- sender transfers control to receiver;
- receiver reads the channel variable;
- receiver records 1;
- receiver transfers control to sender

To send 1 in state 1:

- sender transfers control to receiver;
- receiver reads the channel variable;
- receiver records 1;
- receiver transfers control to sender.

One can determine the time required to send a 0 or 1 by listing the corresponding sequence of TCB primitive calls and adding up their times. Recall that the TCB primitive calls, and their total duration, depend on the state of the channel. Also, recall that the reading (not just the setting) of a 0 or a 1 will have different durations even if they are represented by the same TCB primitive call. For example, if the reading of a 0 in one state is represented by the "open" primitive call with a successful return and in the other state by "open" with a failure return, the reading of the 0 in the two states will have different durations because the latter call always has a shorter duration. The sequences of TCB primitive calls necessary to transfer 0s and 1s using a two-state graph may be different, and thus they may take four different amounts of time, say a, b, c, and d time units, respectively (as shown in Figure 4-1).

To determine the bandwidth of a channel represented with a two-state graph, N(t), one must find the number of possible transmissions of duration t. The bandwidth (i.e., capacity) of a channel can be expressed in terms of $N_h(t)$ as follows:

$C = \lim(\log_2 N_h(t))/t.$

To find $N_h(t)$, let $N_0(t)$ be the total possible number of transmissions of duration exactly t beginning in one of the two states, and let $N_1(t)$ be the total possible number of transmissions of duration exactly t beginning in the other state. (In general, there will be an $N_h(t)$ for the h-th state where h ranges over the state set.) The number of transmissions satisfies a system of difference equations that can be read off the two-state graph. Each equation is based on the fact that the set of transmissions beginning in a given state consists of a union of several disjoint sets, discriminated by the initial symbol of the transmissions. The number of transmissions with a given initial symbol is equal to the total number of (shorter) transmissions beginning in the next state after the transition for that symbol.

The following system of equations can be used for the file-lock channel:

$N_0(t)=N_0(t - a)+N_1(t - c)$

$N_1(t)=N_0(t - b)+N_1(t - d)$

In general, the h-th equation has the form:

$N^*(t', =^* i(N_i(t - T_{hi}),$

where T_{hi} is the time taken by a transition from state h to state i.

Note that $N_0(t)$ is nonzero only for those values of t that are expressible as a sum of multiples of a, b, c, and d. To determine the bandwidth of the channel, it is only necessary to find the asymptotic upper limit of $N_0(t)$ as t approaches infinity [Shannon and Weaver64]. This may be found in the form:

$N_h(t)=A_h*xt$

Substituting this solution, we obtain the system of equations:

$A_h*xt =**(A_i*xt-T_{hi}$

and $C = \lim(\log_2(A_h(x^t))/t = \log_2 x$, when t*infinity.

Note that there may be multiple solutions for x in the above equations. The largest solution provides the bandwidth (capacity).

We can express this system of equations in matrix form as (P-I)A = 0, where P is a matrix of negative powers of x. Since (P-I) is singular, its determinant Det(P-I) = 0. Figure 4-2 shows the system of equations, their determinant, and the solution.

Example 11 - Application to Two Secure Xenix Channels

Two of the Secure Xenix channels whose bandwidths were computed in refer- ence [Tsai and Gligor88] for a PC/AT configuration are the inode table channel and the upgraded directory channel. In this example we illustrate Millen's method described above using measurements of Secure Xenix TCB primitives on an IBM PS/2 model 80 configuration. T_{cs} represents the context switch time, which is 3 milliseconds. The values of T_r (T_s) represent the duration of reading (setting) the covert channel variable, and the value of T_{env} represents the duration of setting up the transfer environment (e.g., a state transition).

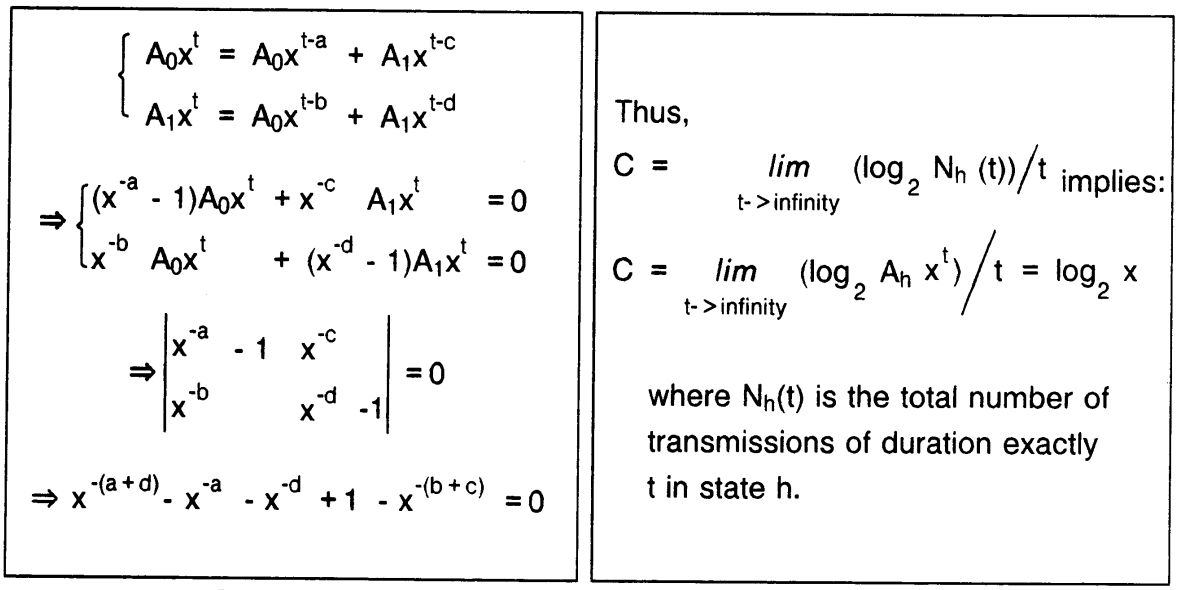

Figure 4-2. Simultaneous Equations, Determinant, Capacity (Bandwidth)

The Inode Table Channel

In this example the state 0 of the inode table channel is represented by the inode table full state, and the state 1 by the node table nonfull state. Figure 4-3 shows the state transitions defined

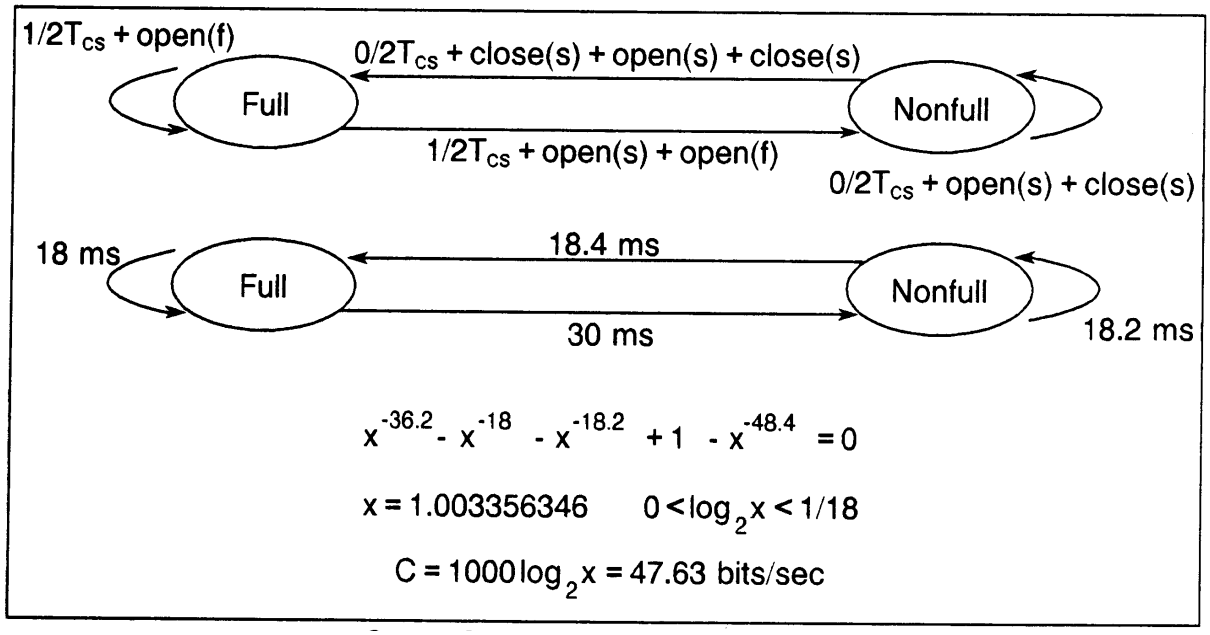

Figure 4-3. State Graphs for the Inode Table Channel

State 0:

When the inode table is full, two T_{cs} and one viewing primitive "open(f)" with a failure return are needed to transfer a 1 from a sending process to a receiving process. Thus, the following times are needed to transfer a 1 from state 0:

$T_r(\text{full}\rightarrow\text{full}) = \text{open(f)}$, $T_s(\text{full}\rightarrow\text{full}) = 0$, $T_{env}(\text{full}\rightarrow\text{full}) = 0$.

When switching from the full state to the nonfull state, an alteration primitive "close(s)," a viewing primitive "open(s)," an environment set-up primitive "close(s)," and two T_{cs} are needed to send a 0. Thus, the following times are needed to transfer a 0 from state 0:

$T_r(\text{full}\rightarrow\text{nonfull}) = \text{open(s)}$, $T_s(\text{full}\rightarrow\text{nonfull}) = \text{close(s)}$, $T_{env}(\text{full}\rightarrow\text{nonfull}) = \text{close(s)}$.

State 1:

When the transition is from the nonfull state to the nonfull state, a viewing primitive "open(s)," an environment set-up primitive "close(s)," and two T_{cs} are needed to transfer a 0. Thus, the following times are needed to transfer a 0 from state 1:

$T_r(\text{nonfull}\rightarrow\text{nonfull}) = \text{open(s)}$, $T_s(\text{nonfull}\rightarrow\text{nonfull}) = 0$, $T_{env}(\text{nonfull}\rightarrow\text{nonfull}) = \text{close(s)}$.

When switching from the nonfull state to the full state, an alteration primitive "open(s)," a viewing primitive "open(f)," and two T_{cs} are needed to transfer a 1. Thus, the following times are needed to transfer a 0 from state 1:

$Tr(\text{nonfull}\rightarrow\text{full}) = \text{open(f)}$, $T_s(\text{nonfull}\rightarrow\text{full}) = \text{open(s)}$, $T_{env}(\text{nonfull}\rightarrow\text{full}) = 0$.

The bandwidth (i.e., capacity) of this channel, denoted by C in Figure 4-3, is 47.63 bits/second.

The Upgraded Directory Channel

In this example the state 0 of the upgraded directory channel is represented by the directory-full state, and the state 1 by the directory-nonfull state. The state transitions defined below and their durations are shown in Figure 4-4.

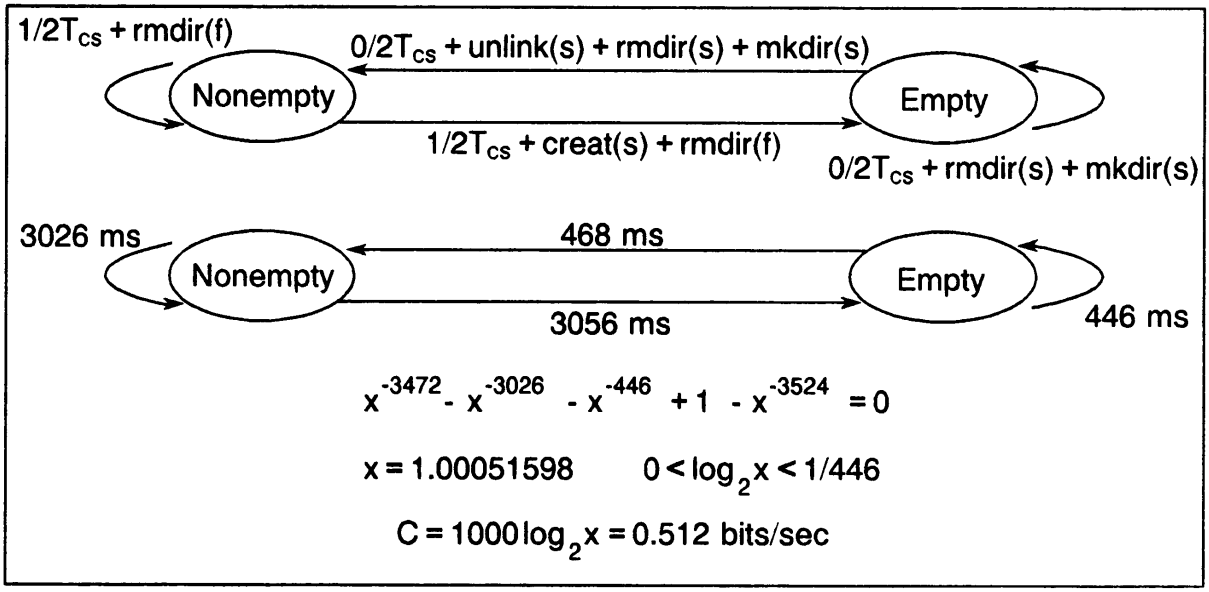

Figure 4-4. State Graphs for the Upgraded Directory Channel

State 0:

When an upgraded directory is nonempty, two T_{cs} and one viewing primitive "rmdir(f)" with a failure return are needed to transfer a 1 from a sending process to a receiving process.

T_r(nonempty→nonempty) = rmdir(f), T_s(nonempty→nonempty) = 0,

T_{env}(nonempty→nonempty) = 0.

When switching from the nonempty state to the empty state, an alteration primitive "unlink(s)," a viewing primitive "rmdir(s)," an environment set-up primitive "mkdir(s)," and two T_{cs} are needed to send a 0. Thus, the following times are needed to transfer a 0 from state 0:

T_r(nonempty→empty) = rmdir(s), T_s(nonempty→empty) = unlink(s),

T_{env}(nonempty→empty) = mkdir(s).

State 1:

When the transition is from the empty state to the empty state, a viewing primitive "rmdir(s)," an environment set-up primitive "mkdir(s)," and two T_{cs} are needed to transfer a 0. Thus, the following times are needed to transfer a 0 from state 1:

T_r(empty→empty) = rmdir(s), T_s(empty→empty) = 0, T_{env}(empty→empty) = mkdir(s).

When switching from the empty state to the nonempty state, an alteration primitive "creat(s)", a viewing primitive "rmdir(f)", and two T_{cs} are needed to transfer a 1. Thus, the following times are needed to transfer a 1 from state 1:

T_r(empty→nonempty) = open(f), T_s(empty→nonempty) = open(s),

T_{env}(empty→nonempty) = 0.

The bandwidth of this channel is denoted by C in Figure 4-4 and is 0.512 bits/second.

4.2.2 Informal Method for Estimating Covert Channel Bandwidth

A simple formula for computing the maximum attainable bandwidth of a noise-less covert channel in absence of any spurious processes that would delay senders and receivers was presented in [Tsai and Gligor88]. The formula is:

$B(0) = b*(T_r + T_s + 2T_{cs})**(-1)$,

In this formula, b represents the encoding factor (which we assume to be 1 in most practical cases), and where n is the number of total possible transitions. T5(i) and Tr(i) are the times necessary to set and read a 0 or a 1 after having transmitted a 0 or a 1. Thus, n = 4. Tenv(i) is the time to set up the environment to read a 0 or a 1. Note that in these formulas it is assumed that all environment setup for both variable reading and setting is done by the receiving processes.

In deriving this formula it is assumed that the setting of 0's and 1's take the same amount of time, and that all transmissions contain an equal distribution of 0's and 1's.

Example 12 - Application of the Bandwldth EstImation Formula

The maximum bandwidths of the two channels of Example 11 can be recalculated by using the above formula, as follows:

The Inode Table Channel

$T_s = [T_s(full→full) + T_s(full→nonfull) + T_s(nonfull→nonfull) + T_s(nonfull→full)]/4$

= [0 + close(s) + 0 + open(s)]/4

=(open+close)/4 = (12 + .2)/4 = 3.05 (ms)

T_r = [T,(full→full)+ T_{env}(full→full) + T_r(full→nonfull) + T_{env}(full→nonfull) + T_r(nonfull→nonfull) + T_{env}(nonfull→nonfull) + T_r(nonfull→full) + T_{env}(nonfull→full)]/4

= [open(f) + 0 + open(s) + close(s) + open(s) + close(s) + open(f) + 0]/4

=open + close/2 = 12.1 (ms)

Therefore,

B(0) = 1000/(l2.1 + 3.05 + 6) 47.28 bits/sec

The Upgraded Directory Channel

$T_s = [T_s(\text{nonempty}\rightarrow\text{nonempty}) + T_s(\text{nonempty}\rightarrow\text{empty}) + T_s(\text{empty}\rightarrow\text{empty})$

$+ T_s(\text{empty}\rightarrow\text{nonempty})]/4$

$= [0 + \text{unlink(s)} + 0 + \text{creat(s)}]/4$

$=(\text{creat} + \text{unlink})/4 = (30 + 22)/4 = 13(\text{ms})$

$T_r = [T_r(\text{nonempty}\rightarrow\text{nonempty}) + T_{env}(\text{nonempty}\rightarrow\text{nonempty}) + T_r(\text{nonempty}\rightarrow\text{empty})$

$+ T_{env}(\text{nonempty}\rightarrow\text{empty}) + T_r(\text{empty}\rightarrow\text{empty}) + T_{env}(\text{empty}\rightarrow\text{empty})$

$+ T_r(\text{empty}\rightarrow\text{non-empty}) + T_{env}(\text{empty}\rightarrow\text{nonempty})]/4$

$= [\text{rmdir(f)} + 0 + \text{rmdir(s)} + \text{mkdir(s)} + \text{rmdir(s)} + \text{mkdir(s)} + \text{rmdir(f)} + 0]/4$

$= [\text{rmdir(s)} + \text{rmdir(f)l/2} + \text{mkdir/2}$

$=(180+3020)/2+260/2=1730\ (\text{ms})$

Therefore,

$B(0) = 1000/(1730 + 13 +6)=0.572\text{bits/sec}$

4.2.3 Differences between the Two Methods

Comparing the results of Examples 11 and 12 one might be tempted to conclude that the two bandwidth computation methods yield similar results for all covert channels. This conclusion, however, is not always the case. Millen's method yields higher bandwidths whenever the times to set up transmission environments and/or those to transmit 0s differ significantly from those to transmit 1s. This may be the case after delays are placed in some but not all TCB primitives of a channel (e.g., in the error return path of a primitive needed to use the channel; this ensures that undue performance penalty is not incurred.). Subsequent recomputation of the (delayed) channel bandwidth by the two methods would yield significantly different results. Experience with using the two methods for Secure Xenix shows that in cases where the times to transmit a 0 and a 1 are close, the two methods yield results that differ by at most 20%.

Millen's method is superior to that presented in [Tsai and Gligor88] not only because it always helps compute the maximum attainable bandwidth but also because during its use one is required to define a realistic scenario of covert channel use. This process helps remove any misunderstandings that might arise when different parties use different assumptions to define the environment set-up times for a channel.

4.3 *TCSEC* REQUIREMENTS AND RECOMMENDATIONS

The *TCSEC* requirements for bandwidth determination of covert channels state, "The system developer ... shall make a determination (either by actual measurements or by engineering estimation) of the maximum bandwidths of each identified channel."

As explained in Section 4.1, the measurements or estimation of the maximum bandwidth must assume that the covert channels are noiseless, that no processes—other than the sender and receiver—are present in the system when measurements are performed, and that the

synchronization time between senders and receivers is negligible. If the channel's bandwidth is estimated using informal methods, measurements of the channel's fastest primitives must be done to determine the values of T5(i), Tr(i), Tenv(j) as defined in Section 4.2.2, and the smallest measured value of T_{cs} must be chosen. For both formal and informal bandwidth determination methods, the selection of the TCB primitives measured should be based on realistic scenarios of channel use and should take into account any parameter of TCB state dependency that is relevant for a channel. The system configuration and architecture parameters should be specified for each set of measurements. All measurements necessary for bandwidth determination should be repeatable. Channel aggregation should be considered even though it is not supported by any TCSEC requirements or recommendations.

The *TCSEC* requirements for level A1 state, "Formal methods shall be used in the [covert channel] analysis."

In the context of bandwidth measurement or estimation, this requirement suggests that Millen's method (1989a)—defined and illustrated in Section 4.2.1 —should be used. Any other relevant information-theory-based method for covert channel bandwidth estimation could be acceptable on a case-by-case basis.

5.0 COVERT CHANNEL HANDLING

In this chapter we present three general methods for the handling of known covert channels that have been proposed and used to date. We also present a set of handling policies based on the analysis of the covert channel threats and risks that is consistent with the objective of the handling guideline of [NCSC TCSEC].

5.1 ELIMINATION OF COVERT CHANNELS

The first method is the *elimination* of covert channels. Elimination requires changing the design and/or implementation of a system to remove covert channels from the system. These changes include:

- The elimination of resource sharing between any potential participants in covert leakage of information by preallocating maximum resource demands to all participants or by partitioning resources on a per-security-level basis;
- The elimination of interfaces, features, and mechanisms which can cause covert leakage.

Example 13 - Elimination of Resource Sharing and Resource Partitioning

To illustrate elimination of covert channels by the elimination of resource sharing, let us reconsider Example 1 . The dynamic allocation/deallocation of objects triggers dynamic allocation of memory segments, which provides a resource-exhaustion channel. If the memory is statically partitioned on a per-process or per-security-level basis, the resource-exhaustion channel is eliminated. However, as pointed out in Example 1, this partitioning is not always practical. For example, if the memory (or any other TCB resource, such as internal tables) is partitioned, memory utilization may decrease because some partitions may not be as frequently used as others. This infrequent use may cause a significant degradation in performance. We can find examples of resource partitioning that do not impose undue performance degradation. For instance, the name space of the UNIX System V interprocess communication objects can be partitioned on a per-security-level basis without significant performance degradation.

Resource partitioning on a per-user, or a per-process, basis is not always possible (e.g., shared hardware resources, such as busses, cannot be partitioned on a per-user or per-process basis). However, the use of these resources can, in principle, be partitioned in time on a per-security-level basis. That is, processes running at the same time can share hardware resources only if the processes run at the same level. For example, in the case of the multiprocessor configurations presented in Example 4, and illustrated in Figure 2-7, a dual-mode process dispatcher can be implemented. In *normal mode*, the use of the CPUs is not partitioned. However, to eliminate the timing channels discussed in Example 4, the processes waiting for service in the "ready" queue(s) can be loaded in available CPUs during the same quantum only if they have the same security level. In this mode, called the *time-partitioned mode*, the timing channels caused by bus or memory locking by each memory reference become harmless. Trusted processes should be exempt from time-partitioned dispatching whenever it can be shown they do not exploit Covert channels. (Furthermore, threat analysis performed in the environment of system use may exempt other non-TCB applications from the time-partitioned mode of operation. This exemption is an accreditation-

policy matter and, thus, beyond the scope of this guide.)

The performance degradation that may be caused from the time-partitioned dispatching depends on the mix of processes ready to run at any instance. In some environments, where families of processes run concurrently (i.e., are coscheduled, we discuss in Section 2.2.3), the performance degradation will be minimized since all processes of a family run at the same security level. Performance degradation will be significant whenever all processes of the "ready" queue(s) run at different security levels because partitioned-mode dispatching will idle all but one CPU. The overall performance degradation can be mitigated whenever partitioned mode dispatching can be turned on/off selectively by security administrators. In Section 5.4 we discuss policy factors, such as threat analysis, required for such actions.

Example 4 - Elimination of TCB Interfaces, Features, or Mechanisms

In Example 2 we presented a covert channel caused by the UNIX interface convention of preventing the removal of nonempty directories. We argued that eliminating this UNIX convention may be impossible in practice due to user program reliance on the inability to delete nonempty directories. However, in other instances, the elimination of TCB interface conventions, features, or mechanisms causing Covert channels is possible. For example, programs can encode classified information by modulating the extent to which they use resources, which is reflected in different accounting (e.g., billing) information returned to users. One could remove this accounting channel by eliminating billing on a user-level basis (i.e., by imposing fixed uniform limits on the extent to which a resource could be used, such as fixed maximum CPU time, fixed maximum I/O time). [Lampson73] Alternatively, this channel can be eliminated by producing accounting information on a per-level basis. Neither alternative seems particularly troublesome, in practice.

5.2 BANDWIDTH LIMITATION

The second method of handling known covert channels is based on *bandwidth-limitation* policies. Such policies require the reduction of the maximum, or alternatively the average, bandwidth of any channel to a predefined acceptable limit. One can limit bandwidths by:

- Deliberately introducing noise into channels (e.g., using random allocation algorithms for shared resources such as indices in shared tables, disk areas, process identifiers; introducing extraneous processes that modify covert channel variables in random patterns); and
- Deliberately introducing delays in each TCB primitive of a real channel.

Example 15 - Introduction of Noise and Delays in Channels

The process identifier channel is an event-count channel that in most systems can have a bandwidth of 10 to 500 bits per second. This channel appears because most TCBs create a new process identifier by incrementing a process-identifier variable whenever a new process is created. Thus, a receiver process could detect whether the sender process transmitted a 0 or a 1 bit by determining whether the identifiers of two processes it creates are consecutive numbers. One can reduce the bandwidth of this channel by changing the process-identifier allocation algorithm of the TCB. That is, the TCB could allocate unused identifiers in the identifier space (pseudo) randomly in a nonmonotonic sequence. Depending upon the randomization characteristics of the allocation

algorithm, the bandwidth of the process-identifier channel can be reduced to negligible values. Similar considerations apply to the other allocation algorithms of object identifiers. Note that using random allocation of identifiers introduces negligible overhead and performance degradation in a TCB.

An additional example of noise introduction in covert channels is the notion of "fuzzy time" introduced in [Hu91]. Security kernels can constrain user processes to use only virtual time (i.e., time related only to a user's process activity but not to real time). [Lipner75] To ensure little correlation between real and virtual time by a user process (i.e., a receiver), the relationship between real and virtual time is randomized. This is the underlying principle of the notion of "fuzzy time." The randomization appears to degrade system performance very little (i.e., 5-6% on a VAX system [Hu91]). Thus, "fuzzy time" seems practical even in systems where performance degradation is a significant concern.

An alternative method of reducing channel bandwidths includes the deliberate introduction of spurious processes. That is, user-level processes are introduced in the system to perform random alteration of channel variables. As illustrated in Figure 2-5, processes Up, . . . ,Uq can introduce noise by altering a channel variable. Furthermore, these processes can introduce delays in channels by interposing themselves between the senders and receivers. Analysis presented in [Tsai and Gligor88] shows that the introduction of spurious processes can reduce up to about 75% of the bandwidth of typical channels. However, the introduction of spurious processes for bandwidth-degradation purposes may not be cost-free. Spurious processes tend to degrade system performance—not only channel bandwidth.

The deliberate introduction of delays in TCB primitives of real channels is typically used only for limiting the bandwidth of resource-exhaustion channels. The reason is that one can place delays in these channels in a way that does not degrade system performance until these channels are used. Resource-exhaustion channels make use of resource-exhaustion exception (error) returns to transmit zeros or ones. By placing delays within the return path of an exception, the channel bandwidth is reduced proportionally with the frequency of either the zeros or the ones in the code used by the channel users. In normal mode of TCB operation, however, performance is not degraded because resource-exhaustion exceptions are generally rare (unless channels are used).

It is generally advisable to introduce settable delays within TCB primitives, for two reasons. First, settable delays give system management the opportunity to determine the extent of performance degradation incurred by setting delays selectively on a per-channel basis. Second, whenever the same operating system is used on different hardware platforms, the delay values need to be changed to account for increased or decreased bandwidth for the same channel.

The placement of delays in TCB primitives can be a more complex task than it may first appear. Tradeoffs appear in the placement of delays in the TCB. On one hand, the placement of a delay in functions closer to the TCB interface (i.e., in high-level functions) offers the potential of minimizing the impact of the delay on the entire TCB. For each covert channel, each TCB primitive can be dealt with separately. Thus, one can choose a minimum delay value for each particular TCB primitive and covert channel variable. On the other hand, disadvantages of this delay placement strategy are:

- More coding is needed because for each covert channel, every TCB primitive of the channel would have to be delayed individually; Example 16 illustrates this concept.
- A minimum delay value may not be achievable for every covert channel because, sometimes, it is unclear from the perspective of high-level TCB functions what variables and other functions the low-level functions use. For example, when a user issues a "creat" call in UNIX, the setting of an error message ENFlLE, when returning from "copen, may be done either in code using the file structure or in code using the i-nodes. This concept is illustrated in Example 17. In this case, it is impossible to achieve minimum delay for both error situations simultaneously.

Delays could be added in low-level TCB functions common to many TCB primitives. This action is possible because, in practice, each resource within the TCB is managed by a few dedicated functions (resource managers). Thus, all user processes that make use of a resource share these low-level resource-management functions. Delays added to low-level functions will virtually delay all TCB primitives that could take advantage of the corresponding covert channel. However, the disadvantage of this approach is that the length of delay must be determined by the highest-bandwidth channel (e.g., by the fastest TCB primitive) using this lowbandwidth channel. Consequently, TCB primitives used by low-bandwidth channels (or slower TCB primitives that reference the same shared global variable) sometimes tend to be delayed unnecessarily.

Example 16 - Delay Placement in a Resource-Exhaustion Channel

For most covert channels, one must place delays in more than one location of the TCB code. The control-flow paths presented in this example refer to the resource-exhaustion channel provided by the variable "inode→i_flag" of UNIX. This example shows that multiple control paths, both through different TCB primitives of a channel and through the same TCB primitive, must be covered by delays.

Some Control-Flow Paths for the inode_i_flag Variable

creat:

(1) creat→copen→namei→access

(2) creat→copen→access

open:

(1) open→copen→namei→access

(2) open→copen→access

unlink:

(1) unlink

(2) unlink→namei→access

utime:

(1) utime→namei→access

(2) utime→access

chsize:

(1) chsize

This example shows that delay addition to a channel should be placed in low-level TCB functions shared by multiple control paths for the same channel. The low-level TCB functions that are common to all control paths of the inode→ i_flag channel include those of "access," "unlink," and "chsize."

Example 17 - Ambiguity in Delay Placement

The setting of a specific error message within a TCB primitive may correspond to the viewing of multiple covert channels. Thus, the highest-level functions of a TCB primitive cannot determine which channel is being used. Therefore, achieving minimum delay in high-level TCB functions is not always possible. The highest-level functions of TCB primitives where different channels can be distinguished should be found and used for placement of minimum delays. The primitive "creat" of UNIX, in which the ENFILE error is set for both the inode_space and file_table channels, illustrates this case.

File Table Channel

(1) creat→copen→faIloc

lnode Space Channel

(1) creat→copen→namei→iget

(2) creat→copen→mknod→ialloc→iget

In event-count channels, the addition of time delays is not advisable. These channels can be used in normal mode of TCB operation rather than in exception cases, and addition of delay would degrade performance significantly. Wherever possible, bandwidth limitation should be achieved by using a randomization algorithm for assigning the next available index or identifier. For TCB primitives that simply read, rather than allocate, indices and identifiers, use of delays may still be necessary whenever the randomization algorithm cannot introduce a sufficient amount of noise to achieve the target bandwidth limit. This situation may arise when the index or identifier range is too small for effective randomization. Example 18 il lustrates the selection of randomization points for the process identifier channel of UNIX (variable proc→p_pid of UNIX).

Example 18 - Randomization Points of the Process-Identifier Channel in UNIX

fork

(1) fork

wait

(1) wait

(2) wait→freeproc

(3) wait→sleep→issig→freeproc

getpid

(1) getpid

setpgrp

(1) setpgrp

In the process identifier channel of UNIX, a randomization algorithm should be used within the TCB functions listed below to assign the next available identifier. The invocation of identifier-reading TCB primitives, such as "getpid" and "setpgrp," could also be delayed unconditionally to help limit the channel bandwidth whenever the identifier randomization is inadequate (e.g., provides monotonically increasing identifiers in all cases).

5.3 AUDITING THE USE OF COVERT CHANNELS

The third method of handling *known* covert channels is that of *deterrence* of Co- vert channel use. This method allows all users to exploit known channels but provides a mechanism discouraging channel use. The main deterrence method is channel auditing. This method assumes audit mechanisms can unambiguously detect the use of a channel. Thus, users can be assured of detection of any unauthorized use of covert channels. Note, however, that the *TCSEC* requires only the *ability* to audit covert channels be provided—not that covert channels be actually audited. This detail limits somewhat the effectiveness of audit as a real deterrent.

Covert channel auditing requires that sufficient data be recorded in audit trails to enable the identification of (1) individual covert channel use, or use of certain channel types; and (2) identification of the senders and receivers of individual channels or of channel types (i.e., the identification of the covert channel users). Furthermore, discovery of covert channel use must be certain (i.e., 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.

Estimation of actual covert channel bandwidth is possible and desirable once covert channel use has been determined by audit-trail analysis. Note that, in genera, it is impossible to discover the actual information being leaked through covert channels 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 zeros or ones are the only recorded patterns one can unambiguously classify as noise.

Most of the problems identified in covert channel auditing are fundamental and are shared by most operating systems; these problems include (1) inability of distinguishing use of covert channels from innocuous use of TCB primitives, and (2) ambiguity in distinguishing senders from receivers among covert channel users. These problems appear because single TCB primitives may both alter and view a variable or attribute, depending on the argument values of that primitive and on the system state, and because different TCB primitives may be shared by different covert channels. Such primitives allow users to disguise covert channel use, thereby circumventing audit, and cause false detection of covert channels [Shieh and Gligor9]. Figures 3-2 and 3-4 show examples of such primitives.

Key design concerns of covert channel auditing are those of determining what events should be recorded by auditing mechanisms and what data should be maintained by auditing tools to ensure that all covert channel use can be discovered. The identification of covert channels can be summarized as sets of < variable, PA_h,PV_i > triples (where PA_h/PV_i represents a TCB primitive altering/viewing the variable, as shown in Figures 3-2 and 3-4), suggesting that recording all events including pairs of < PA_h, variable > and < PV_i, variable > is necessary and sufficient for covert channel auditing. However, recording such events is fraught with both practical and fundamental difficulties because audit-record formats and mechanisms currently used in practice include only process identifiers, object identifiers, process and object security levels, type of event (e.g., primitive identifier), and event outcome (i.e., success or error value); viz., Refs. [NCSC TCSEC, NCSC Audit]. Fields for recording covert channel variables are not included in existent audit-record formats. [Shieh and Gligor9o] provide examples of such fields and their setting.

In resource-exhaustion channels one can sometimes identify <PV_i, variable> pairs from recorded primitive identifiers and event outcomes. For example, whenever the event outcome is an error that can be unambiguously associated with a channel variable, the auditor can infer that the recorded primitive identifier represents PV_i. However, whenever the event outcome is no-error and if $PV_i = PA_h$, the auditor cannot tell whether the recorded primitive identifier is for a PV_i- or a PA_h-type of primitive; nor can the auditor tell whether the recorded primitive identifier represents a PAh type of primitive or an innocuous TCB primitive, whenever PV_i ! = PA_h occurs. Whether the use of a TCB primitive is innocuous or covert channel related depends on the state of the system and on the values of the primitive's parameters. Thus, the recording of channel variables is necessary for all no-error outcomes of a primitive associated with a covert channel.

Fundamental difficulties with recording channel variables appear because many TCB primitives are shared by several covert channels. Thus, a PA_h- or a PV_i-type primitive may refer to variables of multiple channels. The actual use of a single variable cannot be discerned even when all potential variables are known and even when the error outcomes of a PV; primitive can be unambiguously associated with single channel variables. For example, a user can infer a no-error outcome of a shared PV_i primitive on a given variable from an error outcome of the same PV_i primitive on another variable. This process enables users to disguise the use of a channel as transmission of noise (e.g., constant strings of 0s or 1s) on multiple channels (an example of reference [Shieh and Gligor90]). In such cases, auditors have to maintain additional information to enable the detection of all potential use of covert channels.

Examples of the storage channel auditing problems mentioned above have been illustrated in the context of the Secure Xenix system in reference [Shieh and Gligor90]. Solutions to these problems are also presented in that reference. We must note, however, that not all use of covert channels can be audited. Example 4 of section 2.2 illustrates a few instances of covert timing channels usage where auditing is impractical.

5.4 *TCSEC* REQUIREMENTS AND RECOMMENDATIONS

TCSEC requirements for covert channel handling are included in the audit and documentation requirements. Section 8 of the *TCSEC*, "A Guideline on Covert Channels," makes additional recommendations.

The audit requirements of the *TCSEC* state, "The TCB shall be able to audit the identified events that may be used in the exploitation of covert storage channels."

The design documentation requirements state:

> [Documentation] shall also present the results of the covert channel analysis and the trade-offs involved in restricting the channels. All auditable events that may be used in the exploitation of known covert storage channels shall be identified. The bandwidths of known covert channels, the use of which is not detectable by the auditing mechanisms, shall be provided.

The [NCSC TCSEC] guidelines on covert channels suggest the following combination of the above methods: (1) use elimination methods wherever possible to eliminate channels with bandwidths over 0.1 bits/second; (2) use bandwidth-limitation methods to reduce, whenever possible, the maximum bandwidth of every channel that cannot be eliminated to 1 bit/second or less; (3) use deterrence methods, namely audit, for channels with bandwidths over 0.1 bit/second; and (4) use a "don't care" policy for covert channels with bandwidths less than 0.1 bit/second.

The *TCSEC* requirements for handling covert channels and the covert channel guidelines presented in Section 8 of the *TCSEC* suggest the following handling policy:

- Covert channels with bandwidths under some predefined lower limit b are acceptable;
- Covert storage channels with bandwidths over lower limit b shall be auditable; the bandwidths of all storage channels that are not auditable shall be documented;
- Covert channels with bandwidths over some predefined upper limit B > b represent a significant threat and, wherever possible, they should either be eliminated or their bandwidth should be reduced to B bits/second; and
- Covert storage channels with bandwidths over b bits per second should be audited; this gives system administrators the ability to detect and procedurally correct significant compromise.

This policy allows for the existence of *storage channels* that are not auditable. Also, it allows for the possibility that covert storage and timing channels with bandwidths over B = 1 bit/second will exist in secure systems. However, the suggested values of b = 0.1 bits/second and B = 1 bit/second are not justified based on any specific policy. The only basis for deriving these values is the determination that:

- Covert channel handling may impose performance penalties, and that bandwidths of 1 bit/second are acceptable in most environments; and
- Although covert channels with bandwidth of over 1 bit/second may be allowed in a secure system, covert channels with bandwidths of over 100 bits/second approximate the rate at which many (old) computer terminals are run (or users can type). Therefore, the existence of such channels in a secure computer system would seem inappropriate.

(Note: This guide may not contain the current covert channel bandwidth policy, which is subject to change. Please contact the NCSC for information about the current policy.)

5.5 HANDLING POLICIES BASED ON THREAT ANALYSIS

Although the intent of the *TCSEC* handling requirements and guideline is sound, the justification of the particular values of bandwidth limits b (0.1 bits/second) and B (1 bit/second) may be less than satisfactory for the following reasons:

- The threat posed by covert channels depends on the specific application environment of use; therefore, the appropriateness of the bandwidth limits b and B cannot be determined without threat analyses within the specific application environment. Hence, these limits cannot be specified during the design or evaluation process. Whenever practical, a system should include variable covert channel delays whose values can be set by security administrators [IBM87].

- The threat posed by covert channels depends on the characteristics of the covert channels themselves. For example, (1) some covert channels have a maximum attainable value that may be very high but the noise and delay under normal system load decrease the attainable maximum bandwidth of these channels under an acceptable limit B; (2) some covert channels can be exploited more readily than others having simpler scenarios of use; (3) some covert channels cannot be audited because they appear at low system levels where audit is impractical; or (4) some covert channels can be aggregated serially or in parallel, increasing the effective bandwidth available to senders and receivers, and some others cannot be aggregated [Tsai and Gligor88].

These considerations indicate that the values of the bandwidth limits b and B can only be determined after a threat analysis which includes the above factors. Let us consider an example illustrating the necessity of threat analysis in the environment of secure system use.

Example 19 - Application Dependency of Bandwidth Limits

Consider an application environment in which classified satellite images are processed (e.g., satellite images of various agricultural crops in certain countries). Each image frame consists of 512x512 picture elements (pixels), each pixel having 8 bits, and each application includes up to 1 0,000 frames. A multilevel secure system is used which includes a covert channel of 10,000 bits/ second. This means an image frame can be declassified by using this covert channel in approximatively 200 seconds. Thus, up to 18 frames can be declassified in an unauthorized manner per hour. The need to operate this channel for more than one hour to declassify less than 0.2% of the data makes this threat negligible. The likelihood of detecting the use of this channel by (off-line) audit is very high due to its long period of operation. Thus, in this environment b can be set to 10,000 bits/second or even higher. Of course, information concerning the source of the satellite images may have a higher classification. The vulnerability of this information to covert channels may require its separate processing by trusted software rather than by untrusted application code.

In contrast, consider an application environment where 64-bit encryption keys are generated whose lifetimes are comparable with that of a login session (i.e., 8 hours). Even if these keys are encrypted when stored on nonvolatile storage, their actual use by application software would be in cleartext form. If the secure system used in this application contains a 0.1 bit/second channel, each session key can be declassified in less than 11 minutes, rendering the key vulnerable for most of its lifetime. The likelihood of being able to detect the use of this channel through off-line audit may not be very high because of the relatively short period of channel use. On-line audit of this channel

may be even less likely. Thus, in this application environment B could not be set to 0.1 bit/second. Instead, a B of 0.002 bit/second would seem more appropriate because, at that rate, it would take at least 8 hours to declassify a key.

The classification range of the information processed in a trusted system and, therefore, the trusted system class (i.e., B2-A1) must also be considered in threat analysis. Covert channels of high bandwidths (e.g., 1,000-10,000 bits per second) may be acceptable in a B2 system in which only Top Secret and Secret information is processed, and leakage below the Secret level is impossible. In contrast, the same leakage rate may be unacceptable in A1 systems that process multilevel information, since the possibility of unauthorized declassification of Top Secret information might be a real threat.

In threat analyses, one must also consider the characteristics of each covert channel. For example, the CPU scheduling channels of Example 3 may have a maximum bandwidth of 5-300 bits/second on systems comparable to today's fast workstations (depending on the operating system and scheduling parameters [Huskamp78]). However, compared with the upgraded directory channels, the CPU scheduling channels are much more difficult to use in any real system due to lack of control over scheduling parameters and due to noise introduced by background processes. Thus, these channels (and also those illustrated in Example 4 which use shared hardware resources) are significantly less likely to be used in practice than the noiseless upgraded-directory channels of Example 2. On the other hand, other noisy channels such as the various identifier channels may be more likely to be used than the upgraded directory channels because the likelihood of auditing correctly a noiseless channel is higher than that of auditing correctly a noisy channel. Thus, the high likelihood of detecting the use of the upgraded directory channel may deter its use.

This example indicates the need for establishing a threat-analysis policy on a per environment and system basis. It also suggests this analysis cannot be carried out at system evaluation time without postulating the characteristics of the application environment. Finally, this example suggests few of the important parameters that should be considered for such an analysis.

6.0 COVERT CHANNEL TESTING

6.1 TESTING REQUIREMENTS AND RECOMMENDATIONS

The *TCSEC* requirements of test documentation at class B2 state, ". . . [Test documentation] shall include results of testing the effectiveness of the methods used to reduce covert channel bandwidths."

Covert channel testing demonstrates that covert channel handling methods chosen by system designers work as intended. These methods include covert channel elimination, bandwidth limitation, and (ability to) audit. Testing is also useful to confirm that potential covert channels discovered in the system are in fact real channels. Furthermore, testing is useful when the handling method for covert channels uses variable bandwidth-reduction parameters (e.g., delays) that are settable by system administrators (e.g., by auditors).

Bandwidth estimation methods necessary for the handling of covert channels may be based on engineering estimation rather than on actual measurements. Bandwidth estimations provide upper bounds for covert channels before employing any handling methods. In contrast, covert channel testing always requires doing actual measurements to determine the covert channel bandwidths after implementing the chosen handling method in a system.

6.2 TEST DOCUMENTATION

Test plan documentation, including test conditions, test environment set-up, test data, expected test outcome, and actual test result documentation are discussed in the security testing guideline [NCSC Testing] in detail. Therefore, we do not repeat the discussion here. The security testing guideline also gives an example of the test plans for a real channel (i.e., for the upgraded-directory channel of Example 2).

7.0 SATISFYING THE *TCSEC* REQUIREMENTS FOR COVERT CHANNEL ANALYSIS

We present in this chapter the *TCSEC* requirements relevant to covert channel analysis and suggest ways to satisfy them. For each class containing them, we show the requirements of CCA (which include channel identification, bandwidth measurement or estimation, audit, and design documentation). We also summarize the recommendations made in the TCSEC guidelines on covert channels. Our recommendations, though derived from TCSEC objectives, are not requirements.

7.1 REQUIREMENTS FOR CLASS B2

7.1.1 Covert Channel Analysis

Channel Identification

The *TCSEC* requirement for CCA states, "The system developer shall conduct a thorough search for covert storage channels . . ."

Developers shall identify the sources of information used to satisfy this requirement. These sources shall include system reference manuals and the DTLS. They should include source code and processor specifications whenever the identification method includes source code and hardware analysis. Developers should show the identification method they use to be sound and reliable (e.g., repeatable). This implies, among other things, that independent evaluators can use the method on the same sources of covert channel information and get the same results. Otherwise, the identification evidence will lack credibility.

Bandwidth Measurement or Engineering Estimation

The *TCSEC* requirement for this area states, "The system developer shall . . . make a determination (either by actual measurement or by engineering estimation) of the maximum bandwidth of each identified channel."

In measuring or estimating covert channel bandwidth, developers should consider the following factors (as discussed in Section 4.1):

- For maximum bandwidth, assume the channel is noiseless and the presence of other processes in the system do not delay the senders and receivers.
- The choice of informal estimation methods requires defining (and possibly justifying) assumptions about the coding method and, therefore, the distribution of 0s and 1s in all transmissions. Whenever possible, use Millen's information-theory-based method, which yields the maximum bandwidth and also provides the required coding method to achieve it [Millen89a].
- Covert channel measurements should include the fastest primitives for altering, viewing, and setting up the transmission environment. Also, bandwidth measurements should involve the demonstrably fastest process (context) switch time.
- To determine bandwidth, derive the TCB primitives to measure from real scenarios of

covert channel use. Take into account parameter and TCB state dependencies of each selected primitive (if any).

- Specify the measurement environment. This specification includes (1) the speed of the system components, (2) the system configuration, (3) the sizes of the memory and cache components, and (4) the system initialization. Document the sensitivity of the measurement results to configuration changes. (This documentation enables accreditors to assess the real impact of covert channels in different environments of use.)
- Sender-receiver synchronization time may be considered negligible and, therefore, ignored.
- Consider channel aggregation in bandwidth estimation.
- All measurements must be repeatable.

7.1.2 Audit

The TCSEC Audit requirements state, "... The TCB shall be able to audit the identified events that may be used in the exploitation of covert storage channels."

To satisfy this requirement, audit mechanisms should include the following features whenever possible (viz., discussion of Section 5.3):

- The audit record should include the storage channel variables.
- The audit code of the TCB should cover all control paths leading to the alteration or viewing of the storage channel variables.
- The audit record should include sufficient information to identify unambiguously the senders and receivers of a storage channel.
- The audit mechanism should be noncircumventable whenever audit is turned on for a covert channel.

7.1.3 Design Documentation

Part of the *TCSEC* requirements for this area states the following:

> . . . This documentation shall also present the results of the covert channel analysis and the tradeoffs involved in restricting the channels. All auditable events that may be used in the exploitation of known covert storage channels shall be identified. The bandwidths of known covert storage channels, the use of which is not detectable by the auditing mechanism, shall be provided. . . .

Documentation of Identified Channels

The documentation of each identified storage channel should consist of the variable the channel views/alters and the TCB primitives that alter or view the variable. Developers should distinguish potential covert channels from real ones.

Documentation of Bandwidth Estimation

Developers must document measurements of each covert channel primitive and should include the bandwidth computation for each channel. They should document the measurement environment as specified in Section 7.1.1.

Documentation of Covert Channel Auditing

Documentation shall include a definition of each event used in the exploitation of a covert channel. This documentation should cite the definition of the TCB primitives and TCB paths leading to a covert channel variable. Developers should also identify and justify the covert storage channels that cannot be audited.

Channel Restriction and Handling Policies

The documentation shall include a description of (1) how covert channels are eliminated, and (2) how covert channel bandwidth is limited to a value deemed acceptable. Sections 5.1 and 5.2 provide a discussion of channel restriction methods.

Covert channel-handling policies should be consistent with the intent of the *TCSEC* guidelines. Covert channel bandwidth limits (b, B)—as defined in Sections 5.4 and 5.5—are considered outside the purview of the *TCSEC*. System accreditors should specify these limits for the specific threat environment the system will be used in.

7.1.4 Test Documentation

Part of the *TCSEC* requirements for test documentation states, ". . . [Test documentation] shall include results of testing the effectiveness of the methods used to reduce covert channel bandwidths."

See the security testing guideline [NCSC Testing], which discusses these requirements.

7.2 ADDITIONAL REQUIREMENTS FOR CLASS B3

Class B3 incorporates all requirements of class B2. In addition, the following requirements apply.

7.2.1 Covert Channel Analysis

Channel Identification

The only additional B3-class requirement is the identification of timing channels. Developers must define timing channel scenarios and identify all system components providing independent sources of timing (e.g., CPUs and I/O processors). Developers may use the same sources of information and methods for identifying timing channels as those used for identifying storage channels.

Bandwidth Measurement or Engineering Estimation

There are no additional requirements.

7.2.2 Audit

There are no additional requirements.

7.2.3 Design Documentation

Documentation of Identified Channels

The only additional requirement for this class is the documentation of all timing channels. Developers should document these channels by specifying the variable of the TCB state that may be changed by direct or indirect actions of user processes. These channels include CPU-scheduling channels, I/O-processor-scheduling channels, and page-replacement channels.

7.2.4 Test Documentation

There are no additional requirements.

7.3 ADDITIONAL REQUIREMENTS FOR CLASS A1

Class A1 contains all the class B3 requirements. The only additional requirements of class A1 appear in CCA.

Covert Channel Analysis

Part of the *TCSEC* requirement for this area states, ". . . Formal methods shall be used in analysis."

Channel Identification

Developers can apply formal methods on both formal specifications and source code of the TCB. These methods include syntactic information-flow analysis (with or without the use of semantic analysis) and noninterference analysis. Developers shall apply the chosen identification method to the FTLS. Unless the identification of covert channels is made a part of the specification-to-code correspondence (in which case source-code analysis is included), developers should complement the FTLS analysis with formal or informal source-code analysis. Otherwise, covert channels may remain undetected.

Bandwidth Measurement or Engineering Estimation

The requirement to use formal methods suggests that developers should use Millen's method (1989a)—defined and illustrated in Section 4.2.1. Any other information-theory-based method for covert channel bandwidth estimation may be acceptable on a case-by-case basis.

ACRONYMS AND ABBREVIATIONS

AIS	Automated Information System
CCA	Covert Channel Analysis
CPU	Central Processing Unit
DAC	Discretionary Access Control
DoD	Department of Defense
DTLS	Descriptive Top-level Specification
EHDM	Enhance Hierarchical Development Methodology
FDM	Formal Development Methodology
FTLS	Formal Top-level Specification
GVE	Gypsy Verification Environment
HDM	Hierarchical Development Methodology
I/O	Input/Output
IPC	InterProcess Communication
LOCK	Logical Co-processing Kernel
LRU	Least Recently Used
MAC	Mandatory Access Control
MLS	Multilevel Secure
NCSC	National Computer Security Center
SAT	Secure Ada Target
SRM	Shared Resource Matrix
TCB	Trusted Computing Base
TCSEC	Trusted Computer System Evaluation Criteria
TLS	Top-level Specification

GLOSSARY

ACCESS

Ability and means to communicate with (i.e., input to or receive output from) or otherwise make use of any information, resource, or component in an AIS. NOTE: An individual does not have "access" if the proper authority or a physical, technical, or procedural measure prevents them from obtaining knowledge or having an opportunity to alter information, material, resources, or components.

ACCESS TYPE

Privilege to perform an action on a program or file. NOTE: Read, write, execute, append, modify, delete, and create are examples of access types.

ACCREDITATION

Formal declaration by a designated approving authority that an AIS is approved to operate in a particular security mode using a prescribed set of safeguards.

ADMINISTRATIVE USER

A user assigned to supervise all or a portion of an AIS.

AUDIT

Independent review and examination of records and activities to assess the adequacy of system controls, to ensure compliance with established policies and operational procedures, and to recommend necessary changes in controls, policies, or procedures.

AUDIT MECHANISM

The processes used to collect, review, and/or examine system activities.

AUDITOR

An authorized individual, or role, with administrative duties, which include selecting the events to be audited on the system, setting up the audit parameters which enable the recording of those events, and analyzing the trail of audit events.

AUDIT TRAIL

Chronological record of system activities to enable the reconstruction and examination of the sequence of events and/or changes in an event.

BANDWIDTH

A characteristic of a communication channel that is the amount of information that can be passed through it in a given amount of time, usually expressed in bits per second.

BELL-LA PADULA SECURITY MODEL

Formal-state transition model of a computer security policy that describes a formal set of access controls based on information sensitivity and subject authorizations.

CATEGORY

Restrictive label that has been applied to both classified and unclassified data, thereby increasing the requirement for protection of, and restricting the access to, the data. NOTE: Examples include sensitive compartmented information, proprietary information, and North Atlantic Treaty Organization information. Individuals are granted access to special category information only after being granted formal access authorization.

CERTIFICATION

Comprehensive evaluation of the technical and nontechnical security features of an AlS and other safeguards, made in support of the accreditation process, to establish the extent to which a particular design and implementation meets a set of specified security requirements.

CHANNEL

An information transfer path within a system. May also refer to the mechanism by which the path is effected.

COVERT CHANNEL

Unintended and/or unauthorized communications path that can be used to transfer information in a manner that violates an AIS security policy. See also: Covert Storage Channel, Covert Timing Channel.

COVERT STORAGE CHANNEL

Covert channel that involves the direct or indirect writing to a storage location by one process and the direct or indirect reading of the storage location by another process. NOTE: Covert storage channels typically involve a finite resource (e.g., sectors on a disk) that is shared by two subjects at different security levels.

COVERT TIMING CHANNEL

Covert channel in which one process signals information to another process by modulating its own use of system resources (e.g., central processing unit time) in such a way that this manipulation affects the real response time observed by the second process.

DATA

Information with a specific physical representation.

DATA INTEGRITY

Condition that exists when data is unchanged from its source and has not been accidentally or maliciously modified, altered, or destroyed.

DESCRIPTIVE TOP-LEVEL SPECIFICATION (DTLS)

Top-level specification that is written in a natural language (e.g., English), an informal program design notation, or a combination of the two. NOTE: Descriptive top-level specification, required for a class B2 and B3 AIS, completely and accurately describes a trusted computing base. See formal top-level specification.

DISCRETIONARY ACCESS CONTROL

Means of restricting access to objects based on the identity and need-to-know of users and/or groups to which the object belongs. NOTE: Controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (directly or indirectly) to any other subject. See mandatory access control.

DOMAIN

Unique context (e.g., access control parameters) in which a program is operating; in effect, the set of objects that a subject has the ability to access.

EXPLOITABLE CHANNEL

Covert channel that is intended to violate the security policy governing an AIS and is usable or detectable by subjects external to the trusted computing base. See covert channel.

FORMAL SECURITY POLICY MODEL

Mathematically precise statement of a security policy. NOTE: Such a model must define a secure state, an initial state, and how the model represents changes in state. The model must be shown to be secure by proving that the initial state is secure and that all possible subsequent states remain secure.

FORMAL TOP-LEVEL SPECIFICATION (FTLS)

Top-level specification that is written in a formal mathematical language to allow theorems, showing the correspondence of the system specification to its formal requirements, to be hypothesized and formally proven. NOTE: Formal top-level specification, required for a class A1 AIS, completely and accurately describes the trusted computing base. See descriptive top-level specification.

FUNCTIONAL TESTING

Segment of security testing in which advertised security mechanisms of an AIS are tested under operational conditions.

MANDATORY ACCESS CONTROL

Means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity. See discretionary access control.

MULTILEVEL DEVICE

Device that is trusted to properly maintain and separate data of different security levels.

OBJECT

Passive entity that contains or receives information. NOTE: Access to an object implies access to the information it contains. Examples of objects are: records, blocks, pages, segments, files, directories, directory trees and programs, as well as bits, bytes, words, fields, processors, video displays, keyboards, clocks, printers, and network nodes.

OVERT CHANNEL

Communications path within a computer system or network that is designed for the authorized transfer of data. See covert channel.

PROCESS

A program in execution. See domain and subject.

READ

Fundamental operation in an AIS that results only in the flow of information from an object to a subject. See access type.

READ ACCESS

Permission to read information in an AIS.

SECURITY ADMINISTRATOR

An administrative role (or user) responsible for the security of an AIS and having the authority to enforce the security safeguards on all others who have access to the AIS (with the possible exception of the auditor.) Also called system administrator.

SECURITY LEVEL

The combination of a hierarchical classification and a set of nonhierarchical categories that represents the sensitivity of information.

SECURITY POLICY

The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information.

SECURITY POLICY MODEL

An informal or formal presentation of a security policy enforced by the AIS. It must identify the set of rules and practices that regulate how an AIS manages, protects, and distributes sensitive information. See Bell-La Padula security model and formal security policy model.

SECURITY TESTING

Process to determine that an AIS protects data and maintains functionality as intended. NOTE: Security testing may reveal vulnerabilities beyond the scope of the AS design.

SUBJECT

Active entity in an AIS, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state. NOTE: Technically, a process/domain pair.

SUBJECT SECURITY LEVEL

Sensitivity label(s) of the objects to which the subject has both read and write access. NOTE: Security level of a subject must always be dominated by the clearance level of the user with which the subject is associated.

TCB PRIMITIVE

An operation implemented by the TCB whose interface specifications (e.g., names, parameters, effects, exceptions, access control checks, errors, and calling conventions) are provided by system reference manuals or DTLS/FTLS as required.

TOP-LEVEL SPECIFICATION (TLS)

A nonprocedural description of system behavior at the most abstract level; typically, a functional specification that omits all implementation details.

TROJAN HORSE

Computer program containing an apparent or actual useful function that contains additional (hidden) functions that allows unauthorized collection, falsification or destruction of data.

TRUSTED COMPUTING BASE (TCB)

Totality of protection mechanisms within a computer system, including hardware, firmer, and software, the combination of which is responsible for enforcing a security policy. NOTE: The ability of a trusted computing base to enforce correctly a unified

security policy depends on the correctness of the mechanisms within the trusted computing base, the protection of those mechanisms to ensure their correctness, and the correct input of parameters related to the security policy.

USER

Person or process accessing an AIS by direct connections (e.g., via terminals) or indirect connections. NOTE: "Indirect connection" relates to persons who prepare input data or receive output that is not reviewed for content or classification by a responsible individual.

VERIFICATION

The process of comparing two levels of an AIS specification for proper correspondence (e.g., security policy model with top-level specification, top-level specification with source code, or source code with object code). NOTE: This process may or may not be automated.

WRITE

Fundamental operation in an AIS that results only in the flow of information from a subject to an object. See access type.

WRITE ACCESS

Permission to write to an object in an AIS.

REFERENCES

[Andrews and Reitman80]

G. R. Andrews and R. P. Reitman, "**An Axiomatic Approach to Information Flow in Programs**," *ACM Transactions on Programming Languages and Systems*, 2:1, pp. 56-76, January 1980.

[Bach86]

M. J. Bach, ***The Design of the UNIX Operating System***, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1986.

[Bell and La Padula76]

D. E. Bell and L. J. La Padula, ***Secure Computer System: Unified Exposition and Multics Interpretation***, The MITRE Corp., Report No. MTR-2997 Revision 1, Electronic Systems Division, U. S. Air Force Systems Command, Technical Report ESD-TR-75-306, Bedford, Massachusetts, March 1976 (available as NTlS ADA023588).

[Benzel84]

T. V. Benzel, "**Analysis of a Kernel Verification**," *Proceedings of the lEEE Symposium on Security and Privacy*, Oakland, California, pp. 125-131, April 1984.

[Biba77]

K. J. Biba, ***Integrity Considerations for Secure Computer Systems***, The MlTRE Corp., Report No. MTR-3153 Revision 1, Electronic Systems Division, U. S. Air Force Systems Command, Technical Report ESD-TR-76-372, Bedford, Massachusetts, April 1977.

[Boebert85]

W. E. Boebert, R. Y. Kain, and W. D. Young, "**Secure Computing: The Secure Ada Target Approach**," *Scientific Honeyweller*, 6:2, pp. 1-17, July 1985.

[Cipher90]

IEEE Computer Society Technical Committee on Security and Privacy, ***Minutes of the First Workshop on Covert Channel Analysis***, Cipher Newsletter, Special Issue, pp. 8-12, July 1990.

[Clark and Wilson87]

D. D. Clark and D. R. Wilson, "**A Comparison of Commercial and Military Computer Security Policies**," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, pp. 184-194, April 1987.

[Denning76]

D. E. Denning, "**A Lattice Model of Secure Information Flow**," *Communications of the ACM*, 19:5, pp. 236-243, May 1976.

[Denning77]

D. E. Denning and P. J. Denning, "**Certification of Programs for Secure Information Flow**," *Communications of the ACM*, 20:7, pp. 504-513, July 1977.

[Denning83]

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

[DoD Directive]

Department of Defense, ***Security Requirements for Automated Information Systems (AISs)***, DoD Directive 5200.28, 21 March 1988.

[Eckmann87]

S. T. Eckmann, "**Ina FIo: The FDM Flow Tool**," *Proceedings of the 10th National Computer Security Conference*, Baltimore, Maryland, pp. 175-182, September 1987.

[Feiertag80]

R. Feiertag, ***A Technique for Proving Specifications are Multilevel Secure***, Technical Report CSL-109, Computer Science Laboratory, SRI International, Menlo Park, California, January 1980.

[Gallager68]

R. G. Gallager, ***Information Theory and Reliable Communications***, John Wiley and Sons, New York, 1968.

[Gasser88]

M. Gasser, ***Building A Secure Computer System***, Van Nostrand Reinhold, New York, 1988.

[Gligor86]

V. D. Gligor and C. S. Chandersekaran, "**Towards the Development of Secure Distributed Systems**," in Grissonnanche, A. (editor), *Information Security: The Challenge*, IFIP Press, Monte Carlo, pp. 395-406,1986.

[Gligor87]

V. D. Gligor, C. S. Chandersekaran, R. S. Chapman, L. J. Dotterer, M. S. Hecht, W.-D. Jiang, A. Johri, G. L. Luckenbaugh, N. Vasudevan, "**Design and Implementation of Secure Xenix**," *IEEE Transactions on Software Engineering*, 13:2, pp. 208-221, February 1987.

[Goguen and Meseguer82]

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

[Goguen and Meseguer84]

J. A. Goguen and J. Meseguer, "**Unwinding and Inference Control**," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, pp. 75-86, April 1984.

[Haberman72]

A. N. Haberman, "**Synchronization of Communicating Processes**," *Communications of the ACM*, 12:7, pp. 171-176, July 1972.

[Haigh87]

J. T. Haigh, R. A. Kemmerer, J. McHugh, and W. D. Young, "**An Experience Using Two Covert Channel Analysis Techniques on a Real System Design**," *IEEE Transactions on Software Engineering*, 13:2, pp. 157-168, February 1987.

[Haykin83]

S. Haykin, ***Communication Systems***, John Wiley and Sons, New York, 1983 (second edition).

[He and Gligor90]

J. Ne and V. D. Gligor, "**Information Flow Analysis for Covert-Channel Identification in Multilevel Secure Operating Systems**," *Proceedings of the 3rd IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, pp. 139-148, June 1990.

[Honeywell85a]

Honeywell Information Systems Inc., ***SCOMP Interpretation of the Bell-La Padula Model***, Technical Report, McLean, Virginia, March 1985.

[Honeywell85b]

Honeywell Information Systems Inc., ***Multics Security Model - Bell and La Padula***, Multics Design Document MOD-002, Cambridge, Massachusetts, August 1985.

[Hu91]

W.-M. Hu, "**Reducing Timing Channels with Fuzzy Time**," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California, pp. 8-20, May 1991.

[Huskamp78]

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

[IBM87]

IBM Corp., ***Secure Xenix, version 1.1 - Descriptive Top-Level Specifications*****,** June 1987.

[Jones79]

A. K. Jones, R. K. Chansler Jr., I. Durham, K. Schwans, and S. R. Vegdahl, "**StarOS: A Multiprocessor Operating System for the Support Task Forces**," *Proceedings of the 7th Symposium on Operating System Principles*, Pacific Grove, California, pp. 117-127, December 1979.

[Karger87]

P. A. Karger, "**Limiting the Damage Potential of Discretionary Trojan Horses**,"

Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, pp. 32-37, April 1987.

[Karger and Wray91]

P. A. Karger and J. C. Wray, "**Storage Channels in Disk Arm Optimization**," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California, pp. 52-61, May 1991.

[Kemmerer83]

R. A. Kemmerer, "**Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels**," *ACM Transactions on Computer Systems*, 1:3, pp. 256-277, August 1983.

[Kemmerer86]

R. A. Kemmerer, ***Verification Assessment Study Final Report***, National Computer Security Center, Technical Report C3-CR01-86, Library No. S-228,204, March 1986.

[Lampson73]

B. W. Lampson, "**A Note on the Confinement Problem**," *Communications of the ACM*, 16:10, pp. 613-615, October 1973.

[Leach83]

P. J. Leach, P. H. Levine, B. P. Dorous, J. A. Hamilton, D. L. Nelson, and B. L. Stumpf, "**The Architecture of an Integrated Local Network**," *IEEE Journal on Selected Areas in Communications*, 1:5, pp. 842-856, November 1983.

[Lipner75]

S. B. Lipner, "**A Comment on the Confinement Problem**," *Operating Systems Review*, 9:5, pp. 192-196, November 1975.

[Loepere85]

K. Loepere, "**Resolving Covert Channels within a 82 Class Secure System**," *Operating Systems Review*, ACM SlGOPS, 19:3, pp. 9-28, July 1985.

[Luckenbaugh86]

G. L. Luckenbaugh, V. D. Gligor, L. J. Dotterer, C. S. Chandersekaran, and N. Vasudevan, "**Interpretation of the Bell-La Padula Model in Secure Xenix**," *Proceedings of the 9th National Computer Security Conference*, Gaithersburg, Maryland, pp. 113-125, September 1986.

[McHugh and Akers87]

J. McHugh and R. L. Akers, ***A Formal Justification for the Gypsy Information Flow Tool***, Technical Report, Computational Logic Inc., Austin, Texas, 1987.

[McHugh and Good85]

J. McHugh and D. I. Good, "**An Information Flow Tool for Gypsy**," *Proceedings of the*

IEEE Symposium on Security and Privacy, pp. 46-48, April 1985.

[MilIen76]

J. K. Millen, "**Security Kernel Validation in Practice**," *Communications of the ACM*, 19:5, May 1976.

[Millen78]

J. K. Mien, "**An Example of a Formal Flow Violation**," *Proceedings of the IEEE International Conference on Computer Soltware and Applications*, Chicago, Illinois, pp. 204-208, 1978.

[Millen81]

J. K. Mien, "**Information Flow Analysis of Formal Specifications**" *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 3-8, April 1981.

[Millen89a]

J. K. Millen, "**Finite-State Noiseless Covert Channels**," *Proceedings of the Computer Security Foundations Workshop*, Franconia, New Hampshire, pp. 81-85, June 1989.

[Millen89b]

J. K. Millen, **F*oundations of Covert-Channel Detection***, The MlTRE Corporation, Technical Report MTR-10538, January 1989.

[NCSC Audit]

National Computer Security Center, ***A Guide to Understanding Audit in Trusted Systems***, NCSC-TG-001, version 2, June 1988.

[NCSC DAC]

National Computer Security Center, ***A Guide to Understanding Discretionary Access Control***, NCSC-TG-003 , version 1, 30 September 1987.

[NCSC TCSEC]

National Computer Security Center, ***Department of Defense Trusted Computer System Evaluation Criteria***, DoD 5200.28-STD, December 1985.

[NCSC Testing]

National Computer Security Center, ***A Guide to Understanding Security Testing and Test Documentation in Trusted Systems***, Draft, 1989.

[Osterhout80]

J. K. Osterhout, S. H. Scelza, and P. S. Sinhu, "**Medusa: An Experiment in Distributed Operating System Structure**," *Communications of the ACM,* 23:2, pp. 92-105, February 1980.

[Osterhout82]

J. K. Osterhout, "**Scheduling Techniques for Concurrent System Structure**," *Proceedings*

of the 3rd International Conference on Distributed Computing Systems, Ft. Lauderdale, Florida, pp. 22-30, October 1982.

[Reed and Kanodia78]

D. P. Reed and R. K. Kanodia, "**Synchronization with Eventcounts and Sequencers**," *Communications of the ACM*, 21:2, pp. 115-122, February 1978.

[Rushby84]

J. Rushby, "**The Security Model of Enhanced HDM**," *Proceedings of the 7th DOD/NBS Computer Security Conference*, pp. 120-136, September 1984.

[Rushby85]

J. Rushby, ***The SRI Security Model***, Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, April 1985.

[Saltzer and Schroeder75]

J. H. Saltzer and M. D. Schroeder, "**The Protection of Information in Computer Systems**," *Proceedings of the IEEE*, 63:9, pp. 1278-1308, September 1975.

[Schaefer77]

M. Schaefer, B. Gold, R. Linde, and J. Scheid, "**Program Confinement in KVM/370**," *Proceedings of the 1977 Annual ACM Conference*, Seattle, Washington, ACM, New York, pp. 404-410, October 1977.

[Schaefer89]

M. Schaefer, "**Symbol Security Condition Considered Harmful**," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, pp. 20-46, April 1989.

[Schroeder77]

M. D. Schroeder, D. D. Clark, and J. H. Saltzer, "**The Multics Kernel Design Project**," *Proceedings of the 6th ACM Symposium on Operating Systems Principles*, West Lafayette, IN, pp. 43—45, November 1977. (Also available in the *Operating Systems Review*, 11:5, November 1977.)

[Shannon and Weaver64]

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

[Shieh and Gligor90]

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

[Tsai90]

C.-R. Tsai, V. D. Gligor, and C. S Chandersekaran, "**A Formal Method for the**

Identification of Covert Storage Channels in Source Code," *lEEE Transactions on Software Engineering*, 16:6, pp. 569-580, June 1990. (Also in the *Proceedings of the lEEE Symposium on Security and Privacy*, Oakland, California, pp. 74-86, April 1987.)

[Tsai and Gligor88]

C.-R. Tsai and V. D. Gligor, "**A Bandwidth Computation Model for Covert Storage Channels and Its Applications**," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, pp. 108-121, April 1988.

[Walker83]

B. Walker, G. Popek, and G. Thiel, "**The LOCUS Distributed Operating System**," *Proceedings of the 9th ACM Symposium on Operating System Principles*, Bretton-Wood, New Hampshire, pp. 49-70, December 1983.

[Walter74]

K. G. Walter, W. F. Ogden, W. C. Rounds, F. T. Bradshaw, S. R. Ames, K. J. Biba, J. M. Gilligan, D. D. Schaeffer, S. I. Schaen, and D. G. Shumway, ***Modeling the Security Interface***, Technical Report No. 1158, Case Western Reserve University, Cleveland, Ohio, August 1974.

[Whitmore73]

J. Whitmore, et al., ***Design for Multics Security Enhancements***, Honeywell Information Systems Inc., Technical Report ESD-TR-74-1 76, HQ Electronic Systems Division, Hanscom AFB, Massachusetts, December 1973.

[Wray91]

J. C. Wray, "**An Analysis of Covert Timing Channels**," *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, California, pp. 2-7, May 1991.

APPENDIX A
ADDITIONAL EXAMPLES OF COVERT CHANNELS

This appendix provides additional examples of storage and timing channels. For simplicity, in all covert channel examples below we assume the security level of the sending process S dominates that of the receiving process R. However, one can find similar examples where the security levels of S and R are incomparable using the dominance relation defined in the system.

A.1 STORAGE CHANNELS

The following examples of covert storage channels are identified in the literature; for example, see [Lopere85] or [Tsai and Gligor88]. These examples are necessarily genric in nature and are meant to be a starting point for identification of covert storage channels in specific systems.

A.1.1 Table-Space Exhaustion Channels

The table-space exhaustion channel is similar to the memory-resource exhaustion channel. The channel is present on systems in which the TCB allocates a fixed amount of space for its table rather than employing some type of dynamic allocation scheme. For the purpose of creating an example here, suppose that a TCB has allocated a fixed table to keep track of processes created in the system. A sending process S whose security level dominates that of a receiving process R would transmit information in the following manner:

(1) Process R creates processes (one or more of these processes can be deleted by the sender S) until the process table space is exhausted. The new processes simply sleep or block indefinitely.

(2) Process R then synchronizes with process S.

(3) To send a 1 or 0, process S either deletes a process (in common with R) or doesn't and then blocks.

(4) Process R attempts to create another process (again, in common with S). It records a 1 or 0 depending on its success. Process R then synchronized with process S, and the operation continues in this fashion until all of the desired information is transferred.

A.1.2 Unmount of Busy File System Channels

This covert storage channel is exploitable in a segmented memory architecture system in which the file system cannot be unmounted if some segments are still in the address space of a process. If a process attempts to unmount a file system and is given an answer busy or not busy, a higher security level process owning segments contained in the file system can transfer information to a lower security level process attempting the unmount. The scenario is as follows:

(1) Process R (whose security level is dominated by that of S) begins by releasing all of the segments on the file system. Processes R and S have one segment in common which they can both map into their address spaces.

(2) Process R then synchronizes with process S.

(3) To send a 1 or 0, process S either maps the common segment into its address space (or does nothing if it is already there) or unmaps the common segment from its address space (or does nothing if it is already not there). Process S then blocks.

(4) Process R receives a 1 or 0 by attempting to unmount the file system and reviewing the result. If the unmount was successful, R remounts the file system.

(5) Process R then synchronized with process S and the exchange continues until completion of the desired transfer.

A.1.3 Printer Attachment Channel

When physical printers or other I/O devices are shared resources in a system, a sending process S at a high security level could potentially transfer information to a receiving process R at a lower security level by creating contention for the device(s). As always, the sender and receiver must have some way to synchronize. To send a 1, the sender process S simply checks to see if the printer is attached, and attaches it if it is not. To send a 0, the sender process S checks to see if the printer is attached, and detaches it if it is. The receiver process R attempts to attach the printer, receiving a 0 if successful, and a 1 otherwise. The receiver process R then detaches the printer if the attach call was successful.

A.2 TIMING CHANNELS

One way to think of the difference between covert timing channels and covert storage channels is that covert timing channels are essentially memoryless, whereas covert storage channels are not. With a timing channel, the information transmitted from the sender must be sensed by the receiver immediately, or it will be lost. However, an error code indicating a full disk which is exploited to create a storage channel may stay constant for an indefinite amount of time, so a receiving process is not as constrained by time.

As is the case with covert storage channels, covert timing channels will inevitably be present on any system in which sharing of system resources takes place. We present several examples of covert timing channels in the following sections.

The tasks of identification and handling covert timing channels (i.e., bandwidth reduction, elimination, or audit) in a secure system are more difficult than for covert storage channels for the following reasons:

(1) An accepted method (such as the SRM method presented in [Kemmerer83] and reviewed in Section 3.2.3) for identifying all covert timing channels does not exist. Although the SRM method has been presented as a tool for the identification of both covert storage and timing channels, in the case of timing channels it is no more effective than an ad hoc examination of each system call.

(2) Tools (such as Ina Flo or HDM MLS-presented in Appendix B) for identification covert timing channels do not exist. This is largely because the formal specification methodologies that have been developed do not address timing issues, and this situation is not expected to change in the near future [Haigh87]. This is the case because none of the existent tools can help discover timing channel scenarios.

(3) Covert timing channels involve the exploitation of normal system activity (and sometimes the direct exploitation of hardware), thus they are much more difficult to audit than covert storage channels. Attempts to perform meaningful audit of normal system activity will result in the generation of massive amounts of unusable data, added kernel complexity, and reduced performance. For example, it is practically impossible to audit the CPU timing channels of Example 3 and the bus, memory-port, and crossbar-switch contention channels of Example 4, Section 2.2.1.

Despite the problems listed above, to a certain extent one can design a secure system that limits or eliminates some types of covert timing channels. For example, one can eliminate a potential covert timing channel by time-partitioning a shared resource on a security-level basis (see Section 5.1). The addition of noise (for example, in the form of added processes) of added process) can also reduce the bandwidth of most covert timing channels.

The following sections enumerate some of the covert timing channels that are identified in the literature (for example, see [Schaefer77, Huskamp78, Karger and Wray91]). These examples are necessarily generic in nature and are meant to be a starting point for identification of covert timing channels in specific systems. We make two basic assumptions for all the examples detailed in the following sections. First, in each scenario we assume the communicating processes either have a continuous clock available that has reasonable resolution, or the processes create a time reference by using simple counters in memory segments or files. Second, we assume the communicating processes are running alone the system (i.e., little or no noise is present). This second assumption ensures calculating the maximum channels bandwidths.

These two assumptions help identify some of the countermeasures that can be used against some covert timing channels ina secure system. The first is to virtualize the clock in the system by resetting the clock at every context switch [Lipner75]. However, this action could render the system useless, since many system functions depend on a real, continuous time source. Also, this countermeasure is ineffective when the communicating processes have constructed their own time references. The other countermeasure that applies to most covert timing channels is the addition of noise to the system in the form of added processes. While the countermeasure can effectively reduce the bandwidth of the channel, it adds unwanted overhead to the system.

A.2.1. I/O Scheduling Channels

Consider a movable head I/O device that uses a scheduling algorithm called the "elevator algorithm." The algorithm works as follows: requests are enqueued by ascending cylinder number. Requests are then dequeued in order of ascending cylinder number until no greater cylinder number remains (i.e., the upper end of the cylinder is reached) and then are dequeued in descending order until no lower numbered cylinders remain (i.e., the lower end of the cylinder is reached). This process is continuously repeated.

Again, let process R be a receiver at a low security level and process S be the sender at a higher security level. Process R is the owner of cylinders 51 through 59 of a disk, to which S has read access. Process R issues a request for cylinder 55, waits until notified of its completion, and then relinquishes the CPU. Process S then issues a request for either cylinder 53 (to send a 0) or 57 (to send a 1), and then relinquishes the CPU. Process R then issues requests for cylinders 58 and 52, recording a 1 or 0 depending on which request completes first.

Note that similar timing channels can be found for other I/O scheduling algorithms. One way to eliminate these channels is to keep a process from viewing its requests until the entire queue of requests has been completed. This action also leads to underutilization of the i/O processing power.

A.2.2 I/O Operation Completion Channels

Assume process S and process R own separate portions of the same movable-head I/O device. Process S and process R synchronize themselves to run alternately by using the system's synchronization facilities. For an example of the synchronization primitives that could be used, see [Reed and Kanodia78]. T send a 1, process S requests a read on a part of the disk that is farthest from process R's portion of the disk. To send a 0, process S does nothing. Process R issues a read to its portion of the disk and determines the bit of information received depending on the time that it takes for the request to complete.

Note that both the I/O scheduling and the I/O completion channels are similar to the CPU quantum and CPU interquantum channels of Example3, Section 2.2.1.

A.2.3 Memory Resource Management Channels

Below we describe two possible covert timing channels associated with the activity of memory resource management. Both involve a sending process conveying information by modulating the frequency with which a receiver process obtains a resource. The first channel involves the ability of the sender to modulate the data paging rate; the second involves the sender modulating the time the receiver takes to obtain addressability to a segment via an active segment table.

A.2.3.1 Data Page Pool Channels

Suppose a sender process S whose security level dominates that of a receiver process R can request the same page of data in a demand-paging environment. In this scenario, the receiver process R's security level must dominate the security level of the page, and the processes S and R must possess "read" access to the page. The sender process S may not possess "write" access to the page. Imagine also that the page has not been referenced, and so is not resident in main memory. Additionally, the processes S and R must agree on a set of pages to be used to exploit this covert channel. Process S can now modulate process R's response time (and thus send a bit of information to R) when reading the page by either referencing the page or not (bringing the page into memory or not). If process S is reading (or not reading) from a new page each time, it can continually send information to R. Since memory is not infinite, a new page cannot be referenced each time. A potentially continuous channel still exists, though, as long as process S has more pages available to it than the memory management working set size, the pages are referenced ina circular fashion, and the memory manager is using a Least Recently Used (LRU) page-replacement algorithm.

A.2.3.2 Active Segment Table Channels

The active segment table channel is very similar to the data page pool channel, and is only a threat in a segmented architecture. The sending process S either introduces a new segment into its address space (resulting in the segment being entered into the active segment table) or doesn't, depending on the desired value to be transmitted. The receiving process R introduces the same segment into its address space, and perceives the difference in response time, which varies depending upon whether or not the entry is already in the active segment table. Similar to the data page pool case, this channel can be made continuous as long as process S and process R share more segments than there are slots in the active segment table and process S releases the previous iteration's segment (removing it from the table) before proceeding to the next interation.

A.2.4 Device Controller Contention Channels

Consider a system in which multiple single-level devices, of perhaps different levels, are handled by the same controller, and I/O to each device is scheduled serially. Process S, which is writing to device A, can send information to process R (R is dominated by S), which is writing to device B, by varying the time that it spends doming I/O. Note that this channels's bandwidth can potentially be raised by transmitting more elaborate bit patterns than just 0 or 1 with the use of encoding techniques (by associating higher radix digits with different time intervals perceived by the receiver). For example, a perceived interval of 0 to x would indicate a value of 0, interval x to y would be a 1, and interval y to z would be a 2. Note, however, that more elaborate schemes for encoding of data can be more susceptible to the effects of noise.

A.2.5 Exclusive Use of Segments Channels

In come systems, a user is allowed to obtain exclusive use of a segment. If a process currently

has exclusive access to a segment, other requests for access to the segment are blocked until the segment becomes available. Therefore, a sending process S at a high security level could potentially gain exclusive access to a segment, and modulate the time it takes for a receiving process R at a lower security level to gain access to the segment. Note that, as in the previous section, the bandwidth of the channel could potentially be raised by the transmission of more elaborate bit patterns.

A.2.6 Synchronization Primitive Contention Channels

Consider a system where a central lock provides and controls process synchronization primitives. Contention for this central coordination can then be exploited to create a timing channel. The required configuration is similar to that of Example 4 of Section 2.2.1: three processors in system, one process per processor, with a clock process incrementing a counter ina shared segment. The receiver process R continuously reads the shared counter, attempts two synchronization primitive calls, and reads the counter again. The sender process S (whose security level dominates that of the receiver process R) transmits a bit of data by either making a synchronization call (thereby causing contention) or waiting for the amount of time it would take to make two synchronization call (thereby causing contention) or waiting for the amount of time it would take to make two synchronization calls (one with contention), in order to stay synchronized. The receiver process R discriminates binary data by comparing the two reads of the counter in the shared segment. The delay will be longer if the sender process S has created contention.

APPENDIX B

TOOLS FOR COVERT CHANNEL IDENTIFICATION

The *TCSEC* requires one use formal methods in the covert channel analysis of a system targeted for the A1 class. A number of tools exist, and are generally associated with a particular suite of tools such as the Formal Development Methodology (FDM), the Enhanced Hierarchical Development Methodology (EHDM), and the Gypsy Verification Environment (GVE). Although the emphasis has been on examining specifications written in a language such as Ina Jo (FDM), Revised Special (EHDM), or Gypsy (GVE), some work has been done on analysis of source code for covert channels using tools (see Section 3.2). The examination for covert channels involves looking at each variable referenced in the specification, and deciding where information flow is possible.

The goal of using these tools is to identify (with respect to a given policy) insecure flows, so that all such flows in a system can be reduced or eliminated. However, as discussed in Section 3.1, use of a flow tool on a specification does not guarantee that insecure flows do not exist in an implementation; rather, it guarantees that insecure flows do not exist in, and are not required by, the specification. For this reason, one must be careful concerning assertions made about the application of a given tool to a design. Also, since the tools developed to date are not designed to find timing channel scenarios, they are useful primarily in the identification of covert storage channels.

The information-flow tools described below are the FDM Multilevel Secure (MLS) and SRM tools, the Gypsy Flow Analyzer, and the EHDM MLS tool. Chapter 3 describes the use of formal methods in the identification of covert channels in source code.

B.1 FDM INA FLOW TOOL

The Ina Flo Tool [Eckmann87], a software tool to aid CCA, is part of the FDM developed at Unisys. Ina Flo is composed of two tools: MLS, which is similar to the HDM MLS tool, and another tool, which implements the Shared Resource Matrix approach [Kemmerer83].

B.1.1 MLS

The MLS tool of Ina Flo identifies flows in an Ina Jo specification by examining dependencies between variables and formal parameters of transforms from one state to the next. If x and y are variables or formal parameters of a transforms from one state to the next. If x and y are variables or formal parameters of a transform, and the new value of y depends on the old value of x, then information flows from x to y. MLS also includes the following rule for determining security: A flow is secure if and only if the label (security level) of variable y dominates the label of variable x (i.e., the *-property [Bell and La Padula76] is preserved). The user assigns labels to variables and defines a partial ordering on those labels (defines the dominates relation). MLS then generates a

list of conjectures (one list per transform) which, when proven, guarantee there are no storage channels in the specification. Conjectures that cannot be proven represent potential covert channels which must be handled with manual analysis.

Because Ina Jo specifications can be written ina nondeterministic manner, the dependencies between old and new values of variables can be difficult to determine. For this reason, the generation of information regarding nondeterministic flows is optional. A preprocessor called "PREMLS" is available that accepts an Ina for specification and produces a more deterministic version of the same specification.

B.1.2 SRM

The SRM tool of Ina Flo is an implementation of the SRM Method, and is intended to be used on specifications in which the MLS tool cannot be used. This would be the case when the specification does not contain complete security policy information. The SRM tool does not provide as much automated capability as the MLS tool. It simply accepts the Ina Jo specification and generates the corresponding shared resource matrix. Analysis of this matrix is then a manual procedure.

B.2 GYPSY FLOW ANALYZER

The Gypsy Flow Analyzer [McHugh and Good85] is an information-flow tool that is part of the GVE. The basis for this tool comes from the Gypsy optimizer, which contains code to identify "ghost" variables having no effect on the outputs of the program. These variables are found by conducting a detailed flow analysis of the Gypsy specification. The flow analysis involves generating the set of all paths through each routine and determining all contributors to the output of the routine. In addition to the identification of flows resulting from assignment statements, those resulting from control constructs and buffer operations are identified.

The application of a flow tool such as this to the problem of identification of convert channels in secure systems involves the assignment of labels to variables, the definition of a flow policy, and the definition of a partial ordering among different label values, much like is done with the MLS tool of the FDM. Note that the choice of the flow policy is at the discretion of the user of the flow tool. The entire process of using the Gypsy Flow Tool is summarized in [McHugh and Good85] as follows:

(1) Definition of the desired information flow policy expressed as a Gypsy theory (i.e., a set of Gypsy functions, constants, lemmas, and data types).

(2) Identification of the TCB interface set.

(3) Execution of the flow analysis for the TCB.

(4) Construction of information flow analogues for each routine in the TCB interface set using the results of the previous step.

(5) Creation of information flow policy specifications for the TCB interface routines and for literals or constants appearing in the flows to the parameters of the interface set.

(6) Generation of verification conditions for the flow abstraction routines.

(7) Proof of the verification conditions.

Difficulty in proof of the verification conditions indicates existence of flows that violate the stated flow policy. The source of these illegal flows must be pinpointed exactly and treated so that the proof step can be completed. Unfortunately, in a large program, locating illegal flows can be very difficult, and the Gypsy Flow Tool does not provide much help in this area.

B.3 EHDM MLS TOOL

The EHDM MLS Tool [Rushby84] is an information flow tool that is part of the Enhanced Hierarchical Development Methodology. It accepts as input a specification written in Revised Special and produces a set of theorems to be verified. The successful verification of these theorems asserts that the specification is multilevel secure as defined by the SRI model in [Rushby84].

A conceptually simple statement of the SRI model is that the information users can obtain from a system cannot be influenced by users whose security level is greater than theirs. The model assumes a lattice of security levels and a collection of users assigned a security level. If operations in the system possess an invocation level SL_1 then the operation is multilevel secure if:

(1) The value (result) returned to the user depends on objects whose security levels SL_2 satisfy $SL_2 <= SL_1$.

(2) The objects that acquire new values during the operation are at security level SL_2 such that $SL_1 <= SL_2$.

(3) If an object at security level SL_2 acquires a new value dependent on the value of an object at security level SL_1, then SL_2 must dominate SL_1.

The SRI model is conceptually similar to the Bell and La Padula model [Bell and La Padula76]. The MLS tool produces a set of theorems for each operation that correspond to the three conditions stated above. Violations of the model may appear in the generated output, and may indicate design flaws or covert channels.

Two concerns about the MLS tool are raised in [Kemmerer86]. One is that the volume of generated theorems will be very high. The other is that all of the theorems may not be provable in an automated way using the EHDM theorem prover, so the user may have to edit the PROOF module, adding lemmas that make the verification conditions provable.

An earlier version of the MLS flow tool (old HDM) was used in the SCOMP verification effort [Benzel84] to identify a number of covert channels. Isolation of the potential channels consisted of tracing unprovable verification conditions back to the system specification, and then tracing the lines of the specification back to the lines of code, using the results of the specification-to-code correspondence effort before undertaking the task of applying the MLS flow tool to the specification.

B.4 SOURCE-CODE ANALYSIS TOOL

The previous three sections discuss various formal tools that can be used for the identification of covert storage channels by examining the specification of a system. This section outlines the steps of a formal method that has been developed for the identification of covert storage channels by examination of the source code of a system. [Tsai90, He and Gligor90] The advantages of a formal source-code approach to the identification of covert channels are:

(1) All potential storage channels in the implementation examined are discovered.

(2) It avoids discovery of false illegal flows (a problem that appears with the use of the other tools discussed above).

(3) The method helps to determine whether the mandatory access control rules are implemented correctly.

(4) It helps determine the source code locations where audit code, delays, and randomization code should be placed for handling covert channels.

(5) The method is a fully automated search for potential covert storage channels. [He and Gligor90]

In addition to the advantages stated above, a code level examination for covert storage channels seems to be stronger than the specification level searches provided by FDM. GVE, and HDM, since a formal method has not been developed for showing the correspondence of the specification to the code. The code-level search for covert storage channels is theoretically similar to specification-level approaches and is conducted as follows:

(1) Select the set kernel primitives (TCB interface calls) to be analyzed.

(2) Determine the visibility/alterability of kernel variables when primitives are invoked.

(3) Apply the mandatory access control policy to the shared variables and kernel primitives to detect flows which are in violation.

For more details on this method of channel identification, we refer the reader to [Tsai90, He and Gligor90].

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1993	Final

4. TITLE AND SUBTITLE
A Guide to Understanding Covert Channel Analysis of Trusted Systems

5. FUNDING NUMBERS

6. AUTHOR(S)
Virgil Gligor

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Security Agency
Attn: I94 (Standards, Criteria, and Guidelines Division)
Ft. George G. Meade, MD 20755-6000

8. PERFORMING ORGANIZATION REPORT NUMBER
NCSC-TG-030

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Security Agency
Attn: I94 (Standards, Criteria, and Guidelines Division)
Ft. George G. Meade, MD 20755-6000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Library No. S-240,572

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document provides a set of good practices related to covert channel analysis of systems employed for processing classified and other sensitive information. It's written to help vendors and evaluators understand covert channel analysis requirements. It contains suggestions and recommendations derived from Trusted Computer System Evaluation Criteria (TCSEC) objectives but which aren't required by the TCSEC.

14. SUBJECT TERMS
Computer security; Trusted Computer System Evaluation Criteria (TCSEC); automated information system (AIS); covert channel analysis; operating systems.

15. NUMBER OF PAGES
129

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	