiver.exe，由 high 用户启动允许 sender.exe

第十步，运行过程中或结束后，检查 low 用户目录下的 out 文件是否已经包含了 in 文件中的秘密信息。

图 13 实验步骤流程图



3.5.2 利用 pid 实现隐蔽通道思路分析

【实验目标】通过被标识的一种隐蔽存储通道，在高安全级别用户 high 下执行 sender 程序，在低安全级别用户 low 下执行 receiver 程序，将 high 用户下的文件 in 中的秘密内容成功地发送给 low 用户，并保存在 low 的 out 文件中。

【实验环境】

SELinux+gcc

【实验简介】

信道名称：进程标识符信道

信道类型：持久变量型

中介变量：last_pid

存在条件：系统中所有的进程共享同一个进程标识符符列表。无论系统采用何种安全策略，搞安全级别进程与低安全级别进程都要共享这个列表，并且可以通过建立子进程得知当前可用进程标识符信道的值。如果系统的进程标识符信道表是干净的，即没有其他进程消耗进程标识符信道表，高低进程就可以根据进程标识符信道增长的情况了解其他进程的工作情况。

发送方动作：发送方读取机密文件，然后，根据该文件内容的当前自己的 ASCII 码决定创建的子进程数目，待接收方读取信息后，删除这些子进程。

接收方动作：创建子进程，得到当前的进程标识符信道，然后与上次创建子进程得到的进程标识符信道对比，可以知道其他进程创建的子进程数目，解释所得到的信息。

噪音情况：高噪音。系统中存在的任何其他进程都可能创建子进程，从而导致接收方错误解释所得到的信息。如果再安排在其他用户不工作的时刻（比如午夜 12 点）传输信息，就可能获得较大的带宽。

带宽估计：最大考验达到 160byte/s

处理措施：采用随机化进程标识符信道或者加入 idle 进程，该进程随机创建子进程

【思路分析】

本程序主要利用了 Linux 的 pid 进程标识符，可以利用 ps -aux 指令来查看自己 Linux 账户下的进程，对于实验较为重要的指令还有 sleep 和 usleep 指令，这两个指令的作用是使当前运行的程序等待 n 秒，在这里 sender 等待 1s 的过程中，receiver 正在获得藏在 firstpid 和 lastpid 中的 ASCII 码的值，sleep 起到的作用是等待另一方接受执行程序，并杀死进程后，进行下一个循环。实验了两个实际进程中的跳转，CPU 执行程序的跳转实现了隐蔽通道下秘密传输信息流信息的效果，最终也成功将 Alice 这一信息写入了 low 用户的 out 文件中，违反了安全机制的 MAC 策略。

注：输入的信息以！结尾判断信息是否接受完毕，在实验前一定要将权限按实验步骤设置正确，因为是验证信息安全隐蔽通道攻击的实验，权限要是不正确，就失去了实验必要。第六步的测试 MAC 策略是否正确是必不可少的，因为用 sudo 命令创建的用户，其会自动按照预设的权限将 umask 设置为 0022，导致写出的/home/high 或者/home/low 的权限对 group 缺损。特别注意：在进行本次实验时，建议关闭除了两个终端以外的其他进程，不然会对实验产生较大影响。

3.5.3 实验过程展示

完整的实验可以参照展示时提供的视频或者按照步骤自行实验均可。详细视频可见：
https://www.bilibili.com/video/BV1Hf4y1S7ge

3.5.4 Receiver.c 源码

```
1.  //Receiver.c
2.  #include<stdio.h>
3.  #include<sys/types.h>
4.  #include<signal.h>
5.  #include<sys/stat.h>
6.  #include<fcntl.h>
7.  #include<unistd.h>
8.
9.   int main(){
10.  int i,ret,k;
11.  Chat ch;
12.  pid_t firstpid,lastpid,testpid;//标注进程标识符
13.  int fd=open("/home/low/test2",O_CREAT|O_RDWR);
14.  //生成test2保存test中的密文k=1;
15.   if((firstpid=fork())==0)  {
16.      //接收第一个进程，把进程号赋予firstpid
17.      For(i=1;i<=ch;i++){//把ch转化为ACSII码
18.      Sleep(1);
19.      exit(0); }
          //等待Sleep(1)时间
20.       Sleep(1);
21.  do{
22.  Sleep(3);
23.  if((lastpid=fork())==0){  // 接收最后一个进程号，并把它赋予lastpid
24.  Sleep(1);Exit(0);
25. }
26.  else if(lastpid<0){ //如果进程号小于0，报错
27.  perror("error!");exit(-1)}
28.  printf("firstpid=%d,lastpid=%d\n,firstpid,lastpid");
29.  if(k==1) ch=lastpid-firstpid-2;
30.  else ch=lastpid-firstpid-1;
31. printf("Receiving the %c\n",ch); //输出密文
32. write(fd,&ch,1);
33. ret=kill(firstpid,SIGKILL); //杀死所有进程
34.  if(ret==-1){
35.  perror("kill error");Exit(-1);}
36. firstpid=lastpid    //首位进程号相等
38. k++;}while(ch!='!');
36. ch='\n'; write(fd,&ch,1);
37. chmod("/home/low/test2",S_IRUSR|S_IWUSR);close(fd);}
```

3.5.5 Sender.c 源码

```
1.  //Sender.c
2.  #include<stdio.h>
3.  #include<sys/types.h>
4.  #include<signal.h>
5.  #include<sys/stat.h>
6.  #include<fcntl.h>
7.  #include<unistd.h>
8.
9.   int main(){
10.  int i,ret;
11.  Chat ch;
12.  Pid_t  s_pid[256];
13.  Int fd=open("/home/high/test",0_RDWR); //以读写方式打开test文件
14.  Ret=read(fd,&ch,1);//读test第一个字符
15.  While(1){//开始循环
16.       Printf("sending %c\n",ch);
17.       For(i=1;i<=ch;i++){//把ch转化为ACSII码
18.       If((s_pid[i]=fork())==0){//通过循环生成ch个进程，将进程号付给s_pid[i]数组
19.         Sleep(1);
           //等待Sleep(1)时间
20.         Exit(0);                          //退出
21.       }
22.  Else if(s_pid[i]<0){   //如果s_pid[i]的进程号小于0，报错
23.  Perror("error!");
24.  Return -1;
25. }}
26.  If(ch=='!')break;        //当遇到"！"结束程序
27.  Read(fd,&ch,1);}
28.  Printf("sending done!\n");
29.  Close(fd);}
```

参考文献:

[1] Matthew Bishop. Computer Security: Art and Science. Addison-Wesley.

[2] 吴泽智, 陈性元, 杨智, 杜学绘. 信息流控制研究进展. 软件学报, 2017, 28(1):135-159.

[3] 杨明欣, 王玉恒. 隐通道信息流检测方法综述. 2011.

[4] J. D. Guttman and P. D. Rowe, "A Cut Principle for Information Flow," 2015 IEEE 28th Computer Security Foundations Symposium, Verona, 2015, pp. 107-121.

[5] Smith G. (2007) Principles of Secure Information Flow Analysis. In: Christodorescu M., Jha S., Maughan D., Song D., Wang C. (eds) Malware Detection. Advances in Information Security, vol 27. Springer, Boston, MA

[6] Project webpage.https://github.com/vanturman/Convert-Channel.

收稿日期：2020-04-20；**修返日期**：