JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 15, 91-106 (1999)

Estimating and Measuring Covert Channel Bandwidth in Multilevel Secure Operating Systems

SHIUH-PYNG SHIEH
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, Taiwan 300, R.O.C.
E-mail: ssp@csie.nctu.edu.tw

Covert channels are illicit means of leaking sensitive or private information through system global variables that usually are not part of the interpretation of data objects in the security model. We discovered that some covert channels can be modeled as finite-state graphs while others cannot. By using various techniques given in the paper, multiple bits of information can be simultaneously transferred through single or multiple covert channels. We present methods to determine and estimate the maximum bandwidths of both finite-state and infinite-state channels, and give the problems and basic rules for their measurement.

Keywords: covert channel bandwidth, multilevel secure systems, system calls

1. INTRODUCTION

Covert channels present a serious risk to data security in computer systems and networks. These channels are illicit means of leaking sensitive or private information through system global variables that usually are not part of the interpretation of data objects in the security model [6]. As more and more systems link to one another through networks, security risks increase exponentially. It is, thus, essential for any multilevel secure operating systems and their applications to use formal methods of covert-channel analysis for a trusted computing base. A trusted computing base (TCB) is the part of the machine hardware and system software of a secure system that enforces the security policy of the model in the system and whose correctness must be verified. A TCB provides at its interface to users a set of primitives or system calls which users can invoke and, thus, make use of the system. These TCB primitives can be used to alter or view the system global variables.

Covert channels can be classified into two types: *storage* channels and *timing* channels [9]. Although fundamentally the same, storage channels and timing channels differ in the way that information is encoded. In a storage channel, there is a shared global variable in the system that acts as the medium for information transfer, where a user can potentially

Received October 19, 1996; accepted September 6, 1997
Communicated by Arbee L.P. Chen.
[1]XENIX(TM) is a registered trademark of Microsoft Inc. UNIX(TM) is a trademark of AT&T Laboratories. Secure XENIX(TM) was developed by IBM Federal Sector Division for B2-level evaluation and is now marketed as Trusted XENIX(TM) by Trusted Information Systems Inc. The work of this paper was done on Secure XENIX, an early version of Trusted XENIX.
[2]This work was supported in part by the National Science Council, Taiwan, under contract NSC-85-2622-E-009-006R.

change its value by invoking a TCB primitive, and another user can potentially view the change directly or indirectly. A timing channel requires the ability of co-operativeness to reference a real-time clock so that the receiver can detect a timing difference that can be used as the basis for encoding data for information transfer. We will focus on the covert storage channels in the paper. Unless specified, when we mention covert channels, we mean covert storage channels.

The National Computer Security Center (NCSC) has developed requirements for the information rate estimation of covert channels in multilevel secure systems at level B2 or above [1, 2]. A guideline including those requirements states that covert channels of less than one bit per second are usually considered acceptable while a rate of more than 100 bits per second is considered unacceptable. Because of the need to find maximum bandwidths of covert channels, many researches have been done in the area.

Tsai and Gligor [12] presented a Markov model to simulate the use of covert storage channels and to compute their maximum bandwidths under different system loads and program behaviors. The model was developed based on the assumption that the distributions of zeros and ones are equal because the maximum capacity of a symmetric discrete memoryless channel is achieved using input (0's and 1's) with equal probability [3]. When its transition times are approximately equal, it is reasonable for the finite-state channel to have uniform transition times, which are equal to the mean of its transition times. Thus, the effort required for estimation can be reduced.

In other cases where the difference between the time required to transmitting a bit of zero and that required for a bit of one through a covert channel is significant, the above assumption becomes inadequate. Other coding techniques has been proposed in an attempt to solve problem. In 1964, Shannon proposed a technique for computing the capacity of finite-state communication channels with non-uniform transition times [13]. Millen adapted this technique to finite-state covert channels that use individual channel variables [10]. However, we will prove in sections 3 and 4 that analysis on individual channel variables is not sufficient because (1) many channel variables are dependent, and (2) some channels cannot be modeled as finite-state channels.

This paper is organized as follows. In section 2, we explain the notion of covert channels. In section 3, we give the concepts of serial aggregation, parallel aggregation and dependent channel variables, and explain how to model finite-state channels and compute their bandwidths. In sections 4 and 5, we discuss covert channels that cannot be modeled as finite-state graphs as well as the important issues for covert channel bandwidth measurement. Finally, we give the conclusions.

2. NOTION OF COVERT CHANNELS

The notion of covert channels was originally introduced by Lampson in a discussion of the confinement problem as one kind of communication channel that a service program should be confined to use [8]. Later, Kemmerer proposed the shared resource matrix methodology [7], which focused primarily on the discovery of covert channels in a formal or descriptive top-level design specification of operating system kernels and trusted processes rather than in source code.

Kemmerer pointed out that the minimum conditions for the existence of a storage channel are as follows: (1) there must exist a global variable in the system to which both the

sending and the receiving processes have access; (2) there must be a way (usually by invoking system calls) in which the sending process can alter the value of the global variable, and the receiving process can detect or observe (view) any change in the global variable; and (3) there must be a way in which the sending and the receiving processes can synchronize their operations so that the events of information flow can happen. Denoting the global variable, the sending and the receiving processes by Var, S and R, respectively, a storage channel can be graphically represented as shown in Fig. 1.

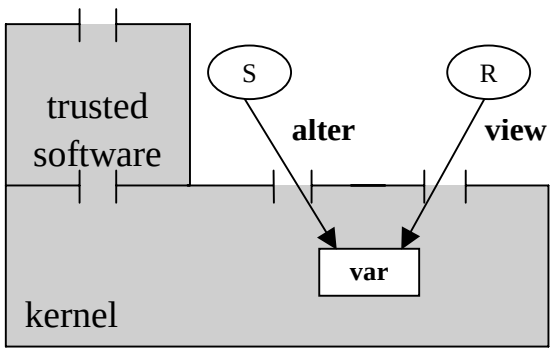


Fig. 1. Graphical representation of a storage channel.

The earliest work on covert-channel analysis of source code was that of Tsai, Gligor and Chandersekaran [11]. In their work, a formal method for the identification of potential covert channels was proposed. They provided a general but concise definition of covert channels in terms of security models and their interpretation in secure systems. That is, there is a potential covert channel between two subjects of a secure system if and only if (1) the two subjects have the potential to communicate with each other in the system and (2) such communication between the two corresponding subjects in the security model is not allowed under the non-discretionary security policy. The definition also points out the difference between *potential* covert channels and *real* covert channels that can actually be exploited by users for information transfer. That is, any communication channels that satisfy the two conditions above may not necessarily be real covert channels; real-time scenarios for actual use must be constructed for the potential covert channels.

The reasons why a potential covert channel may not necessarily be a real covert channel are that: (1) covert-channel analysis is usually performed statically and syntactically based on design specifications and implementation, and not dynamically at the time when the channel is actually being used; (2) in general, every covert channel has a condition associated with it which enables or disables information flows through the channel; and (3) real-time operation of the system may never make the condition become true and, thus, will never enable the channel of information flows. The task of determining whether a potential covert channel is a real channel involves creating a real-time scenario that can enable the condition. Potential covert channels that do not have such scenarios are not classified as real communication channels. Since the automated tool that has been designed and implemented performs static covert-channel analysis of the TCB source code

of multilevel secure operating systems, such as Trusted XENIX(TM), it can only identify the set of potential covert channels in these systems [5]. To detect the actual use of a real channel, the states of the corresponding global variable must be audited [14] .

There are basically two classes of covert storage channels: *resource exhaustion* and *event count*. Resource-exhaustion channels exist wherever system resources are shared among users at more than one security level. To use a resource-exhaustion channel, the sending process decides to exhaust or not exhaust a system resource, such as disk space, to encode a bit of 0 or 1. The receiving process detects the bit by trying to allocate the same system resource. Depending on whether it can allocate the resource, the receiving process can determine the value of the bit from the sending process. Basically, a resource-exhaustion channel can only transfer one bit in one transmission. In event-count channels, the sending process encodes multiple bits by requesting or not requesting a shared system resource (but not exhausting the resource). By querying the current status of the resource, either through some facility provided in the system, such as the system call *ustat* in UNIX (TM), or by observing the return result of some system call that allocates the resource, such as the system call *fork*, the receiving process can detect the bits from the sending process. Usually, the use of resource-exhaustion channel variables can be modeled as finite-state channels while that of event-count channel variables cannot. We will discuss this more in sections 2 and 3.

Some examples of resource-exhaustion and event-count channels in Trusted XENIX [4] are tabulated in Figs. 2 and 3. In these Fig.s, the left-most column is marked with system calls and the top row with shared global variables. Entries in the tables indicate

channel / TCB var / Primitive	inod space	file interlock	file table
open	AV	aV	AV
creat	AV	aV	AV
fork	a	a	aV
wait			
ustat			
chsize	a	aV	

Fig. 2. Examples of resource-exhaustion channels in trusted XENIX.

channel / TCB var / Primitive	Process ID	number of free inodes	number of free blocks
open		A	A
creat		A	A
fork	AV		
wait	a		
ustat		V	V
chsize		a	A

Fig. 3. Examples of event-count channels in trusted XENIX.

whether the system call can alter (*A* or *a*) and/or view (*V* or *v*) the corresponding global variable. A capital *A* in an entry indicates that the system can alter the global variable as a means of encoding and transferring information through the variable. Similarly, a capital *V* in an entry indicates that the system call can view the value of the global variable in order to detect the information from the sending process. Thus, for any shared global variable, the set of the system calls that have a capital *A* and those of those that have a capital *V* constitute the covert channels via the variable. For example, the system calls *creat* and *open* can be used by the sending and receiving processes to transfer information through the shared global variable representing the *file table* in the system. On the other hand, a small *a* in an entry means that, although the system call can alter the global variable, the alteration cannot be used to encode information in the global variable. For example, execution of the system call *fork* alters the file table because it increments the file reference count for the child process. This alteration, however,, is different from that of allocating a file table entry (allocating a entry requires increasing the file reference count from zero to one) because there is no way for the receiving process to determine whether the file reference count is increased; thus, this type of alteration does not comply with the semantics or scenarios for sending a bit of information. It represents a potential covert channel that is not real because the scenario for encoding information by executing *fork* simply does not exist. The same explanation applies to the entries that have a small *v*.

Some covert channels can be modeled as finite-state graphs, while others cannot. We call channels that cannot be modeled as finite-state graphs *infinite-state* channels. In general, resource-exhaustion channels can be modeled as finite-state channels while event-count channels can be modeled as infinite-state channels. We will discuss methods for bandwidth estimation in the following sections.

3. FINITE-STATE CHANNEL BANDWIDTH

The covert channel scenario defines how the sending process and receiving processes leak information. This definition may include a description of the synchronization methods used by the sender and the receiver, the creation and the initialization of the objects used by the sender/receiver to leak information, the initialization and resetting of the covert channel variable and so on. If channels are aggregated serially or in parallel, and if specific encodings are used, the aggregation and encoding methods can be defined.

To compute the maximum bandwidth of covert channels, we must consider both individual channels and aggregated channels. Storage channels can be parallel-aggregated, serial-aggregated, or a combination of both. Storage channels can be *serial-aggregated* if a process can invoke primitive operations to alter the channel variables without introducing additional context switches. When channels can be *serial*-aggregated, the aggregate bandwidth will be higher than that of any one of the individual channels. The improvement is significant when the sum of the time for alteration and viewing is much smaller than the context switch time [12].

Storage channels can be *parallel-aggregated* in a uniprocessor system whenever multiple shared covert channel variables can be altered or viewed by a single primitive operation. This means that a primitive may send or receive multiple bits through multiple channel variables. For example, in a single invocation, the primitive *creat* in Fig. 4 can simultaneously alter and view three channel variables: inode space, file interlock and file

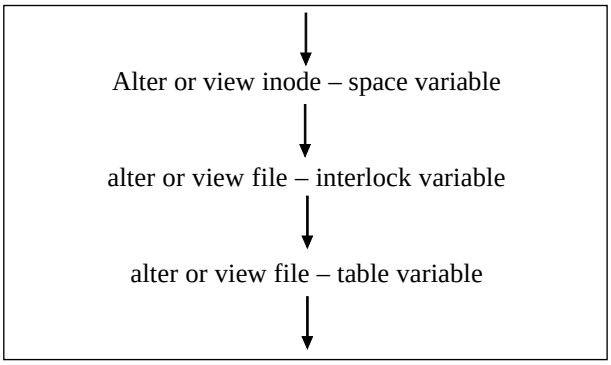


Fig. 4. Fragment of control flow of “creat”.

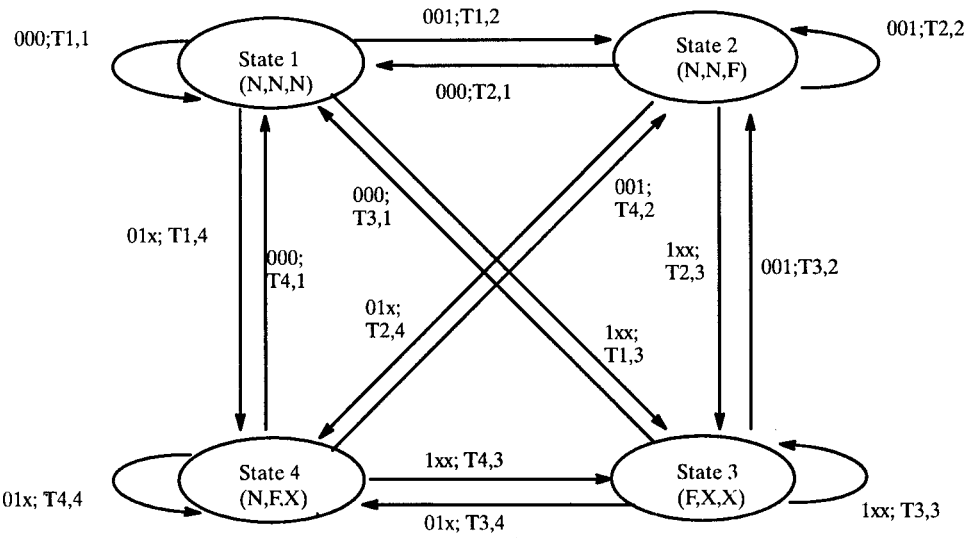
table. The information encoded in these shared covert channel variables can be aggregated to transmit additional information. It is clear that the maximum bandwidth of a parallel-aggregated channel cannot be greater than the sum of the bandwidths of each of the individual channels.

In most UNIX system cases, the shared global variables of a parallel-aggregated channel are *dependent*. A channel variable is dependent on another variable if alteration or viewing of the former cannot be performed unless that the latter is successfully completed. For example, the control flow of *creat* in Fig. 4 shows that *creat* cannot alter or view file-interlock or file-table variables unless it can alter or view inode-space variables successfully. Furthermore, when a variable is dependent on another variable, a no-exception of the latter can be inferred from the exception of the former. For example, when a file-table exception [ENFILE] is reported, the user can infer that neither the file-interlock exception [ETXTBSY] nor the inode-space exception [ENOSPC] prevail. Fig. 5 summarizes the relations. The left-most column of Fig. 5 includes error or no-error return of primitive *creat* while the top row with channel variables. A ‘0’ in an entry indicates that the user can infer a no-exception of a variable from an error or no-error return of the primitive *creat.* A ‘1’ in an entry indicates that the user receives an exception of a variable from an error return of the primitive creat. A ‘X’ in an entry indicates that the user cannot infer the value of a variable from an error return.

infer / return value:	inode space	file– interlock	file table
no – error return	0	0	0
error: file – table [ENFILE]	0	0	1
error: file – inter [ETXTBSY]	0	1	X
error : file – table [ENOSPC]	1	X	X

Fig. 5. Inference from an error and no – error return of “creat”.

As an example of simultaneous use of dependent covert channel variables, suppose that a single primitive can view channel variables **V1**, **V2** and **V3**. Also, suppose that the flow of control within this primitive is sequential; **V1** is accessed before **V2**, and **V2** is accessed before **V3**. The use of the three channels can be represented a four-state graph, as shown in Fig. 6. In this graph, nodes represent the states of the three variables; an edge indicates a state transition after an invocation of a viewing primitive, and the values associated with each edge are the coded message and elapsed time of this transition. Thus, the elapsed time T_{ij} for transition from state i to state j equals $2T_{cs} + T_{aij} + T_{vij} + T_{envij}$, where T_{cs} is the context switch time; T_{aij} is the elapse time of a altering primitive; T_{vij} is the elapse time of a viewing primitive; T_{envij} is the environment setup time; i and j vary from 1 to 4. When allocating the CPU to another process, the kernel performs a context switch from the current process to a new process. For example, during a transition from state (N, N, N) to state (N, F, X) in Fig. 6, a receiver receives the coded message 01x with elapsed time $T_{1,4}$.



Legend: (N, F, X): **V1** is non−full, **V2** is full, **V3** don't care
01x; T1,4: transition from state 1 to state 4 with the following effects:
V1 viewed with no−exception;
V2 is viewed with exception;
V3 is not viewed (don't care).

Fig. 6. A four-state graph for a parallel-aggregated covert channel using dependent variables V1, V2, V3.

The maximum transmission rate through a channel is defined as the channel capacity:

$$C = \lim_{t \to \infty} (\log_2 N_i(t)) / t , \quad (1)$$

where $N_i(t)$ is the number of possible symbol sequences of the total time t beginning at state i. In general, $N_i(t)$ obeys the difference equation:

$$N_i(t) = \sum_j N_j(t - T_{ij}), \tag{2}$$

where i and j vary from 1 to n, and $T_{ij} = 2T_{cs} + T_{aij} + T_{vij} + T_{envij}$ is the time taken by a transition from state i to j. To determine the capacity of a channel, it is only necessary to find the asymptotic upper limit of

$$N_i(t) = Aix^t.$$

Substituting this solution into Eq. 2, the system of equations is obtained:

$$A_i x^t = \sum_j A_j x^{t-T_{ij}},$$

This system of equations can be expressed in matrix form as $(P - I)A = 0$, where P is a matrix of the negative powers of x. Since $P - I$ is singular, its determinant $\mathrm{Det}(P - I) = 0$. Thus, Eq.1 implies

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

The channel capacity C is calculated from the largest root of the system of equations.

As the number of states n increases, the size of the $n \times n$ matrix $(P - I)$ increases by an order of n^2. Thus, solving n simultaneous equations becomes a complex task. To find a simple solution, the following assumptions are made: $\forall\, i, j, i, j = 1, ..., n$, $T_{aij} = T_a$, $T_{vij} = T_v$ and $T_{envij} = T_{env}$. Let $T = 2T_{cs} + T_a + T_v + T_{env}$; thus, $T_{ij} = 2T_{cs} + T_a + T_v + T_{env} = T$. Therefore, we have

$$Det(P - I) = \begin{vmatrix} x^{-T} - 1 & x^{-T} & \cdots & \cdots & x^{-T} \\ x^{-T} & x^{-T} - 1 & & & \vdots \\ x^{-T} & \ddots & & & \\ \vdots & & \ddots & & x^{-T} \\ x^{-T} & & & x^{-T} & x^{-T} - 1 \end{vmatrix} = (-1)^{n-1}(nx^{-T} - 1) = 0$$

$$\Rightarrow C = \log_2 x = \log_2 n/T$$

Because $n = N+1$, where N is the number of variables viewed by a primitive in which the flow of control is sequential, the maximum transmission rate of a parallel-aggregated, sequential-dependent channel is:

$$C = \log_2(N + 1)/(2T_{cs} + T_a + T_v + T_{env}).$$

Taking advantage of the dependent relations among channel variables shown in Fig. 5, we can encode messages as in Fig. 7. The original message 00, 01, 10 and 11 can be coded as the message 000, 001, 01x and 1xx. The transmission rate of the parallel-aggregated, dependent channel shown Fig. 7 is twice that of an individual channel. Intuitively, it can be concluded that the maximum transmission rate of a finite-state channel is affected significantly by two factors. First, the time for altering and viewing a channel variable should be minimized. Second, the altering (viewing) primitive should alter (view) as many variables as possible. On the other hand, finding the maximum bandwidth of a infinite-state channel is a different issue. We will discuss this in the next section.

Original Message	Coded Message
00	000
01	001
10	01
11	1

Fig. 7. Message encoded through a parallel – aggregated, dependent channel.

4. INFINITE-STATE CHANNEL BANDWIDTH

In general, event-count channels cannot be considered as finite-state channels. In an event-count channel, the sending process may change the value of the variable by zero, one or many to encode information. Because the change of the variable value may be any integer ranging from 0 to the system's limit, the event-count channels cannot be modeled as finite-state channels. We refer to them as infinite-state channels. The process-identifier event-count channel is one channel of this type. When a new process is created, the process identifier is generated by incrementing the previous process identifier. The sending process may create a child process and obtain a new process identifier. The receiving process may detect it by creating two successive processes to see if the two process identifiers are represented by consecutive numbers. The sending process makes several "*fork*" kernel calls, and the receiving process obtains the difference between the process identifiers (PIDs). The receiver can choose an appropriate threshold value for the PIDs to decode the information he has received from the sender. If the threshold is chosen appropriately, the transmission noise can be reduced to a negligible value. In an attempt to find the maximum bandwidth, it is appropriate to neglect the noise because the noise can only reduce the bandwidth.

We herein propose another encoding technique which can determine the maximum bandwidth of event-count, infinite-state channels. We take advantage of the multiple states to transfer multiple bits during each transmission. For example, the original messages which are three bits in size are converted to coded messages as shown in Fig. 8. A coded message can be transmitted in one instead of three transmissions. Intuitively, the more bits coded in one transmission, the higher the bandwidth we should get when these coded messages are equally distributed. However, the time needed for an infinite-state channel to encode and transmit n bits in one transmission, $T_{infinite\text{-}state}$, is equal to $(2^n - 1)T_{increment} + T_r + 2T_{CS} + T_{env}$, where $T_{increment}$ is the time required to increment the variable. The sending

Original Message	Coded Message
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Fig. 8. Message encoded through an infinite – sate channel "process – ID".

time, $(2^n - 1)T_{increment}$, may increase exponentially proportional to the number of bits coded and transmitted. Thus, we should have a optimal solution. Let us compare the bandwidth of an infinite-state channel, in which the messages are coded, with that of the same type of channel in which messages are not coded. In the uncoded case, only a bit of 0 or 1 is transmitted in one transmission; therefore, we can model the event count channel as a two-state graph in which a user can choose to increment or not increment the event-count variable by one. Thus, Ts is equal to either 0 or $T_{increment}$. In general, the sum of T_r, 2 T_{CS} and T_{env} is far greater than T_S and, thus, dominates the state-transition time. Hence, the state-transition times for the two-state graph are approximately equal. Gallager has shown that the maximum bandwidth can be achieved when both the distributions of both 0's and 1's are equal and the state-transition times are equal [3]. It is reasonable to let *Ts* equal $T_{increment}/2$. Thus, the time needed for a two-state channel to transmit n bits, $T_{two\text{-}state}$, is equal to $n(T_{increment}/2 + T_r + 2\,T_{CS} + T_{env})$.

Let $f(n) = T_{two\text{-}state} / T_{infinite\text{-}state}$ and $m = \dfrac{(T_r + 2T_{CS} + T_{env})}{T_{increment}/2}$. We obtain

$$f(n) = \frac{(m+1)n}{m + 2^{2+1} - 2}.$$

Fig. 9 demostrates the relation between n and $f(n)$ when m is equal to 100 and 1000, respectively. Apparently, when $f'(n) = 0$, $f(n)$ is optimal. We get

$$f'(n) = (m+1)\left[\frac{1}{m + 2^{n+1} - 2} - \frac{n2^{n+1} \ln 2}{\left(m + 2^{n+1} - 1\right)^2}\right] = 0.$$

Thus,

$$m = 2^{n+1}(n \ln 2 - 1) + 2.$$

When an m is given, the value of n that provides the maximum bandwidth of a infinite-state, event-count channel exists and can be determined. Their relation is shown in Fig. 10. We can conclude from the results shown in Fig. 10 that more bits should be coded and transferred in one transmission when m gets larger.

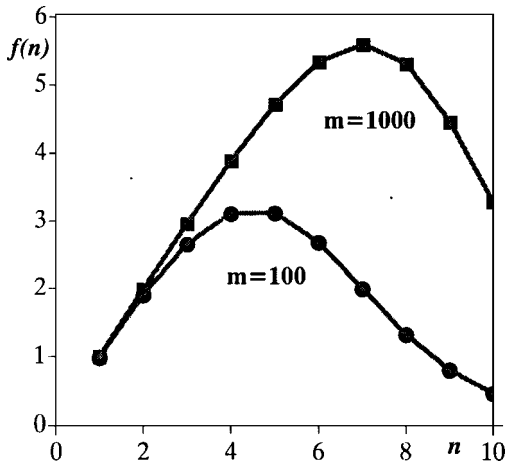


Fig. 9. Bandwidth improvement rate $f(n)$ with respect to the number of bits coded and transferred in one transmission when m = 100 and m = 1000, respectively.

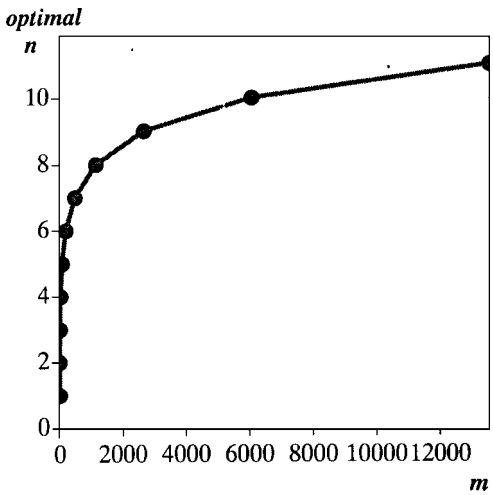


Fig. 10. Optimal n that can achieve the maximum bandwidth when given an m.

The accuracy of bandwidth estimation depends on the measurement of the primitive execution times. On the other hand, bandwidth estimation results are needed to perform measurement. We will discuss their relation ship in the next section.

5. BANDWIDTH MEASUREMENT

Little research to date has been able to measure covert channel bandwidths with much success. This is because (1) the execution time of primitives cannot be accurately measured, and (2) it is very difficult and complicated to construct the scenarios for the

actual use of covert channels. Covert-Channel bandwidth measurement is intended to discover the actual maximum bandwidth of each channel. In order to achieve this goal, three basic rules need to be followed: (1) the time needed for a primitive to view a variable with either error or no-error return should be known; (2) only the fastest pair of TCB primitives that alter and view each covert-channel variable should be chosen for testing; and (3) the chosen alteration primitives and viewing primitive should be able to cooperate to transmit information. Usually, if a primitive spends the least time viewing a variable with a no-error return, it also spends the least time viewing one with an error return. In other cases, including finite-state and infinite-state channels, we need the help from the estimation results to select primitives. If the primitives are appropriately selected, the actual maximum transmission rate can be measured.

The use of TCB primitives for covert-channel transmission of information can be both system-state (environment) and call-argument dependent. Therefore, the fastest primitive of a covert channel may not be chosen for data transmission. For example, the primitive *creat* can alter (i.e., decrement) the total number free inodes (nfi) only if the object to be created does not exist. If the object exists, *creat* has no effect on nfi. In addition, the primitive *creat* can be used to alter (i.e., increment) the total number of free blocks (nfb) in the system if the file being created currently exists. That is, if the file exists, *creat* truncates the file and, as a result, increments nfb. Otherwise, *creat* has no effect on nfb. On the other hand, the disk-block-space is also affected by this condition. Furthermore, alteration of disk-block-space channel, and of the nfi and nfb channel, and of the nfi and nfb channels by the primitive *creat* is determined by the file system specified in the argument of the *creat* invocation.

Many other primitives exist which are both system-state (environment) and call-argument dependent. Consider again a channel that modulates the nfb and the disk-block-space channel. The primitive *chsize* can be used to alter these channel variables (i.e., deallocate memory and increase the total number of free blocks) only if the file on which it is applied exists, and only if its argument indicates file shrinkage. When used to expand the size of an existing file, primitive *chsize* does not alter the channel variables but merely changes the $ip->i\_size$ field of the inode. Therefore, *chsize* cannot be selected for both nfb channel maximum bandwidth estimation and measurement though the execution time of *chsize* is shortest when it is used to expand the size of a file (see Fig. 11).

It should be noted that in many cases of resource-exhaustion channels, the test program does not leak any string of 0/1 bits. This is because whenever one of these resources are exhausted, the system performance deteriorates to such an extent that no information can possibly be transmitted within one second. In such cases, it is sufficient to measure the elapsed time from the beginning of the covert-channel primitive invocation until the resource-exhaustion error is returned to determine the upper bound of the achievable bandwidth.

According to the results of experiments, we conclude that the measured maximum bandwidth is lower than the estimated maximum bandwidth for two reasons: first, degradation may be caused by additional user load; second, the channel may have noise which is introduced by a process not involved in communication.

Unconfined processes may introduce noise and, thus, lead to false detection. To minimize false detection when covert channels are used, the noise must be removed during audit-trail analysis. The DoD guideline on covert channels [1] suggests that covert channels exceeding a threshold of 0.1 bit per second should be audited. This audit capability

UNIX primitives	execution time (ms)
chsize (expand)	0.4
chsize (shrink)	420
close	0.2
creat	30
exec	60
exit	10
fork	2.5
link	46
open (existing)	12
open (non-existing)	30
read	1
rmdir (non-empty)	3020
rmdir (empty)	180
unlink	22
ustat	0.4
write	0.72

Fig. 11. The execution time for a subset of trusted XENIX primitives.

provides the system administration with a means of detecting a significant compromise. In order to compare the bandwidth with the threshold, we need a method that can be used to measure the information rate from a sequence of observed data.

Let us consider the probability that a digit will appear again, which depends on past history. For example, a new digit is likely to be equal to 0 again, given that it has been equal to 0 in recorded history. We will analyze the entropy of an observed code and determine its information rate [14]. Let $X_1, X_2, \ldots, X_n, X_i \in \{0,1\}$, be independent and identically distributed (i.i.d.) Bernoulli random variables. Assume that the code is a Bernoulli(θ) process with an unknown parameter $\theta = Pr(X_i = 0)$, which is uniformly distributed on the unit interval. The probability $P_n^k(X_n, X_{n-1},..., X_1)$ of k selection of '0' in n positions is $\int_0^k \theta\,(1-\theta)^{n-k} d\theta$. Given $P_n^k(X_n, X_{n-1},..., X_1)$, we can calculate the posterior probability that the (n+1)-th digit will be '0' again:

$$
\begin{aligned}
&P_{n+1}^{k+1}(X_{n+1} = 0 \mid X_n, X_{n-1},..., X_1) \\
&= \frac{P_{n+1}^{k+1}(X_{n+1} = 0 | X_n,...,X_2, X_1)}{P_n^k(X_n,...,X_2,X_1)} \\
&= \frac{\int_0^1 \theta^{k+1}(1-\theta)^{n-k}\,\mathrm{d}\theta}{\int_0^1 \theta^{k}(1-\theta)^{n-k}\,\mathrm{d}\theta} \\
&= \frac{k+1}{n+2}.
\end{aligned}
$$

On the other hand, the posterior probability that the (n+1)-th digit will be '1' is

$$P_{n+1}^{k+1}(X_{n+1} = 1 \mid X_n, X_{n-1},..., X_1) = \frac{n-k+1}{n+2}.$$

Let P_0 and P_1 represent $P_{n+1}^{k+1}(X_{n+1}=0 \mid X_n, X_{n-1},..., X_1)$ and $P_{n+1}^{k+1}(X_{n+1}=1 \mid X_n, X_{n-1}, \ldots, X_1)$, respectively. Thus, the entropy is equal to

$$H_{n+1}^{k+1}(X_{n+1} \mid X_n, X_{n-1},..., X_1)$$

$$= -P_{n+1}^{k+1}(X_{n+1}=0 \mid X_n, X_{n-1},..., X_1)\log P_{n+1}^{k+1}(X_{n+1}=0 \mid X_n, X_{n-1},..., X_1)$$

$$- P_{n+1}^{k+1}(X_{n+1}=1 \mid X_n, X_{n-1},..., X_1)\log P_{n+1}^{k+1}(X_{n+1}=1 \mid X_n, X_{n-1},..., X_1)$$

$$= -\frac{k+1}{n+2}\log\frac{k+1}{n+2} - \frac{n-k+1}{n+2}\log\frac{n-k+1}{n+2}.$$

The entropy is highest when k is equal to $n/2$. This indicates that the higher the randomness and uncertainty is, the higher the information rate is. If the time needed to transmit a '0' and a '1', respectively, are different, in order to yield the maximum information rate (that is, the channel capacity) through this channel, zeros and ones are not sent with the same frequency. Assume that t_0 is the minimum time needed to transmit a bit of '0', and that t_1 is the minimum time needed to transmit a bit of '1'. The information rate of a channel C is equal to $H_{n+1}^{k+1}(X_{n+1} \mid X_n, X_{n-1},..., X_1)/t$, where t is the time needed to transmit one bit. Thus, the information rate of a channel is

$$C = \frac{H_{n+1}^{k+1}(X_{n+1} \mid X_n, X_{n-1},..., X_1)}{P_0 t_0 + P_1 t_1}$$

$$= -\frac{\frac{k+1}{n+2}\log\frac{k+1}{n+2} + \frac{n-k+1}{n+2}\log\frac{n-k+1}{n+2}}{\frac{k+1}{n+2}t_0 + \frac{n-k+1}{n+2}t_1}.$$

When $dC/dk=0$, we can derive the k that gives the channel capacity C_{max}. That is,

$$n+2\log\frac{k+1}{n+2} - (\frac{nt_1}{t_0}+2)\log\frac{n-k+1}{n+2} = 0.$$

For a given set of n, t_0, and t_1, we can uniquely determine the k that gives the channel capacity.

Unconfined processes may introduce noise into a channel. In this case, the randomness and uncertainty of the code sequence is generally low; that is, its information rate is low. The measured information rate B can be calculated as follows:

$$B = \frac{nH_{n+1}^{k+1}(X_{n+1}|X_n, X_{n-1},..., X_1)}{T},$$

where T is the total elapse time for transmitting an observed data sequence of length n. If B is greater than the threshold of 0.1 bit/sec, this transmission should be reported to the auditor. Otherwise, this transmission can be classified as noise.

6. CONCLUSIONS

Some covert channels can be modeled as finite-state graphs while others cannot. In this paper, we have proposed various coding techniques for determining the actual use of both finite-state and infinite-state covert channels to achieve maximum bandwidth. We have also presented methods for determining and estimating the maximum bandwidths of both types of channels. In order to resolve the fundamental difficulties of covert channel bandwidth measurement, we have provided basic rules which can help system administrators to ease and speedup the entire measurement process. Results of our analysis and experiments indicate that the methods proposed here are adequate.

REFERENCES

1. *Trusted Computer System Evaluation Criteria*, DoD STD-5200.28, 1985.
2. *A Guide to Understanding Covert Channel Analysis of Trusted Systems*, NCSC-TG-030, 1993.
3. R. G. Gallager, *Information Theory and Reliable Communication*, John Wiley, 1968.
4. V. D. Gligor, C. S. Chandersekaran, S. Chapman, L. Dotterer, M.S. Hecht and W.-D. Jiang, "Design and implementation of secure XENIX," *IEEE Transactions on Software Engineering*, Vol. 13, No. 2, 1987, pp. 208-221.
5. J. S. He, "An automated tool for the identification of covert storage channels," in *Proceedings of the IEEE Workshop on Computer Security Foundations*, 1990.
6. J. C. Huskamp, "Covert communication channels in timesharing systems," Technical Report UCB-CS-78-02, Ph.D. Thesis, University of California, 1978.
7. R. A. Kemmerer, "Shared resource matrix methodology: An approach to identifying storage and timing channels," *ACM Transactions on Computer Systems*, Vol. 1, No. 3, 1983, pp. 256-277.
8. B. Lampson, "A note on the confinement problem," *Communication of the ACM*, Vol. 16, No. 10, 1973, pp. 613-615.
9. S. B. Lipner, "A comment on the confinement problem," *ACM Operating System Review*, Vol. 9, No. 5, 1975, pp. 192-196.
10. J. K. Millen, "Finite-state noiseless covert channels," in *Proceedings of the IEEE Workshop on Computer Security Foundations*, 1989, pp. 81-86.
11. C. R. Tsai, V. D. Gligor and C. S. Chandersekaran, "A formal method for the identification of covert storage channels in source code," *IEEE Transactions on Software Engineering*, Vol. 16, No. 6, 1991, pp. 581-592.
12. C. R. Tsai, V. D. Gligor, "A bandwidth computation model for covert storage channels and its applications," in *Proceedings of the IEEE Symposium on Security and Privacy*, 1988, pp. 108-121.
13. C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, The University of Illinois Press, Urbana, Illinois, 1964.
14. S. P. Shieh, V. G. Gligor, "Detecting Illict leakage of information in operating systems," *Journal of Computer Security*, Vol. 4, No. 23, 1997, pp. 123-148.

Shiuh-Pyng Shieh（謝續平）received the M.S. and Ph. D. degrees in electrical engineering from the University of Maryland, College Park, in 1986 and 1991, respectively. He is currently the Director of the Computer Center and a Professor with the Department of Computer Science and Information Engineering, National Chiao Tung University. From 1988 to 1991, he participated in the design and implementation of the B2 Secure XENIX for IBM, Federal Sector Division, Gaithersburg, Maryland, USA. He is also the designer of SNP (Secure Network Protocols). Since 1994 he has been a consultant for the Computer and Communications Laboratory, Industrial Technology Research Institute, Taiwan, in the area of network security and distributed operating systems. He is also a consultant for the National Security Bureau, Taiwan. Dr. Shieh has been on the organizing committees of a number of conferences, such as the International Computer Symposium, and the International Conference on Parallel and Distributed Systems. Recently, he is the general chair of 1998 Network Security Technology Workshop, the program chair of 1999 Mobile Computing Conference and 1997 Information Security Conference (INFOSEC'97). His research interests include internetworking, distributed systems, and network security.