THE COVERT CHANNEL PROBLEM

by

Yahia A. Fadlalla and Rodney H. Cooper

TR96-111, August 1996

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
www: http://www.cs.unb.ca

The Covert Channel Problem

by

Yahia A. Fadlalla
and
Rodney H. Cooper

Faculty of Computer Science
Univeristy of New Bruswick, Fredericton, N.B.
August 1996

Table of Contents

List of Figures

ABSTRACT

The Canadian Trusted Computer Product Evaluation Criteria (CTCPEC), the U.S. Trusted Computer System Evaluation Criteria (TCSEC), the Information Technology Security Evaluation Criteria – the harmonized criteria of France, Germany, The Netherlands and The United Kingdom (ITSEC), and other criteria have been developed to aid in the analysis of computer systems to ensure that two processes at different security levels cannot directly communicate information in violation of security policies. Despite the guidelines in these criteria along with other techniques, many systems suffer from processes that communicate by means of *covert channels*.

In this report, covert channels are defined. A review of the related research activities is given along with methodologies of how to detect covert channels and how to handle them. Since the covert channel problem encroaches on other areas in computer science such as communication theory, networks, and databases, these relationships will be explored with emphasis on multilevel database systems.

1 INTRODUCTION

A large number of databases in departments of defense, the intelligence community, and civilian government agencies contain data that are classified or sensitive to different security levels. All database users are assigned security clearances. It is the responsibility of a multilevel secure database management system (MLS-DBMS) to assure that each user gains access to only those data for which he has the proper clearances. Although their assigning procedures are less formal than those of the government, most commercial DBMSs provide data security by controlling access privileges of users to data. While these *discretionary* access controls (DAC) provide adequate mechanisms for preventing unauthorized disclosure of information to most "honest" users, malicious users who are determined to get access to the data must be restricted using other means. Studies have shown that the mechanism provided by database systems often can be bypassed owing to flaws in the systems which host the DBMS [36, 74].

The term *covert channel*, in general, is used to refer to any communication channel that can be exploited by a process to transfer information in a manner that violates a system's security policy. Since current covert channel criteria in the CTCPEC [6], TCSEC [11], TDI[1], and ITSEC[2] are essential elements in determining between B1, B2, B3, and A1 trust levels (and their international equivalents), an understanding of this concept is critical [36]. The TCSEC [11] and CTCPEC [6] place covert channel analysis requirements on trusted systems: "...thorough search for covert *storage* channels and make a determination ... of the maximum bandwidth of each identified channel ... be able to audit the identified events that may be used in the exploitation of covert storage channels." It is important that the covert channel problem is solved in a way so as not to limit the usefulness of a system or to adversely affect system performance. Because there is no multilevel DBMS that is rated at the B2 trust level (or higher) available on the market today, there is only very little experience in building and evaluating trusted multilevel database systems. Unfortunately, all of the earlier analytical and clarification research directed at covert

[1] Trusted Database management system Interpretation of the TCSEC.
[2] Information Technology Security Evaluation Criteria, harmonized criteria of France – Germany – The Netherlands – The United Kingdom.

channels still leaves us with no single agreed-upon exposition for operating system covert channels [36].

Since the covert channel problem encroaches on other areas in computer science such as communication theory, networks, and databases, these relationships will be explored. The next section reviews security concepts and discusses covert channels. Section 3 reviews covert channels as communication channels and defines noisy and noiseless channels. Section 4 reviews previous research activities related to covert channels. It discusses methods for analyzing and detecting covert channels, methods for estimating a covert channel bandwidth, and methods for handling covert channels. Section 5 discusses covert channels in networks and section 6 reviews multilevel database covert channels. Section 7 concludes.

2 BASIC CONCEPTS

In this section some relevant multilevel security concepts and covert channels are discussed.

2.1 Multilevel Security

The set of rules by which a secure DBMS controls access to data is known as the system's *security policy*. An *object* is a passive entity such as a data-file, a record, or a field within a record. A *subject* is an active *process* (a program in use) that can request *access* to *objects* (i.e., information resources). Among types of objects there exists a finite set of distinct ways in which objects of a particular type can be manipulated; each type of manipulation is called an access. Every access to an object is made on behalf of some subject. The access relations or the access rules are best abstracted by means of an access matrix. Before granting access to a subject, a check is performed using the access control matrix (Figure 2.1). The matrix contains one *label* type along each axis and contains the authorized modes of access in each matrix element. A label is a security attribute that is associated with an object or a subject. It is used to describe any access mediation information. For example, the complete subject/object matrix contains all current subject labels along one axis and all current object labels along the second axis. Every object is assigned a classification, and every subject a clearance. Classifications and clearances are

collectively referred to as *access classes* (or *levels*). An access class consists of two components: a hierarchical component (usually TOP-SECRET, SECRET, CONFIDENTIAL, and UNCLASSIFIED) together with a set of non-hierarchical or *need-to-know* categories (e.g., NATO, Nuclear, Army, etc.). All data is labeled hierarchically and categorically, so a particular data item might carry the security label [TOP-SECRET, NATO, Nuclear]. The set of access classes ([sensitivity level, category set] pairs) is partially ordered and forms what is known as a lattice [9, 47]. By partially ordered it is meant that given any two security classifications, either one is greater than or equal to the other, or the two are said to be non-comparable. For example, TOP-SECRET is greater than SECRET, but [TOP-SECRET, NATO] and [TOP-SECRET, Army] are non-comparable, because neither is greater than the other: a [TOP-SECRET, NATO] user cannot obtain access to [TOP-SECRET, Army] information, and vice-versa. The partial order relation of security classifications is called a *dominance* relation. Given two access classes c_1 and c_2, $c_1 \geq c_2$ (c_1 dominates c_2) if and only if the hierarchical component of c_1 is greater than or equal to that of c_2 and the categories in c_1 include those of c_2. For example, [TOP-SECRET, NATO, Army] dominates [TOP-SECRET, Army]. There are two security (access) class bindings: *fixed* or *static binding*, in which the security class of an object is constant, and *variable* or *dynamic binding*, where the security class of an object varies according to its contents [3, 7, 9]. Users and their processes are usually bound statically [3, 7, 9].

objects

subjects	obj_1	obj_2	obj_3	obj_4	obj_5
sub_1	R	W	R/W		R
sub_2	W	R		R	R/W
sub_3	W		R/W		W
sub_4	R		W		R/W

Figure 2.1: A Portion of an Access Control Matrix, R=Read and W=Write.

The idea of *mandatory* (or *non-discretionary*) access controls (MAC) together with a mechanism called the trusted computing base (TCB) for the enforcement of these controls are defined. As mentioned in [29], MAC is based on the Bell-LaPadula model [3] which imposes the following restrictions on all data access:

1. *The simple security* property: a subject is allowed a read access to an object only if the subject's access class dominates the object's.
2. The *-*property* (pronounced "the star property"): a subject is allowed a write access to an object only if the object's access class dominates the subject's.

These properties are generally regarded as necessary in a security policy but may not be regarded as sufficient. For example, some security policies do not permit reading or writing unless the hierarchical components of the access classes involved are identical in addition to non-hierarchical domination. The sufficient conditions will include the usual discretionary access controls of commercial DBMS. To meet the CTCPEC [6] and TCSEC [11] requirements, it must be possible to demonstrate that a given system is secure [29]. To this end, the secure DBMS designers follow the concept of implementing a TCB. The TCB is the set of security critical components[3] and security relevant components that enforce a system's security policy. It attaches labels to users, processes, or objects, enforces access controls and must be continuously protected and tamper proof.

When building a TCB or a secure system, it is important to take into account that security is NOT an add-on feature. For a secure computer system, there must be a security policy and a TCB to enforce that policy. A *threat/risk assessment* is conducted to determine what degree of confidence one can have that a product's security policy is correctly implemented. In this respect, there are four functional criteria that must be addressed [6]. These criteria are *confidentiality*, the ability to prevent disclosure of information to unauthorized individuals; *integrity*, the ability to prevent modification by unauthorized individuals; *availability*, the ability of a product to withstand a denial of service attack or failure; and *accountability*, the ability to monitor and hold individuals responsible for their actions. A further requirement is to ensure that the four basic criteria are complete and cohesive, that is *assurance*. Assurance is the degree of confidence one

[3] Those components which directly contribute to the provision of one or more security services.

can have that a product's security policy is correctly implemented. Assurance is gained through the use of rigorous and comprehensive development and implementation strategies, and subsequent thorough evaluation and testing procedures that are adhered to throughout the system lifecycle. The system lifecycle must include disposal when production is terminated.

As seen, effective criteria (TCSEC, CTCPEC, and others) and the Bell-LaPadula model have been developed for the analysis of computer systems to ensure that two processes at different security levels cannot directly communicate information in violation of security policies. However, despite the guidelines in these criteria and the Bell-LaPadula model, many systems suffer from processes that communicate by means of covert channels.

2.2 Covert Channels

As mentioned in [33], overt channels use a system's protected data objects to transfer information. That is, one subject writes into a data object and another subject reads from that object. Channels, such as buffers, files, and I/O devices, are overt because the entity used to hold the information is a data object; that is, it is an object that is normally viewed as a data container. Covert channels, in contrast, use entities not normally viewed as data objects to transfer information from one subject to another. File locks, memory size, and passing of time are examples of these entities or nondata objects. Covert channels can be mainly classified as being either *storage* channels or *timing* channels. Storage channels are channels which allow the coded transfer of unauthorized information through a nondata object legally (illegally means in violation of the security policy) written by one process (usually a high process) and legally read by another (usually a low process)[4]. Covert timing channels are those channels that allow one process to signal unauthorized information to another process by modulating the use of system resources so that changes in response times can be observed by the other process. Several definitions for covert channels have been proposed. As mentioned in [36], covert channels are described with adjectives such as storage, timing, static, time-decaying, leakage, bypass, backflow,

[4] The high process dominates the low process. Usually, the reason that these nondata objects can "legally" be written by a high process and "legally" read by a low process is that these objects are outside the scope of the security policy (so all accesses are legal).

information, signaling, inference, and aggregation. There are definitions indicating that covert channels exist only with the existence of a *Trojan Horse*[5]. Other definitions indicate otherwise. Since there is no multilevel DBMS that is rated at the B2 trust level (or higher) available on the market today there are no established guidelines or expositions of exactly what constitutes a database covert channel [36].

In general, a grey area exists between storage and timing covert channels depending on how long the value of the underlying storage cell may be maintained. As stated in [43], a channel whose underlying storage cell holds its value for the length of a boot load must be viewed as a storage channel. On the other hand, if the value of the storage cell is driven by mechanisms related to the overall system speed (process switching time, page replacement time, etc.), then the channel is a timing channel. A difference between a covert storage channel and a covert timing channel is that the latter is memoryless, where the former is not [18]. One way that may distinguish between a storage and a timing channel would be to determine what would happen if the operation of the system were completely halted, the channel is a timing channel if the information would be lost [24, 42]. If the covert data is still obtainable, the channel is a storage channel. There is, of course, an aspect of timing in both types of channels, but with covert timing channels, time is an essential element. That is, with timing channels, information transmitted from the sender must be sensed by the receiver immediately, or it will be lost.

The tasks of identifying and handling covert timing channels (i.e., elimination, bandwidth reduction, or audit) in a secure system are more difficult than for covert storage channels. One reason is that, in addition to exploiting normal system activity, covert timing channels can also involve the direct exploitation of system hardware [18]. Another reason is due to the lack of tools/methodologies that identify covert timing channels. Despite these problems, one can limit or eliminate some types of covert timing channels.

---

[5] A Trojan Horse is a program containing an apparent or actual useful function that contains an additional (hidden) function which allows unauthorized collection, falsification, or destruction of data. As an example; a sort routine may have hidden in it a Trojan Horse such that whenever a user invokes the sort routine in addition to accessing the user's file to be sorted, it accesses other files of the user and copies them into files belonging to some unauthorized user [14].

Additional Notes:

1. Students are strongly advised to choose 9 ch of their Arts electives from the following partial list:

HIST 1305	HIST 1400	HIST 1100	HIST 1300	HIST 2815
HIST 2825	HIST 2910	HIST 2925	HIST 1005	HIST 1245
HIST 1006	HIST 1315	HIST 2015	HIST 2025	HIST 1246
ENGL 1000	ENGL 1145	ENGL 1146	ENGL 2703	ENGL 2263
PHIL 1000	PHIL 2103	PHIL 2104	PHIL 2073	POLS 1000
SOCI 1000	PSYC 2103	PSYC 2203	PSYC 2643	GER 2031
GER 2042	RUSS 2043	RUSS 2053	CLAS 1003	CLAS 1013
CLAS 3003	CLAS 3033			

   Good communication skills are of great importance to all university graduates and are in high demand by employers. All students are strongly advised to take additional courses in the Humanities, particularly those that require the writing of essays.

2. In addition to the normal Arts requirement, students entering Computer Science are urged to consider very seriously the study of French while they are at university. In a bilingual province and country, knowledge of both official languages is an important asset in most professional occupations. The French Department offers language courses suitable for all students regardless of the level of proficiency they have previously been able to acquire. There are often several course offerings from other departments available in French.

3. Computer Science freshmen are advised to register for only five courses per term (only one Computer Science course per term). If you have good marks from high school and maintain a 'B' average in first term, you may take an additional approved computer science course in second term (CS 2513 or CS 2503). This will lead to more flexibility in the upper years of your program and may help with either summer employment opportunities or admission to the Co-Op Program.

4. Students are advised that those with assessment gpa of 3.5 or higher may be eligible for scholarships. Please apply for these as appropriate if you qualify or come close to qualifying. Scholarships are awarded only to those who apply.

5. Planning for the CS 4997 will begin late in the term before the one in which the work is actually done. For this reason most students will register for it as a second term course in year 4. Prof. T. Austin is the Coordinator for CS 4997 and students are advised to watch the Computer Science bulletin board between rooms E113 and E114, Gillin Hall, for announcements and notices of required meetings which will normally be scheduled for 2:30 P.M. on Tuesdays and Thursdays.

6. Students with a B average are eligible to take 5000 level courses. The description for these courses is given on page F.33-34 of the 1996/97 Undergraduate Calendar.

2.2.1 Covert Channel Examples

Example 1

To give an example of a storage covert channel, consider a system that does not permit the same name to be used for multiple files, regardless of the security level. If the system returns an error message informing a process that a filename cannot be registered because another file exists with the same filename, this information route can be exploited as a covert channel [24]. One process could register a filename whenever it wants to transmit a binary 1 and the other process can attempt to register the same filename. The second process will either be able to or not. When it cannot, the first process has transmitted a 1; when it can, the process has transmitted a 0.

Example 2

When a process instruction is being executed in the Central Processing Unit (CPU) the process is said to be *running*. To prevent any one process from monopolizing the CPU, the operating system shares access to the CPU by allotting each process a time period called a *quantum* or *time slice*. When the time allotted to a process is used up the hardware causes an *interrupt* to occur which suspends execution of the process and returns control to the operating system. The time that elapses between time slices allotted to a given process is called the "interquantum-time channel" [28]. This channel can be exploited as a covert timing channel. One example of the exploitation of such a channel is to consider a process that uses the time between two successive CPU quanta to transmit information to another process. The sender and the receiver are assumed to have access to a clock. To send information, the sender and the receiver agree on set times for sending the information. The transmission strategy is for the sender to execute at time t_i if the ith bit is 1, and to block itself if the ith bit is 0. The receiver tries to execute at time t_i, if he fails, then the sender is executing at that time. If only the sender and the receiver are in the system, the receiver can decode each transmitted bit correctly with probability one.

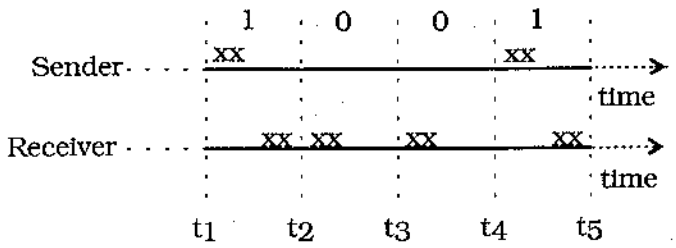


Figure 2.2: CPU interquantum channel, receiver detects whether sender runs at time t_1, t_2, ..., t_n and receives 0s and 1s.

3 Threat/Risk Assessment

It is natural that risk accompanies any security issue. In general there are two ways to approach risk assessment. One is a quantitative approach in which a quantity termed the Annual Loss Estimate (ALE) is produced. It is a single figure that indicates the expected loss due to the cost of the various threats to a system, multiplied by the expected number of instances of the threats in a year. It must be realized that the ALE is derived from data and probabilities which frequently do not have an empirical basis. The key to the value of a risk assessment based on this methodology is ensuring that the data and probabilities used in the assessment are as near to real life as is possible.

The second approach is qualitative where experience or certain known requirements and occurrences are relied upon to assess the risk. The success of this approach in assessing risk is based directly on the capability to identify threats and vulnerabilities correctly. From the knowledge of threat and vulnerabilities their effects can be determined, and defensive measures can be identified.

3.1 Risk Management

The aim of risk management is to make informed decisions with full recognition and acceptance of *residual risk* (a risk that can not be avoided or eliminated). Risk management is a risk of assurance rather than a risk of security. It is useful to understand risk management when addressing covert channels. Therefore, a simple risk model will be used [24]. The model (Figure 3.1) is compromised of three elements; the assets, the threat agent, and the safeguards. An asset is an entity which has a value to an organization (in the above non-multiple filenames channel, the asset is the classified information). The threat agent is an entity that desires to and is able to trigger an event that compromises an

asset's security. In the above covert channel examples, the threat agent would be the sending process (or any malicious process or software, e.g. Trojan horse). The safeguards are mechanisms that protect the assets from threat agents by creating other mechanisms that countermeasure the threat activities. Safeguards are prone to vulnerabilities perceived by threat agents despite that they are implemented according to policies and standards. Through different perceived asset values, the threat agent is capable to assess the assets value. If it is possible that the safeguards can be circumvented and an access can be gained to the assets, the threat agent will initiate a threat event or an attack against the safeguards. If the attack was unsuccessful, then the safeguards have succeeded in protecting the assets, otherwise a security violation and a complete or partial loss of the assets will occur. In such case, the safeguards are not capable of protecting the assets.

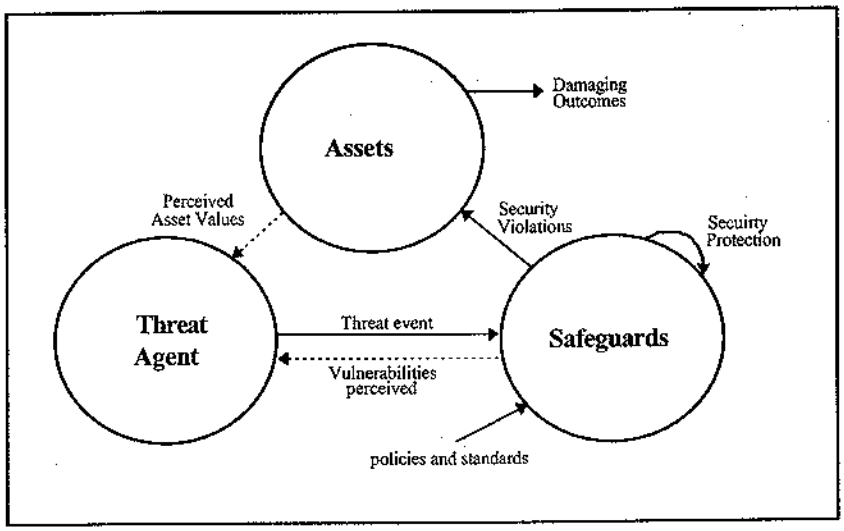


Figure 3.1: Threat Model.

4 COVERT CHANNELS AS COMMUNICATION CHANNELS

In this section the basic notion of how to measure the rate at which information can be transmitted over a communication channel is discussed. Then it is shown how this concept can be applied to a covert channel in a computer system.

As shown in Figure 4.1, a communication channel has two ends. One end is the input, where a process X is generating a message in the form of a sequence of symbols, and

entering them into the channel. At the other end is the output, where another process Y is reading the symbols emerging from that end of the channel. The *capacity* of a channel is the maximum rate at which information can be transmitted from input to output in bits per second (BPS) [52].

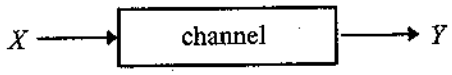


Figure 4.1: Information Channel.

The channel is said to be *noiseless* if the symbols emerging are identical to those entered. For a noiseless channel, the rate at which information is transmitted to the output is the same as the rate at which information is generated at the input end. The information rate is measured in bits per second, which can be further subdivided as the product of bits per symbol and symbols per second. The calculation of capacity reduces to the question of how fast the channel works if each symbol is represented as a fixed number of bits [52].

It is not hard to view the non-multiple filenames covert channel example given above as a communication channel, where the high (sending) process is the input source X, and the low (receiving) process is the output destination Y. The channel is noiseless, with one bit per symbol. Its capacity in bits per second depends on how long it takes to go through the complete cycle for one bit. If it takes, say, five milliseconds, then the capacity is two hundred bits per second.

4.1 Noisy Channels

Anything that might make a channel's output different from its input is noise, that is, anything that introduces transmission errors [52]. Noise may be caused by spurious user processes that run concurrently in the system and perform authorized writes to each shared variable. A channel is noiseless either when there is no spurious process in the system or when all spurious processes between the sender and the receiver processes do not, or cannot, alter the variable. In the non-multiple filenames channel example the correctness of received bits might be disrupted. For example, a spurious process that has the same security level as the cooperating process might create a file under the same name at the time the file is deleted by the cooperating process. A failure will be encountered by

the receiving process even though the file is not created by the cooperating process. Hence the receiving process codes an extra "0." Noisy channels still carry information at some rate less than the bit rate at which symbols are sent. But, by adding more noise, the channel capacity goes down, and there is nothing that malicious programs can do to transmit information faster than the capacity allows.

Hence, once aware of a potential covert channel, system designers may introduce noise through modifications in the scheduling algorithm or other appropriate places. Their options are limited by the desire not to penalize legitimate users. It is difficult, in general, to introduce covert channel countermeasures without degrading the overall performance of the system, hence, tradeoffs must be made.

5 RELATED RESEARCH ACTIVITIES

In addressing the threat of covert channels, two major challenges have been identified. The first challenge is in developing techniques to detect covert channels in a comprehensive systematic manner [35]. A number of covert channel analysis techniques have been proposed. Usually these techniques base their analysis either on code inspection or inspection of the high-level specification. The second and more difficult challenge is, of course, to remove the channels or at least lower their bandwidths without unacceptably degrading the performance of the system [35].

5.1 Covert Channel Detection

Until the mid-eighties, recognizing and dealing with storage and timing channels when performing a security analysis of a computer system appeared to be a very difficult and often ignored task. Methods for discovering and dealing with covert channels had for the most part been *ad hoc*, and not restricted to a particular specification language [34]. At the present time, more methodical approaches to recognize and deal with covert channels are presented. All of the widely used methods for covert channel detection are based on the identification of illegal information flows in top-level specifications or source code.

5.1.1 Syntactic Information-Flow Analysis

An approach on a syntactic analysis, suggested originally by Denning [7], was developed. Denning's approach was intended for use on application programs rather than

the operating system. There are a number of variations of this approach. Denning identified two different flows: explicit flows, such as the flow from A to B (denoted A → B) in the assignment statement "B := A," and implicit flows, which occur in conditionals such as "IF A=X THEN B:=Y." The approach assigns sensitivity levels to program variables. Furthermore, one defines a flow policy: "if information flows from variable A to variable B, the security level of B must dominate that of A." When a flow policy is applied to program code, it helps generate flow formulas. For example, for the statement "X := Y," its flow formula could be: security_level(X) ≥ security_level(Y).

All flow formulas must be proven correct; usually a theorem prover is employed for this purpose. The flow of a statement can lead to a covert channel if its particular flow formula cannot be proven correct. In such a case, further analysis is necessary; one must perform semantic analysis to determine whether the unproven flow leads to a real and not a potential covert channel. Some of the potential channels do not have scenarios of real use. These potential channels are created by the identification method. For example, when Honeywell's SCOMP (Secure COMmunications Processor) was analyzed, there were sixty-eight unproven formulas. These formulas represented potential covert channels, but only fourteen of them turned out to be real covert channels [18, 52]. A general reason that a potential channel might not necessarily be a covert channel is that, some flow conditions may never become true at run time. Hence, illegal flow that could create a covert channel may never be enabled. To determine whether a potential covert channel is a real covert channel, one must find a real-time scenario enabling an illegal flow.

The syntactic approach has some attractions to it; it can be automated, it can be applied to formal specifications and source codes, and it detects all flows that lead to covert channels. On the other hand, the syntactic approach is not precise; it is vulnerable to the discovery of false illegal flows. Thus, additional efforts are required to eliminate such flows by manual semantic analysis. The analysis of design specifications leads to the discovery of false information flows, namely of flows that appear in the formal specifications but do not appear in code [67].

The syntactic approach on information flow analysis discussed above focused on whether or not actual code satisfies the information flow policies of a security model

rather than identifying covert channels. The approach assumed that each variable (or object) is either explicitly or implicitly labeled with a specific security level (or access class). This means that, in practice, the syntactic approach provides only a subset of the necessary conditions for covert channel identification. Covert channels use entities not normally viewed as data objects [33]. Consequently, these entities may not be part of the interpretation of a given security model. Instead, their security level may vary dynamically depending on flows between labeled objects. In addition, a security model interpretation may exclude some resources as implementation detail. Hence, the flow analysis needed for covert channel identification must include variables that are not part of a security model interpretation [67].

5.1.2 Non-interference Analysis

Non-interference is based on viewing the computer system as an abstract machine; for example a user process can view an operating system as a black box that provides certain services when requested. Usually a request results in a response to the requesting process, either a data value, an error message, or a positive acknowledgment. A process' requests are the abstract machine's inputs; the responses are its outputs; and the content of the internal variables and data structures it maintains at any given time is known as its current state. When analyzing a system, these variables and data structures are available through either the source code or a more abstract formal/descriptive specification.

Non-interference was defined by Goguen and Meseguer [20]. They defined the concept of non-interference between two user processes – assuming an initial or start-up state for the machine, a user process is non-interfering with another when the output observed by the second user process is unchanged if all inputs from the first user process, ever since the initial state, were removed as though they had never been entered. Goguen and Meseguer reasoned that if inputs from one user process could not affect the outputs of another, then no information could be transmitted from the first to the second.

Let X and Y be two user processes of an abstract machine. Further, let w be a sequence of inputs to the machine, ending with an input from Y. Assuming the machine was in its initial state when w was entered, let the output of that last input be $Y(w)$. To express non-interference, assume that w/X is the subsequence that remains of w when all

X inputs are deleted from it. Then X is non-interfering with Y if, for all possible input sequences w ending with a Y input, $Y(w) = Y(w/X)$.

When thinking about covert channels, the traditional way is to see each individual X input having an effect on the next Y output. Non-interference looks at it in another way; Y might enter an input to request an output at any time. Suppose that Y entered an input every time X did. Ignoring other inputs, the overall input sequence would be like:

$$X_1 Y_1 X_2 Y_2 \ldots X_n Y_n.$$

The non-interference definition applies to all the initial sequences of the whole sequence as well as to the whole sequence itself, namely:

$$X_1 Y_1$$
$$X_1 Y_1 X_2 Y_2$$
$$X_1 Y_1 X_2 Y_2 \ldots X_n Y_n.$$

Suppose that each input is reported as a Y output after some delay, then a covert channel arises and it is just as bad as if the X input comes out immediately in the next Y output. Therefore, the non-interference analysis requires that every Y output is unaffected by all previous X inputs. Hence, it is necessary to analyze all previous X inputs. However this analysis is unnecessary because the current state has all the information needed to determine the next Y output. Non-interference can be expressed in terms of the current state instead of the whole past input history: if X is non-interfering with Y, an X input should have no effect on the Y outputs in the states before and after it. Non-interference requires that an X input should have no effect on any subsequent Y output.

Two states are Y-equivalent if (1) they have the same output in response to the same Y input, and (2) the corresponding next states after any input are also Y-equivalent. Goguen and Meseguer [20] proved a theorem, called the Unwinding Theorem. This theorem says X is non-interfering with Y if and only if each X input takes each state to a Y-equivalent state. Unwinding leads to practical ways of checking non-interference. A Multilevel security policy requires that each process X at a given security level should interfere only with a process Y of an equal or higher security level. To apply this requirement in practice, the abstract machine state-variables must be defined, and the Y-equivalent states must be identified.

One way to identify Y-equivalent states in a multilevel system is to label state-variables with security levels. If Y is cleared for a security level , say *sl*, then the two states are Y-equivalent if they have the same values in those state-variables having a security level dominated by *sl*. There are three properties that must be proved in a non-interference analysis: (a) The state-variable level assignment must have the property that the effect of any input turns equivalent states into equivalent states, (b) Any return values reported back to Y depend only on variables visible to Y (variables at or below Y's level), and (c) an input from a higher level process X cannot affect the variables visible to user process Y.

Non-interference analysis can be applied to formal system specifications as well as source code. However, it has a disadvantage that it may be impractical to be applied to a significant size system (containing large number of variables) for that automated tools are unavailable to date [18].

5.1.3 The Shared Resource Matrix (SRM) Approach

The SRM method was proposed by Kemmerer [33]. The method provides a systematic approach for conducting a covert storage channel search, and for identifying the ending state of that search. This is crucial for managing a covert channel analysis over the lifecycle of a product (as products are revised, so too must the security analysis activities if the system is to maintain its assurance rating). Covert channel detection is performed into two steps [33]: First, all the shared resources (e.g., system variables) that can be referenced or modified by more than one subject through system calls are defined. The second step is that, each resource is carefully examined to determine whether it can be used to transfer information from one subject to another covertly. The method assumes that subjects of a system are processes. The method further refines the view of each shared resource by indicating its attributes, because two processes may view different attributes of the same shared resource. For example, the first process may be able to determine only whether a shared file is locked, while the second process may only view the size of the file. One sets up a matrix whose rows correspond to the shared resource attributes and whose columns correspond to the system primitives, some examples of a system primitive are Write-File, Read-File, Lock-File, and Unlock-File.

After determining the row and column headings one must determine which primitive references which attribute. This is done by a careful review of the description of each primitive, whether it is an English requirement, formal specification, or implementation code. The matrix generation is completed when each element of the matrix has been considered and marked, indicating whether a modification or reference could occur. Figure 5.1 shows a resource matrix that was filled in from an English system description [33].

Resource Attribute \ Primitive		Write File	Read File	Lock File	Unlock File	Open File	Close File	File Locked	File Opened
Process	ID								
	Access Rights			R		R		R	R
	Buffer	R	M						
File	ID								
	Security Classes			R		R		R	R
	Locked By	R		M	R				
	Locked	R		R,M	R,M	R		R	
	In-use Set		R	R		R,M	R,M		R
	Value	M	R						
Current Process		R	R	R	R	R	R		

Figure 5.1: Resource Matrix; R=Reference, and M=Modify.

Indirect references are also added to the matrix. A resource attribute is referenced indirectly by a system primitive operation if the operation can reference a second attribute which may contain information from the first, by virtue of some other operation that may reference the second and modify the first. Indirect references may contribute to further indirect references, so an iterative process called *transitive closure* (not the standard mathematical transitive closure, since it relates to the modify operator as well as the reference operator) is used to ensure that all indirect references have been included. For instance, suppose an operation Login references the password file and modifies the Active-User attribute. Furthermore, suppose a second operation references the Active-User attribute. The shared resource matrix for these two operations would indicate a reference to Active-User, but no reference to the password file in the column that corresponds to the second operation. However, it may be the case that the Active-User attribute is modified in a manner which compromises a user's password [33]. Thus, it is necessary to indicate this indirect reference in the matrix. Then, when analyzing the matrix for possible channels, one must ensure that the modification to Active-User does not reveal information about the user's password.

The transitive closure of the matrix is generated by looking at each entry that contains an R (Reference). If there is an M (Modify) in the row in which this entry appears, then it is necessary to check the column that contains the M to see if it references any attributes that are not referenced by the original primitive. That is, if the column that contains the M has an R in any row in which there is no R in the corresponding row of the original column, then an R must be added to that row in the original column. For instance, consider the column for Write-File in Figure 5.1. There is an R in the locked row of this column, and the attribute is modified by the Lock-File primitive. Therefore, it is necessary to see which attributes were referenced to make this modification [33]. The attributes Access-rights, Security-classes, Locked, In-use set, and Current process are referenced. Access rights, Security classes, and the In-use set are not directly referenced by the Write-File primitive, so they must be added to that column. This process is repeated until no new entries can be added to the matrix. Figure 5.2 depicts the transitive closure matrix for the resource matrix of Figure 5.1 [33]. The matrix is then examined for rows containing

both an R, direct or indirect, and an M entry; these represent resource attributes that support communication channels from the process using one operation to modify the attribute and a second process using another (or the same) operation to read the attribute. If the sensitivity level of the first process is not less than or equal to that of the second process, there is a potential covert channel.

The SRM has the advantage that it can be applied to an informal specification of a system (as well as a formal one), although the results are only as good as the information supplied. It can also be applied to source code. Because the current process identifier is considered a shared resource, covert timing channels related to process scheduling will not be differentiated by the method from storage channels [18]. Furthermore, the SRM method does not require that security levels be assigned to internal system variables represented in the matrix (as did the syntactic information-flow analysis), and thus it eliminates a major source of false illegal flows. Although the SRM approach is applicable to source code, tools to automate the construction of the shared resource matrix, which is time-consuming, do not exist to date [18]. The method does not consider the conditions under which a flow occurs. Quite often, a system call will access different variables depending on which execution path is taken through the call. In simple cases, this can be addressed by viewing each execution path through the call as a separate call. This increases the number of columns in the matrix. Another disadvantage of the SRM is that it does not provide a criteria for distinguishing between a potential channel and a real channel.

5.1.4 A Formal Method For Identification of Covert Channels in Source Code

This method was proposed by Tsai *et al* [69]. The method identifies covert storage channels based on three steps: (1) the analysis of programming language semantics, code, and data structures used within a system's kernel to discover variable alterability/visibility, (2) resolving aliasing of kernel variables to determine their indirect alterability, and (3) information flow analysis to determine indirect visibility of kernel variables. Within a primitive, a function f_1 depends on another function f_2 if f_1 calls f_2.

The set of all functions that appear in the function call dependency (FCD) chain of a primitive is called the FCD set of the primitive. Tsai and Gligor [68] define a *visible*

variable at the kernel interface to be a variable such that changes in its value can be detected by a user process that invokes a kernel primitive. A variable returned from a function to another function within a primitive operation may not become visible unless that variable can be made visible to the user process by the caller function. For this purpose the analysis of the FCD set for each primitive is important for the determination of the visible variables. The flow of information from a variable v_1 to another variable v_2 is denoted $v_1 \rightarrow v_2$.

Resource Attribute \ Primitive		Write File	Read File	Lock File	Unlock File	Open File	Close File	File Locked	File Opened
Process	ID								
	Access Rights	**R**	**R**	R	**R**	R	**R**	R	R
	Buffer	R	**R**,M						
File	ID								
	Security Classes	**R**	**R**	R	**R**	R	**R**	R	R
	Locked By	R	**R**	**R**,M	R	**R**	**R**	**R**	**R**
	Locked	R	**R**	R,M	R,M	R	**R**	**R**	**R**
	In-use Set	**R**	R	R	**R**	R,M	R,M	**R**	R
	Value	**R**,M	R						
Current Process		R	R	R	R	R	R		

Figure 5.2: Transitive closure matrix; R=Direct Reference, **R**=Indirect Reference, and M=Modify.

The dependency notion between variables defines the indirect visibility of variables. The visibility of variable v_1 depends on the visibility of variable v_2 if, whenever v_2 is visible, then v_1 is visible. The indirect visibility of variables is discovered by information flow analysis. From language semantics, the information flow is determined by analyzing statements of code.

A variable is *alterable* if the value of the variable can be changed by any function of the FCD set of a primitive. The alterability of a variable can take place in each of the following actions; increment/decrement, assignment, insertion into and deletion from a list, and allocation and release of an entry in a table. The information flow in an assignment implies that the variable at the left-hand side is alterable. Note that the alteration of a data structure may require more than one language statement (e.g. table entry allocation/release).

An assignment such as $v_2 = v_1$ implies bi-directional information flow between v_2 and v_1 whenever v_2 is a pointer. v_2 is called an *alias* of v_1 because v_2 represents the same structure as v_1. To resolve variable aliasing means to unify the names of the variable. To discover the name of each structure and to distinguish shared variables from local variables, all aliases must be resolved. Aliases are ignored when they are local variables (for they are not visible outside the function). Tsai and Gligor [68] replace all aliases by the original variable name. Thus, different variables representing the same object cannot exist in the code of a primitive operation.

After the information flows and variable aliasing between the functions of each FCD set have been defined, the visibility/alterability of each variable in a primitive operation is discovered. Then, the shared alterable/visible variables for each primitive are searched for. The set of primitives associated with a variable that is both visible and alterable is selected for covert channel identification. Others, namely primitives that are associated with variables that are only visible or only alterable, need not be included in the identification process. A potential covert channel needs a sender process that could change (alter) a shared variable, and a receiver process that could detect the change in the variable. Thus, primitives associated with visibility only and alterability only cannot be used for covert transmission of information.

Whenever the security level of the sender process is higher than that of the receiver process in a multilevel security environment, a potential covert channel exists. By applying the mandatory access checks of a security model interpretation, all potential covert channels provided by the shared variables are discovered. The method was applied to the Secure Xenix Kernel code; the following results were obtained [18]: — fewer than 400 kernel variables are visible or alterable; fewer than 100 variables are both visible and alterable; and 24 variables created covert channels [67].

There are two advantages with this method; it leads to the discovery of all storage channels in kernel implementations and it can be automated. A disadvantage of the method is that its manual application to real TCBs requires extensive use of highly skilled personnel. For example, its application to the Secure Xenix system required two programmer-years of effort [18]. Thus, using the method in real systems requires extensive use of automated tools. Although it is applicable to any implementation language, the method's automation requires that different parsers be built for different languages. Another disadvantage of the method is that the channels identified become more difficult to remove than had they been caught during a system specifications development. One can re-design a system specifications to remove the channel without incurring a significant cost. If the channel is not caught during the system design and is caught after it is implemented into a source code, it becomes more expensive and more difficult to remove.

5.2 Estimation of Covert Channel Bandwidths

The TCSEC [11] requirements for determining covert channel bandwidths state that the system developer has to measure/estimate the maximum bandwidth of each identified covert channel (i.e., capacity). The measurement or estimation of the maximum attainable bandwidth must assume that the covert channels are noiseless, that no processes – other than the sender or receiver – are present in the system when measurements are performed. Bandwidth computation/estimation is necessary to determine the appropriate method for covert channel handling (discussed in Section 5 below). If the channel cannot be removed, then its bandwidth has to be lowered to a predetermined value. Introducing unnecessarily

large delay values causes unnecessary performance degradation for the entire system. Hence, accurate bandwidths are important for appropriate covert channel handling [68].

5.2.1 Calculating Covert Channels Bandwidths Based on Information Theory

Based on Shannon's information theory [59] for calculating the capacity of an information channel, Millen [53] presented a method for calculating the bandwidths of covert channels.

The maximum information rate of a channel is known from information theory to be its capacity, defined as follows:

$$C = \lim_{t \to \infty} (\log_2 N(t)) / t \tag{1}$$

where $N(t)$ is the number of possible symbol sequences of total time t. If there are two distinguishable symbols of lengths d_1 and d_2 (the sender would take d_1 seconds to transmit, say a 0, and d_2 seconds to transmit a 1), then

$$N(t) = N(t - d_1) + N(t - d_2) \tag{2}$$

According to Shannon's information theory [59, p. 37], $N(t)$ is the asymptote for large t to Ax^t, where A is a constant and x is the largest real solution of the equation

$$x^{-t_1} + x^{-t_2} + \ldots + x^{-t_n} = 1 \tag{3}$$

and therefore, by substituting

$$N(t) = Ax^t \tag{4}$$

in equation (2), we get

$$Ax^t = Ax^{t-d_1} + Ax^{t-d_2} \tag{5}$$

which can be restated as

$$Ax^t (1 - x^{-d_1} - x^{-d_2}) = 0 \tag{6}$$

Because Ax^t approximates $N(t)$ asymptotically, a solution of equation (6) will give the asymptotic solution of equation (2):

$$1 - x^{-d_1} - x^{-d_2} = 0 \tag{7}$$

When specific values for d_1 and d_2 are substituted, solutions for x can be found. The largest real solution is used to calculate the channel capacity:

$$C = \lim_{t \to \infty} (\log_2 Ax^t) / t = \log_2 x.$$

5.2.2 Informal Method for Estimating Covert Channel Bandwidths

Tsai and Gligor [68] introduced a formula for computing the maximum attainable bandwidth. The time for the sender process to alter a variable, and the time for the receiver process to detect or view that variable change, are denoted by T_S and T_R, respectively. The formula is:

$$\mathrm{B} = b * (T_R + T_S + 2T_{CS})^{-1},$$

where b (in bits) is the information encoded through the variable, that is, the encoding factor (which was assumed to be 1 in most practical cases). $T_R = \sum_{i=1}^{n} \frac{T_R(i) + T_{env}(i)}{n}$, where n is the number of total possible transitions. $T_R(i)$ is the time necessary to read a 0 or a 1, and T_{env} is the time to set up the environment to read a 0 or a 1. In deriving these formulas it was assumed that the environment setup for both variable setting and reading is done by the receiving process. Further, it was assumed that the setting of 0s and 1s takes the same amount of time, and that all transmissions contain an equal distribution of 0s and 1s (uniformly distributed). $T_S = \sum_{i=1}^{n} \frac{T_S(i)}{n}$, where $T_S(i)$ is the time necessary to set a 0 or 1. T_{CS} denoted the *context switching* time; when allocating the CPU to another process, the kernel performs a context switch from the current process to a new process. Each process may involve a context switch.

5.2.3 The Two Methods Compared

Millen's method [53] is more precise than that presented by Tsai and Gligor [68], because during its use one is required to define a realistic scenario of covert channel use. That is, Millen's method does not require that 0s and 1s are uniformly distributed. Also, it differentiates between the time it takes the sending process to transmit a 0 and the time it takes to transmit a 1.

Experience with using the two methods for Secure Xenix shows that in cases where times to transmit a 0 or a 1 are close, the methods yield results that differ by at most 20% [18].

5.3 Covert Channel Handling

From a security point of view, covert channels with low bandwidths represent a lower threat than those with high bandwidths. However, for many types of covert channels, reducing the bandwidths below a certain rate (which depends on the specific channel mechanism and the system architecture) also has the effect of degrading the performance provided to legitimate system users. Hence, a trade-off exists between system performance and the existence of a covert channel. Because of the threat of compromise that would be present in any multilevel computer system containing classified and sensitive information, the system should not contain covert channels with high bandwidth. The CTCPEC [6] does not impose any bandwidth restrictions on covert channels: "A 'reasonable' bandwidth restriction today may become too difficult to achieve in the future." The CTCPEC leaves it to the vendor to determine, given the operational environment in which a system will be used, the maximum covert channel bandwidth acceptable to the system's users. According to the TCSEC [11], a covert channel that has a bandwidth of > 100 BPS is considered high (100 BPS is the approximate rate at which many 'old' computer terminals are run [18]) and must be removed, or made to have a bandwidth that is in a lower classification. A covert channel that has a bandwidth of 10-100 BPS must be removed, made to have a bandwidth that is in a lower classification, or detected by the system, with the system auditing attempted uses of the channel. A covert channel that has a bandwidth of 1-10 BPS must be removed, made to have a bandwidth that is in a lower classification, detected by the system with the system auditing attempted uses of the channel, or documented as a covert channel in the system security administrator's manual. A covert channel that has a bandwidth of < 1 BPS must be removed, detected by the system with the system auditing attempted uses of the channel, documented as a covert channel in the system security administrator's manual, or simply ignored. Hence, there are five possible choices for resolving a covert channel relative to satisfying the B2 requirement in the TCSEC [42]; remove the channel, lower the bandwidth, audit attempted uses of the channel, document the channel, or simply ignore the channel.

5.3.1 Removing the Channel

Removing a covert channel requires changing the system mechanism that causes the covert leakage of information. If a covert channel is caused by a shared resource, then the sharability of that resource must be controlled or eliminated in order to eliminate the channel.

5.3.2 Lowering the Bandwidth of the Channel

Lowering the bandwidth means making the results of certain system operations less predictable, that is, noisier. A method of reducing channel bandwidths is to deliberately introduce spurious processes [77]. That is, user-level processes are introduced in the system to perform random alteration of channel variables. It should be noted that rules for limiting the bandwidth of covert storage channels are different from those for limiting the bandwidth of covert timing channels. For timing channels, either the speed of the transmitting or the receiving process can be reduced to affect a reduction in bandwidth. If the transmitter is limited, that is, it cannot transmit the desired number of bits during the executions of the Trojan Horse program, the receiver will not be able to receive the desired information. Also, if the receiver is limited so as not to be able to receive all of the bits being transmitted, it is pointless for the transmitter to transmit them. However, in storage channels, the receiver's speed is not limited, the receiver always has enough time to sense the data. Hence, all desired data can be transmitted during the executions of the Trojan Horse program.

5.3.3 Auditing the Use of Covert Channels

Auditing (sometimes called *logging*) is one of the most important and effective methods of increasing security in computer systems. Auditing is used as a method to handle covert channels. It allows all users to exploit known channels but provides a mechanism discouraging channel use. It is used as a deterrence of covert channel use. Thus, users can be assured of detection of any use of covert channels. Note, however, that the TCSEC [11] and the CTCPEC [6] require only that the *ability* to audit covert channels be provided – not that covert channels be actually audited. This detail limits somehow the effectiveness of audit as a real deterrent.

Explicit requirements for monitoring weaknesses of otherwise secure systems have been imposed for evaluation of commercially available systems. In particular, the ability to "audit identified events that may be used in the exploitation of covert storage channels" has been an explicit requirement for computer systems in the security classes B2-A1 of the TCSEC [11, 18]. As stated in [60], in spite of this requirement and the significance of covert channels to system security and integrity, neither the notion of auditing storage channels has been defined precisely nor any tools for such auditing have been provided for any multilevel secure system to date.

[18] and [60] stress upon the fact that covert channel auditing requires that sufficient data be recorded in audit trails to enable the identification of (1) individual covert channel use; (2) identification of transmitters and receivers of individual covert channel types (i.e., unambiguous identification of covert channel users). Furthermore, discovery of covert channel use must be *certain* (i.e., the covert channel auditing must not be circumventable), and false detection of covert channel use must be avoided. Circumvention of covert channel auditing is undesirable because it allows leakage of information to remain undetected. False detection of covert channel use is also undesirable because it may make it impossible to distinguish between innocuous user activity and covert channel use [18].

Estimation of actual covert channel leakage bandwidth is possible and desirable once covert channel use has been determined by audit trail analysis. Note that, in general, it is impossible to discover the actual information being leaked through a covert channel from audit trails because a user can encrypt it before leakage. Also, one cannot distinguish between real information and noise leakage merely by inspecting audit trails. Constant streams of either zero or ones are the only recorded patterns one can unambiguously classify as noise [18]. It is practically impossible to audit the use of certain covert channels, for example, the CPU timing channel discussed earlier in example 2.

5.3.4 Documentation of Covert Channels

Some covert channels are impractical to detect or audit, and their bandwidths are too low. These channels may be resolved by documentation. A security officer would like to know of the existence of such channels, so that programs can be examined for their

potential abuses. Hence, these covert channels and all others, even if they are otherwise ignored, are documented in the report resulting from the required covert channel analysis.

6 COVERT CHANNELS IN NETWORKS

Low level protocols typically provide a method of sending a block of data (of variable size within a given range) to a particular address on a network. In addition to the communication between the sender and the addressee that these methods obviously allow, extra information can often be detected by other network users which is independent of a data block's actual context (and therefore also independent of its presentation). When such detection is used for the transfer of information, covert channels exist.

6.1 Background

This section gives essential background to understand the next section (covert channel detection in networks). [23] and [62] may be consulted for more details.

A *packet* is equivalent to a network protocol data unit (NPDU). Packets contain not only user data intended for another DTE (Data Terminal Equipment) on a network, but also, control information by means of which the attached DTE and the network communicate.

6.1.1 Connection Control

An entity (e.g., a process in a multiprocessing environment, or a subroutine) can transfer data to another entity without prior coordination. This is known as *connectionless data transfer*. This mode of data transfer is not popular. A more commonly used mode is *connection-oriented transfer*. In this mode, a logical association, or *connection* is established between the entities before the actual data transmission begins. Three phases are involved in this type of mode [62]; connection establishment, data transfer, and connection termination. This mode can be viewed as sending messages through telephone lines, while the former is viewed as sending messages through a postal mailing system.

6.1.2 Packet-Switched Networks

A *packet-switched* network is a communication network that transmits data in packets. The network consists of a set of interconnected packet-switched nodes.

Transmitted data is in the form of a stream of packets. Each packet is routed through the network. As each node along the route is encountered, the packet is received, stored temporarily, and then transmitted along a link to the next node in the route.

Single packet activity can be explained with reference to Figure 6.1 (based on a similar figure in [62]). Consider data to be sent from station *A* to station *F*. *A* constructs a packet containing the data plus control information, including *F*'s address, and sends it to node 1. Node 1 stores the packet and determines the next node to the route (say 4). Hence, node 1 queues the packet for transmission over the 1-4 link. When the link is available, the packet is transmitted to node 4, which will forward the packet to node 6, and finally to station *F*.

Communication will typically involve a stream of packets exchanged in both directions between stations. Two approaches are used to manage the transfer and routing of these streams of packets; *datagram* and *virtual circuit*. Datagram is an example of connectionless data transfer; each packet is treated independently. The virtual circuit approach is an example of connection-oriented data transfer; a logical connection is established before any packets are sent. For example, suppose that *A* has one or more messages to send to *F*. It first requests a connection to *F*. Node 1 decides to route the request and all subsequent data to 4, which, in turn, decides to route the request and all subsequent data to 6, which finally delivers the request to *F*. If *F* is prepared to accept the connection, it sends out an accept to 6, which passes it back to 4, which passes it back to 1, and finally to *A*. Now, stations *A* and *F* may exchange data over the logical connection (or virtual circuit) that has been established. Each packet now contains a virtual circuit identifier as well as user data. Also, each node on the pre-established route knows where to direct the packets (no routing decisions are required). Hence, each packet from *A* traverses nodes 1, 4, and 6; each packet from *F* traverses nodes 6, 4, and 1. Eventually, one of the stations terminates the connection with a terminate request. Each station, at any time, can have more than one virtual circuit to any other station, and can have virtual circuits to more than one station.

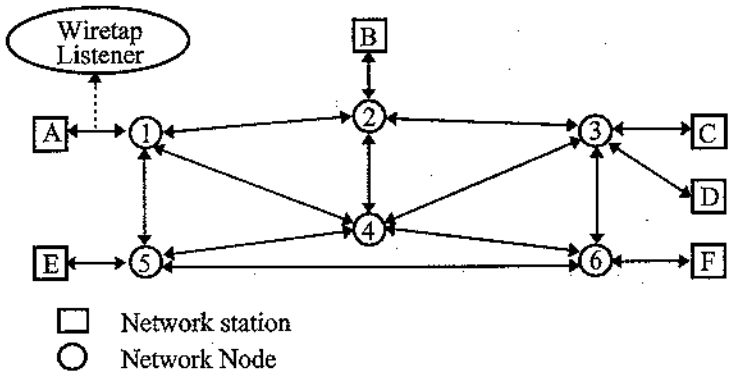


Figure 6.1: Packet-Switched Network.

6.1.3 A Packet-Switched Network Access Standard: X.25

The CCITT[6] recommendation X.25 is perhaps the best known and widely used protocol standard. It was originally approved by ISO[7] in 1976 and has been revised twice since, in 1980 and 1984 [62]. It specifies an interface between an open system and a packet-switched network – in other words, it specifies an interface between a DTE and DCE (Data Circuit-terminating Equipment), the DCE provides access to a packet-switched network. The protocol calls out the first lower three levels; the physical level, the data link level, and the packet level. The packet level specifies a virtual circuit service.

X.25 has two types of packet format [23]; data packets and control packets. In addition to user data, a data packet includes the virtual circuit number, the send and receive sequence numbers, and the M-bit (it also includes the D-bit and the Q-bit which are explained in the next section below). As mentioned above, a DTE is allowed to establish more than one virtual circuit with other DTEs (up to 4095 simultaneous virtual circuits [62]). To sort out which packet belongs to which virtual circuit, each packet contains a 12-bit virtual circuit number (4-bits as group number and 8-bits as channel number). The More bit (M-bit) is used in data packets to indicate to the network a sequence of more than one packet so that packets within a complete sequence can be combined by the network. Control packets include a virtual circuit number, a packet type identifier, and additional information regarding the particular control function.

[6] Consultative Committee of International Telegraphy and Telephone.
[7] International Standards Organization.

6.2 Direct Covert Storage Channels

As previously mentioned, there are two status bits in the first octet of all data packets, the Quality bit (Q-bit) and the Data Confirmation bit (D-bit). The Q-bit allows the user to distinguish two types of data field information. The D-bit is used for end-to-end acknowledgment of data packets, that is acknowledgment between the local DTE and the network (D = 0). D = 1 indicates that the acknowledgment is between a remote terminal and the network. The usual case is D = 0. Both Q-bit and D-bit are potential 1-bit channels. If both are modulated, two bits of covert data may be sent.

6.3 More Covert Channel Examples

1) A Derived Covert Storage Channel:

If a number of differently sized packets can be sent to legitimate addresses, then a wiretapper (or an entity within the addressed network component) can derive information from which block size was chosen by the transmitter. The length of a packet may vary, depending on its type. For example, the length of a data packet varies with the number of user data octets sent, which varies from 1 to 128. The only other restriction is that the packet contains an integral number of octets. A 1-bit channel could be set up by either sending a data packet with a specific number of octets or any other number of octets [24].

2) A Covert Timing Channel:

If a wiretapper (or an entity within the addressed network component) can distinguish a number of different delays between successive transmissions imposed by a user, then he may deduce information from the particular delays chosen [24]. For example, the sender would take d_1 seconds to transmit a packet (to signal a 0) or d_2 seconds to transmit it (to signal a 1).

7 MULTILEVEL DBMS COVERT CHANNELS

As mentioned earlier, studies have shown that the mechanism provided by database systems often can be bypassed owing to flaws in the systems which host the DBMS [36, 74]. Moreover, there is no multilevel DBMS that is rated at the B2 trust level (or higher) available on the market today and therefore there is only very little experience in building and evaluating trusted DBMSs. In a multilevel relational DBMSs, where data is usually

interrelated, covert channels pose a real threat to data compromise. There are no established guidelines or expositions of exactly what constitutes a database covert channel [36]. Since multilevel DBMSs are fertile ground for encoded (unauthorized) information transfer, a covert channel recognition/resolution model for users, developers, and evaluators alike is essential.

The next section gives covert storage channel examples in a multilevel DBMS.

7.1 Covert Channel Examples

One way in which covert channels arise in multilevel databases is through the creation and destruction of objects. A high user can encode classified information in the existence of objects (tables, or records within a table) which are then detected by a low user who can decode the information.

A Trojan Horse program (a trap) can be inserted during any phase of a system life cycle (i.e., during programming, operation, maintenance) by any individual involved in any of the phases. The main problem in handling this threat is the difficulty of discovering the existence of a Trojan Horse. A number of Trojan Horse threats in a trusted multilevel DBMS (based on similar scenario given in [36]) are given below:

The Situation

- A department of defense of a country keeps track of its weaponry manufacturers with a tracking file (TRACKF). That file is structured as follows:

CLASS	PRODUCT-ID	PROD-CAPACITY	PROD-DATE	LOCATION
S	*XCM*	8	910101	60N40E

CLASS: TOP-SECRET (TS), SECRET (S), or UNCLASSIFIED (UNCLAS).
PRODUCT-ID: Product identification.
PROD-CAPACITY: The yearly production capacity for each manufacturer.
PROD-DATE: The starting date of production for each manufacturer.
LOCATION: The location of each manufacturer.

- PRODUCT-IDs are joined to a Product-Description file (PRODESC) in which each product, its usage, and other related information is described.

Common Knowledge

- The product capacity that the database shows is multiplied by 100 to give the actual capacity of each manufacturer.
- 10% of the manufacturers produce spare parts of all types of war products.
- PRODUCT-ID's for all nuclear products are changed every week. They are changed every month for other products.
- The total number of manufacturers located in different sites is 720.
- Any information about any nuclear product is TOP-SECRET.
- The total number of nuclear weapons manufacturers is TOP-SECRET.

A Sample of TRACKF Data

CLASS	PRODUCT-ID	PROD-CAPACITY	PROD-DATE	LOCATION
S	*XCM*	8	910101	60N40E
S	*XCP*	9	910131	30S20W
TS	*XAB*	3	921231	10N50E
S	*XCT*	5	019115	05N70E
.				
.				
.				
UNCLAS	*XXX*	6	...	
UNCLAS	*XXX*	4	...	
UNLCAS	*XXX*	7	...	
.				
.				
.				
UNCLAS	*XXX*	2	...	
UNCLAS	*XXX*	1	...	
.				
.				
.				
UNCLAS	*XXX*	9	910531	...

A query issued by a "SECRET" level user:

"SELECT * FROM TRACKF;"

Discretionary and Mandatory Access Controls satisfied.

Response

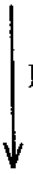
... *XCM* ...
... *XCP* ...
... *XCT* ...
... *XCM* ...

Database Covert Channel Description:

Legitimate storage objects satisfying the term of the query are returned to the user (subject) more than once. Since the user is cleared to receive each object, then no overt illegal disclosure occurs. Yet, by interpreting the number of times each legitimate storage object is returned, the subject could receive encoded data. In this example, say the object *XCM* is returned 24 times, *XCP* is returned once, and *XCT* is returned twice, then this information can be interpreted as *XAB*, where each number (24, 1, and 2) represent the indexed position of *X*, *A*, and *B*, respectively, in the English alphabet.

A query issued by a "SECRET" level user:

"SELECT * FROM TRACKF WHERE PRODUCT-ID EQ "XAB";"



Response

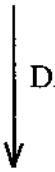
... *XXX 3* ...
... *XXX 6* ...
... *XXX 4* ...
... *XXX 7* ...

Database Covert Channel Description:

Here, legitimate storage objects (i.e. whose classification is dominated by the subject's clearance) are returned, but they are from a different (wrong) named object of identical structure to the named object referenced in the query. That is, these storage objects appear in structure to satisfy the query and are legally disclosed to the subject according to DAC and MAC subpolicy. These storage objects are "wrong" in that they do not correspond to the query condition, yet they are legally disclosed to the subject. They are correct in terms of DAC and MAC, but simply they do not answer the question asked. In this example, the user gets all unclassified information. Notice that the row ... *XXX 3* ... does not exist in the SECRET or UNCLASSIFIED information that the database shows. Hence, the SECRET user discovers that the attribute "3" is a PROD-CAPACITY of a TOP-SECRET level data.

A query issued by a "SECRET" level user:

"SELECT * FROM TRACKF;"



DAC & MAC satisfied.

Response

.
.
... *XCM* ...
... *XCM* ...
... *XCM* ...
... *XCM* ...
... *XCM* ...
... *XCM* ...
... *XCM* ...
... *XCM* ...
... *XCM* ...
... *XCP* ...
... *XCP* ...
... *XCT* ...
... *XCM* ...
... *XCM* ...
... *XCP* ...
... *XCP* ...
... *XCP* ...
... *XCT* ...
.
.

Database Covert Channel Description:

Legitimate and correct data is returned to the user in a particular order ("Sneaky" fashion) so that the user can acquire unauthorized information based on this ordering. Notice that the row ...*XCM*... occurred 9 times, the row ...*XCP*... occurred twice, the row ...*XCT*... occurred once, the row ...*XCM*... occurred twice this time, the row ...*XCP*... occurred 3 times, and the row ...*XCT*... occurred once. Collecting and arranging the number of times each of these rows occurred, we get 921231, the PROD-DATE field of the TOP-SECRET record.

A query issued by a "SECRET" level user:

"SELECT * FROM TRACKF;"

↓ DAC & MAC satisfied.

Response

.
.
...*XCM*...
...*XCM*...
. } 10 times
.
...*XCM*...
...*XCP*...
...*XCP*...
. } 14 times
.
...*XCP*...
...*XCT*...
...*XCT*...
. } 50 times
.
...*XCT*...
...*XCM*...
...*XCM*...
...*XCM*...
...*XCM*...
...*XCM*...
.
.

Database Covert Channel Description:

The row ...*XCM*... occurred 10 times, the row ...*XCP*... occurred 14 times, the row ...*XCT*... occurred 50 times, and the row ...*XCM*... reappeared 5 times. Taking the number of times each of these rows occurred we get: 10, N (letter indexed by 14 in the English alphabet, 50, and E (letter indexed by 5) ⇒ 10N50E, the LOCATION field of the TOP-SECRET record.

A query issued by a "SECRET" level user:

"SELECT * FROM TRACKF WHERE PRODUCT-ID "NOW" EQ PRODUCT-ID "LAST WEEK";"

DAC & MAC satisfied.

↓

Response

.
.
...*XCM*...
.
.

Database Covert Channel Description:

Number of rows returned = 570
90% of 720 = 648 (recall that 720 is the total number of manufacturers out of which 10% produce spare parts)

```
  648
- 570
   78
```

at least 78 nuclear manufacturers exist. This TOP-SECRET information is inferred by the SECRET user. *Inference* is a serious problem in multilevel trusted databases. [36] argues that inference channels are covert channels. [16] and [50] refer to inference channels as covert channels. However, because these channels do not require processes (one high and one low) to form the illegal communication path, they are excluded by others.

Another example of a relational DBMS is through returning legitimate storage objects to a subject, *but not all* of the storage objects satisfying the query are returned [36]. By knowing the correct number of authorized objects satisfying the query, the subject could interpret encoded data about unauthorized objects based on *the number of missing objects*.

8 CONCLUSIONS

To satisfy the requirements for multilevel security in computer systems, the covert channel problem must be faced. This report referred to covert channel identification and handling methods which help assure that existent covert channels do not compromise a system's secure operation. Exploitation of covert channels will grow in the future as more systems are put into operation in the multilevel mode. By continuing the development and improvement of tools that support covert channels analysis, perhaps the task of detecting covert channels can be made routine enough so that the problem is not avoided. Absolute security is not possible for any system without severely limiting its functionality. This is particularly true for MLS-DBMS. However, it is possible to make reasonable tradeoffs between functionality and security. Within the B2 security requirements, the covert channel problem is thought to be solvable in a way so as not to limit the usefulness of the system. Since MLS-DBMSs provide fertile ground for encoded (unauthorized) information transfer, a covert channel recognition/resolution model for users, developers, and evaluators alike is essential.

BIBLIOGRAPHY

1. Alagic, S., Relational Database Technology, *Springer-Verlag,* 1986.

2. Al-Kadi Ibrahim, "Origin of Cryptology: The Arab Contributions," *Cryptologia*, Vol. 16, No. 2, pp. 97-125, April 1992.

3. Bell, D. E., and La-Padula, L. J., "Secure Computer System: Unified Exposition and Multics interpretation," *Technical Report ESD-TR-75-306, AFSC,* Hanscom Airforce Base, Bedford, MA, 1976.

4. Boebert, W. E., Dillaway, B. B., and Haigh, J. T., "Mandatory Security and Data Management," *Proceedings, The National Computer Security Center Workshop on Database Security,* June 1986.

5. Braun, H., "Security Considerations Within Military Information Systems," *Seminar on Basic Documentation Practices (AGARD-R-788),* Ankara, Turkey, pp. 11/1-17, 7-8 September 1992.

6. Canadian System Security Center, Communication Security Establishment, Government *of* Canada, *The Canadian Trusted Computer Product Evaluation Criteria,* Version 3.0e, January 1993.

7. Denning, Dorothy, E., "A Lattice Model of Secure Information Flow," *Communication of the ACM*, Vol. 19, No. 5, pp. 236-243, May 1976.

8. Denning, Dorothy, E. and Peter. J. Denning, "Certification of Programs for Secure Information Flow," *Communication of the ACM*, Vol. 20, No. 7, pp. 504-513, July 1977.

9. Denning, Dorothy, E., Cryptography and Data Security, *Addison-Wesley*, Reading, Massachusetts, 1983 (reprinted).

10. Diffie, W. and M. Hellman, "New Directions in Cryptology," *IEEE Transactions on Information Theory*, Vol. IT-22, pp. 644-654, 1976.

11. DoD Computer Security Center, *US Department of Defense Trusted Computer System Evaluation Criteria,"* 5200-28-STD, December 1985.

12. Dougall, E. G., "Computer Security, IFIP Transactions," A-37, *North Holland*, 1993.

13. Eggers, K. W., and Mallett, P. W., "Characterizing Network Covert Storage Channels," *Proceedings, 4th Aerospace Computer Security Applications Conference,* Orlando, FL, December 1988.

14. Gardner, P. E., "Evaluation of Five Risk Assessment Programs," *Computers and Security*, Vol. 8, pp. 279-285, 1989.

15. Gallager, R. G., Information Theory and Reliable Communication, *John Wiley and Sons,* New York, N.Y, 1968.

16. Garvey, T. D., Lunt, T. F., and Stickel, M. E., "Abductive and Approximate Reasoning Models for Characterizing Inference Channels," *Proceedings, The 4th Workshop on the Foundations of Computer Security,* Franconia, NH, June 1991.

17. Gilles G. and Outerbridage, R., "DES Watch: An Examination of the Sufficiency of the Data Encryption Standard for Financial Institution Information Security in 1990's," *Cryptologia*, Vol. 15, No. 3, pp. 177-193, July 1991.

18. Gligor, V. D., A Guide to Understanding Covert Channel Analysis of Trusted systems, *National Computer Security Center,* NCSC-TG-030, Version 1, November 1993.

19. A Guide to Understanding Covert Channel Analysis of Trusted Systems, *National Computer Security Center*, NCSC-TG-030, Version 1, November 1993.

20. Goguen, J. A. and Meseguer, "Security Policies and Security Models," *Proceedings, IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 11-20, April 1982.

21. Hahn, M., "Redefining Data Sharing," *EDPACS*, Vol. 19, No. 9, March 1992.

22. Haigh, T. J., R. A. Kemmerer, J. McHugh, and W. D. Young, "Experience Using Two Covert Storage Channel Analysis Techniques on a Real System Design," *Proceedings, 7th International IEEE Symposium on Security and Privacy,* Oakland, CA, 7-9 April 1986.

23. Halshal, F., Data Communications, Computer Networks and Open Systems, *Addison-Wesley,* Third Edition, 1992.

24. Harrison, S., "Introduction to Covert Channels in Communications Systems," *Proceedings, 5th Annual Canadian Computer Security Symposium,* pp. 363-376, 1993.

25. Hinke, T. H., "Inference Aggregation Detection in Database Management systems," *Proceedings, IEEE Symposium on Security and Privacy,* Oakland, CA, April 1988.

26. Hsiao, D. K., Kohler, J., and Stroud, S. K., "Query Modifications as a Means of Controlling Access to Multilevel Secure Databases," *1991 IFPI, Database Security, IV: Status and Prospects*, S. Jajodia and C. E. Landwehr (Editors), Elsevier Science Publishers, North-Holland, pp. 221-240, 1991.

27. Hu, W. M., "Reducing Timing Channels with Fuzzy Time," *Proceedings, IEEE Symposium on research in Security and Privacy*, Oakland, CA, pp. 8-20, 1991.

28. Huskamp, J. C., Covert Communication Channels in Timesharing Systems, *Technical Report UCB-CS-78-02*, Ph.D. Thesis, University of California, Berkeley, CA, 1978.

29. Jajodia, S. and R. Sandhu, "Database Security: Current Status and Key Issues," *ACM SIGMOD RECORD,* Vol. 19, pp. 123-126, December 1990.

30. Jajodia, S. and R. Sandhu, "Towards a Multilevel Secure Relational Data Model," *ACM SIGMOD 1991 Conference Proceedings,* pp. 50-59, 1991.

31. Jajodia, S., "Tough Issues: Integrity and Auditing in Multilevel Secure Databases," *Proceedings, 13th National Computer Security Conference,* Washington, D.C., Vol. 2, pp. 577-580, 1-4 October 1990.

32. Kaiser, W. G., "The Making of a B2 System," *Information Age (UK),* Vol. 1, No. 10, pp. 41-46, January 1988.

33. Kemmerer, R.A., "Shared Resource Matrix Methodology: An Approach to Identify Storage and Timing Channels," *ACM Transactions on Computer Systems,* Vol. 1, No. 3, pp. 256-277, August 1983.

34. Kemmerer, R. A., "A Practical Approach to Identifying Storage and Timing Channels, "*Proceedings, IEEE Computer Society Symposium on Research in Security and Privacy,* Oakland, CA,, 26-28, April 1982.

35. Kemmerer, R. A. and P. A. Porras, "Covert Flow Trees: A Visual Approach to Analyzing Covert Storage and Timing Channels," *IEEE Transactions on Software Engineering*, Vol. 17, No. 11, pp. 1116-1184, November 1991.

36. Knode, R. B., "A Covert Channel Taxonomy for Trusted Database Management Systems," *C31 Systems Division, Atlantic Research Corporation,* Hanover, MD, 1992.

37. Lamport, Leslie, "What is Meant for a Concurrent Program to Satisfy a Specification: Why No One Has Specified Priority," *Proceedings of ACM Symposium on Principles of Programming Languages*," pp. 78-83, 1985.

38. Lamport, Leslie, "A Simple Approach to Specifying Concurrent Systems," *Communications of the ACM*, Vol. 32, No. 1, January 1989.

39. Lampson, Butler W., "A Note on the Confinement Problem," *Communications of the ACM,* Vol. 16, No. 10, pp. 613-615, October 1973.

40. Lapid, Y., Ahituv, N., and S. Neuman, "Approaches to Handling 'Trojan Horse' Threats," *Computers and Security,* Vol. 5, pp. 251-256, September 1986.

41. Lipner, Steven B., "A Comment on the Confinement Problem," *Proceedings, 5th Symposium on Operating System Principles,* Austin, Texas, 19-21 November 1975.

42. Loepere, K., "Resolving Covert Channels Within A B2 Class Secure System," *ACM SIGOPS*, pp. 9-28, June 1986.

43. Lunt, T. F., and E. B. Fernandez "Database Security," *SIGMOD RECORD (USA), Vol. 19, No. 4*, pp. 90-97, December 1990.

44. Lunt, T. F., "Aggregation and Inference: Facts and Fallacies," *Proceedings, IEEE 1989 Symposium on security and privacy,* Oakland, Ca, May 1989.

45. Lunt, T. F., "Polyinstantiation: an Inevitable Part of A Multilevel World," *Proceedings, IEEE 1991 Symposium on security and privacy,* Oakland, Ca, pp. 236-238, 1991.

46. Lunt, T. F., Denning, D., Schell, R., Heckman, M., and Shokley, W., "The SeaView Security Model," *IEEE Transactions on Software Engineering,* Vol. 16, No. 2, pp. 190-209, June 1990.

47. Lunt, T. F., Research Directions in Database Security, *Springer-Verlag*, 1992.

48. McCord, R., "A Critically Database for Military Users," *Conference Proceedings MILCOMP '89, Military Computer Systems and Software,* London, UK, pp. 159-63, 26-28 September 1989.

49. McCullough, D., "Covert Channels and Degrees of Insecurity," *Proceedings, 1988 Franconia Computer Security Foundations Workshop: The Mitre Corporation,* 1988.

50. Meadows, Cathrine, "An Outline of a Taxonomy of Computer Security Research and Development," *1992-1993 ACM SIGSAC New Security Paradigms Workshop*, pp. 33-41, 1993.

51. Melliar-Smith, P. M., and Moser, L. E., "Protection Against Covert Channels and Timing Channels," *Proceedings, Computer Security Foundations Workshop IV,* Los Alamitos, CA, pp. 209-214, 18-20 June 1991.

52. Millen, J. K., "Foundations of covert Channel Detection," *MTR-10538, The Mitre Corporation*, Bedford, MA, January 1989.

53. Millen, J. K., "Finite-State Noiseless Covert Channels," *Proceedings, Computer Security Workshop II,* Franconia, NH, pp. 81-86, 11-14 June 1989.

54. "Minutes of the First Workshop on Covert Channel Analysis", *Cipher: Newsletter, IEEE Computer Society Technical Committee on Security and Privacy*, July 1990.

55. Mirkovic, M., B. Pavic, M. Vujasinovic, and R. Bojovic, "*Computer Data Ciphering by Use of Chinese Remainder Theorem*," Communication, Control, and Signal Processing, E. Arikan, Editor, Elsevier Science Publishers B.V., 1990.

56. Morgenstern, M., "Controlling Logical Inference in Multilevel Database Systems," *Proceedings, IEEE Symposium on Security and Privacy*, pp. 245-255, April 1988.

57. Muftic, S., Security Mechanisms for Computer Networks, *Ellis-Harwood limited,* 1989.

58. Patterson, W., "Mathematical Cryptology for Computer Scientists and Mathematicians," *Rowman and Littlefield Publishers*, 1987.

59. Shannon, C. E., and W. Weaver, The Mathematical Theory of Communication, *The University of Illinois Press*, Urbana, Illinois, 1964.

60. Shieh, S. P. and V. D. Gligor, "Auditing the Use of Covert Channels in Secure Systems," *Proceedings, IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1990.

61. Stachour, P. and Thuraisinghan, B., "Design of LDV: A Multiple Secure Relational Database System," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 2, No. 2, pp. 190-209, June 1990.

62. Stallings, William, Handbook of Computer Communications Standards: Volume 1, 2nd Edition, MacMillan, 1990.

63. Stallings, William, Network and Internetwork Security, Principles and Practice, *Prentice Hall,* 1995.

64. Sterne, D., G. Benson, C. Landwehr, L. LaPadula, and R. Sandhu "Reconsidering the Role of the Reference Monitor," *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, Franconia, NH, June 14-16, 1994.

65. Sterne, D., G. Benson, and H. Tajalli, "Redrawing the Security Perimeter of a Trusted system," *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, Franconia, NH, June 14-16, 1994.

66. Trostle, J. T., "Multiple Trojan Horse System and Covert Channel Analysis," *Proceedings, Computers Security Foundations Workshop IV,* Los Alamitos, CA, pp. 22-33, 18-20 June 1991.

67. Tsai, C.R., "Covert Channel Analysis in Secure Computer Systems", *Department of Electrical Engineering, University of Maryland*, Ph.D. Dissertation, August 1987.

68. Tsai, C. R. and V. D. Gligor, "A Bandwidth Computation Model for Covert Channels and its Applications," *Proceedings, IEEE Symposium on Security and Privacy,* Oakland, CA, 1988.

69. Tsai, C. R., V.D. Gligor, and C .S. Chandersekaran, "A Formal Method for Identification of Covert Storage Channels in Source Code," *Proceedings, IEEE Symposium on Security and Privacy,* Oakland, CA, April 1987.

70. Tsai, C. R. V.D. Gligor, and C. S. Chandersekaran, "On The Identification of Covert Storage Channels in Secure Systems," *IEEE Transactions on Software Engineering,* Vol. 16, No. 6, pp. 569-580, June 1990.

71. Wells, D. L., "Achieving Database Protection Through the Use of Subkey Encryption," *The University of Wisconsin-Milwaukee*, Ph.D. Dissertation, 1980.

72. Whitfield Diffe, Martin E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," *Proceedings of IEEE*, Vol. 67, No. 3, March 1979.

73. Wiseman, S., A. Wood, and S. Lewis, "The Trouble with Secure Databases," *Conference Proceedings, MILCOMP '89, Military Computer Systems and Software,* London, UK, pp. 164-170, 26-28 September 1989.

74. Wiseman, Simon, "The Conflict Between Confidentiality and Integrity," *Conference Proceedings, MILCOMP '91, Military Computer Systems and Software,* London, UK, pp. 241-242, 1991.

75. Wray, J. C., "An Analysis of Covert Timing Channels," *Proceedings, IEEE Computer Society Symposium on Research in Security and Privacy,* Oakland, CA, 20-22, pp. 2- 7, May 1991.

76. Voydock, V. L. and S. T. Kent, "Security Mechanisms in High-level Network Protocols," *Computing Surveys*, Vol. 15, No. 2, pp. 135-171, June 1983.

77. Fadlalla, Y. A and R. H. Cooper, "An Approach to Solving the Covet Storage Channel Problem," Proceedings of CADEM '95: International Conference of Computer-Aided Design and Management, Xian, China, 12-16 October 1995.