The Covert Channel Limiter Revisited

Keith Loepere
Advanced Software Engineering Technology
Honeywell Bull
Billerica, MA

January 20, 1989

Abstract

In a previous article[1], I introduced the idea of a mechanism (the covert channel limiter) that would watch for the potential uses of covert channels and affect the responsible process (or process group) only when such potential uses exceeded the allowable bandwidth for covert channels. Recent work involving the design of the Opus operating system (target class B3) has refined and extended this idea. This paper extends the informal basis for the covert channel limiter and extends its possible utility.

1 Introduction

When attempting to resolve covert channels during the B2 evaluation of Multics, covert channels were discovered for which removal was undesirable because of the difficulty involved in removing them or because of the resulting loss of functionality. A trivial example is the maintenance of time of last use for files (segments). It was recognized that the opening (activation) of a segment, which would cause the time of last use to be advanced, was a covert channel because any process capable of reading the segment (such as one whose authorization is greater than the access class of the segment) could advance the time of last use and any (other) process that could also read the segment, in particular, one whose authorization was less than that of the process using the segment, could read the time of last use. However,

[1] Loepere, K., "*Resolving Covert Channels Within a B2 Class Secure System*," Operating Systems Review Volume 19 #3, July 1985.

most activations of segments by processes whose authorization is greater than the access class of the segment do not represent attempts to transmit data through this covert channel. The idea of the covert channel limiter was to simply note events such as these that are potentially the use of a covert channel. Only if the usage gets too high to meet the covert channel guidelines is action taken against the offending process (or process group).

2 Covert Channel Bandwidth

The big question with respect to the covert channel limiter is: What actions (such as state changes or time delays) signal the potential use of a covert channel and what potential channel bandwidth is implied by these events?

The first part of this question (what actions) can be answered only via the covert channel analysis performed as a part of security evaluation. (A later section of this paper discusses some examples of actions resulting from the presence of certain system functionality to which the covert channel limiter may be applied.) Once an action has been analyzed to imply a potential usage of a covert channel, the question is: How many covert channel events does this action imply? This depends on the ability that a program has to cause this action. For example, if action A and action B always occur together, no matter what a program may do, then these pair of actions together constitute only one covert channel event. However, if a program may cause action A and action B to occur together by a single programmatic action, but only one of these actions to occur through some (different) programmatic action, then these pair of actions occurring together form two covert channel events. (For that matter, even just one of these actions occurring alone must be considered to be two events, since the program has the ability to encode two events through its choice of causing only action A or only action B or the pair of actions together.)

The question of bandwidth can be answered in a general fashion. Note, first, that it is the sum of all covert channel usages that counts. If a transmitting process uses N covert channels, each of which in and of itself would only allow a 1 bit per second transfer rate, that process could transmit at the rate of N bits per second. It follows that the covert channel limiter must sum over all channels in (apparent) use by the process (or process group) in the system.

It is very tempting to over estimate the information content of any given potential covert channel event. For instance, given N files, one might think

that the information content of advancing the time of last use of a file is log2(N) because the method of transmission might involve advancing the time of last use of only one of the N files and the method of reception might involve finding which file had its time of last use so advanced. Although this observation is true, it is not very useful. First of all, N (whether it applies to the total number of files, processes or whatever the system may have) is not large enough to make log2(N) very large. To really transmit a stream of data, a process would need to cause more than one covert channel event. As there are more covert channel events, the potential information content of each is lessened. (Imagine an encoding taking N things of which M have a state change and the rest do not. The number of different combinations of these state changed objects increases with M, but the information content for each state changed object (that is, the combinations divided by M) decreases as M increases.)

Another way of looking at this, again with the example of advancing the time of last use of a file, is to imagine that the transmission involves encoding information by when the time of last use was updated. The amount of information that could be encoded is a function of the resolution of the time value (and could be many bits if reasonable control over when the time of last use gets set is possible), but, again, when talking about affecting multiple objects to transmit a stream of data, as one tries for greater control over setting the time of last use of each object (that is, allowing oneself more time between changes), the information content transmitted in a particular unit of time decreases.

Since information content per time is the issue here, what we want to know is what is the maximum information content per event that a process could sustain. Since the ability to encode data (log2 of the elapsed time between actions or log2 of the possible combinations of state changes) goes up slower than the elapsed time between actions, the maximum is obtained through minimizing the elapsed time between actions, making the maximum sustained information content per event to be 2. The information content of any given covert channel event may be less than 2, but, in the long run, it is known to be no more than 2. (A simple way of achieving this maximum (of 2) is to simply choose one object. The method of transmission involves either affecting its state to transmit a 1 bit or not affecting its state to transmit 0. Assuming an equal distribution of 1 and 0 bits, each state change for the object must be assumed to be coupled with a lack of a state change, thus making each state change have an information content of 2 bits.)

Thus, the covert channel limiter need only count the number of events in

a given time unit and multiply by 2 to get an upper bound on the information potentially transmitted during that time unit.

3 Applicability to Multi-Class Lock Management

The covert channel limiter provides a solution to the problem of managing locks on data held by processes of differing authorizations. Consider an implementation in which a process attempting to make a lock request against a lock already held (in a conflicting mode) by some other process is immediately told of the conflict. (An implementation that causes the requesting process to block until its request is satisfied or until the request timed out would just turn this into a timing channel. Since it is the receiving process that blocks (and the receiver can have an arbitrarily large number of blocked processes), the bandwidth is not affected by this alternate implementation, so the distinction of storage versus timing channel is really moot. Note, though, that the channel is a time decaying one regardless of the implementation.) Locks can be used to transmit data only if two conditions are met:

- The data to be locked is of lesser access class than the authorization of the locking process. (The assumption here is that a process is not allowed to request a lock on data whose access class is greater than its authorization.)
- Some other process of lesser authorization must attempt to get a lock while the data is locked.

Any attempt to obtain a conflicting lock against a lock held by a greater authorization process counts as a potential covert channel transmission by the higher authorization process. Such an attempt would increment the higher authorization process' covert channel event counter. The scheduler could affect the higher authorization process based upon its covert channel counter when it is next scheduled.

4 Applicability to Page Residency

For most systems in which each process has its own page tables, covert channels involving page residency can be easily handled. Some current solutions involve delaying any process that takes a page fault (that is, finds its own

page table entry invalid) regardless of whether the page is in memory or not, so that the process cannot detect if another process paged in the page. With the covert channel limiter, taking a page fault against an in memory page simply increments the covert channel count for the process that brought in the page, and then only if that process has an authorization greater than that of the current faulting process.

5 Applicability to CPU Scheduling

The covert channel resulting from the order of CPU scheduling can be handled by comparing the authorizations of the process being out-scheduled and the process being in-scheduled. Scheduling a process with a lesser authorization counts as a covert channel event for the process being out-scheduled.

6 Applicability to a Class of Timing Channels

Timing channels can be very difficult to analyze and remove. However, an understanding of how many of these channels come into being provides a basis for an approach to handle many of these channels in a simple, uniform way.

Consider a kernel implementation that utilizes a wait/notify mechanism (as does Multics and UNIX[2]) for kernel synchronization. With this mechanism, a process that needs to wait for an event puts itself into a wait state waiting for a particular event ID. When this event occurs (most typically the unlocking of an associated lock), this event ID is notified and all processes waiting for the event are put into the ready state. This unifying mechanism for kernel synchronization provides a unifying point for much covert channel detection. Many of the timing channels appearing in a system are caused by the time delay resulting from a process waiting to be notified of an event by a greater authorization process. By judiciously keeping track of the process ID (and therefore the authorization) of the process that causes an event to be notified, and comparing that authorization with that of the processes being notified, many potential timing channels can be located. Notifying a process whose authorization is lower than that of the process causing the notify would increment the causing process' covert channel event count.

The two main uses of the wait/notify mechanism in a typical kernel are for kernel (and possibly other) locks and page I/O completion. The

[2]UNIX is a trademark of AT&T.

utility of this idea to locks is obvious. For I/O completion, there is a minor complication that the notifying process must be considered to be the process that started the I/O, not the process fielding the completion interrupt.

It is not clear whether this idea to move covert channel detection into the wait/notify logic is appropriate. Doing so clearly increases the cost of the wait/notify logic at the saving of covert channel detection in many areas of the kernel. This is a trade-off that needs to be examined more closely.