An Analysis of Covert Timing Channels

John C. Wray*
Digital Equipment Corporation
295 Foster Street (LTN1-1/E03)
Littleton, MA 01460-1123

Abstract

Covert channels have traditionally been categorized as either storage channels or timing channels[6, 7]. This paper questions this categorization, and discusses channels that cannot be clearly identified as either storage or timing channels, but have aspects of both.

A new model of timing channels is presented, which allows for channels that have characteristics of both storage channels and timing channels, and a method is given for constructing all channels in a computer system that have timing channel characteristics. Since Kemmerer's shared resource matrix methodology[5] has the potential to detect all channels that have storage characteristics, the two methods jointly have the capability to construct all channels in a computer system.

The approach to timing channels provides a justification of specific mechanisms for reducing their bandwidth which were employed in the VAX[1] Virtual Machine Monitor, as described in [3] and [2].

1 Introduction

Covert channels have traditionally been divided into two classes, storage channels and timing channels. The distinguishing feature between the two classes is that in timing channels, information is conveyed by the timings of events, and thus they require the receiving process to have access to an independent clock with which these timings may be measured, whereas storage channels are exploitable without the aid of an external time reference.

In the covert channel analysis of VAX Virtual Machine Monitor[4], this distinction was not found to be useful. Some channels were found which, by minor variations in the channel exploitation, appeared to change from storage channels to timing channels, and vice-versa. Since the underlying mechanism of the channel remained unchanged across the different exploitations, this casts doubt on the validity of the categorization.

*This paper presents the opinions of its author, which are not necessarily those of the Digital Equipment Corporation. Opinions expressed in this paper must not be construed to imply any product commitment on the part of the Digital Equipment Corporation.

[1]VAX is a trademark of Digital Equipment Corporation

This paper presents an example of such a channel, and presents a modified treatment of covert channels, whereby ***storage nature*** and ***timing nature*** are attributes of the channel, and a given channel may possess either or both. Channels that possess timing nature are examined, leading to a method whereby they may be constructed from a knowledge of a system's asynchronous behavior.

2 The Disk-arm Channel

A covert channel involving the placement of the arm of a shared disk-drive was first identified in KVM-370[1]. This variant of the channel operates as follows:

Assume that the shared disk drive contains adjacent cylinders 51 through 59, and is shared for read-access by two virtual machines **HIGH** and **LOW**, operating at high and low secrecy levels respectively. When the disk controller has multiple seek requests outstanding, it implements an elevator algorithm to determine the order in which to service the requests.

Initially, **LOW** issues a read request to cylinder 55, waits for the request to complete, and then gives up the CPU. **HIGH** now runs, and issues a request to seek to either cylinder 53 (to send a zero bit) or 57 (to send a one), and immediately relinquishes the CPU. **LOW** then issues seek requests to both cylinders 52 and 58 and observes the order of completion of these requests. If **HIGH** had issued its seek request to cylinder 53, the arm would have been traveling in the direction of decreasing cylinder number, and would have continued in this direction, servicing **LOW**'s requests in the order 52 followed by 58. If **HIGH** had issued its seek to cylinder 57, then **LOW**'s request would have been serviced in the reverse order.

This channel was described as a timing channel because it relies on the relative timing of the two seeks initiated by **LOW**. However, this variant of the channel requires no external clock, the data flowing through the channel being determined purely by the order of two events (the completion of the seeks). The channel would still operate if **LOW** issued a single seek to cylinder 58 and measured the time taken for this request to complete. If **HIGH**'s seek had been to cylinder 57, then **LOW**'s seek would complete much sooner than if **HIGH** had moved the arm towards cylinder 53. Given this exploitation, the channel is clearly a timing channel.

The channel does not strictly require the disk controller to implement an elevator algorithm. Consider the same disk, using an arbitrary algorithm to determine the direction in which to move the arm, the only constraint being that if the arm is moved over a cylinder to which a seek is pending, then that seek shall complete. Assume that **HIGH** issues a seek request to either cylinder 51 (to send a zero) or to cylinder 59 (to send a one), and waits for the seek to complete before relinquishing the CPU. **LOW** then issues two seek requests to cylinders 53 and 57. The completion order of these two seeks is completely determined by the prior positioning of the arm by **HIGH**. Again, if **LOW** were able to accurately measure the time taken for the completion of a seek request, it could exploit the channel using only a single seek.

These different exploitations all seem to be variations on the same underlying channel mechanism - the property that the time taken for a seek request to complete depends on the position (and possibly prior direction of motion) of the disk arm at the time the seek request is issued. Hence, it is attractive to view these exploitations as simply variations of the same channel. However, if the distinguishing feature of timing channels is that they require an external clock, then it would appear that the channel variants involving two seek operations are storage channels, while the single-seek versions are timing channels.

This channel (and some additional variants) were addressed in VAX Virtual Machine Monitor by the application of a variety of techniques, the goal of which was to make the timing of a seek operation (either as a pure seek or as part of a read or write operation) invisible to user-processes, both in absolute terms and in relation to other seek operations to the same disk device. These techniques are described in [3].

3 Clocks

Since the conventional definition of a timing channel refers to the use of a clock, it is instructive to examine what are the defining properties of a clock. In its broadest sense, any method for measuring the progress of time may be deemed a clock. We view the passage of time as being characterized by sequences of events which can be distinguished one from another by an observer [2].

Given internal memory, an observer will be able to distinguish its internal events from one another, and will thus be able to use these events to measure the passage of time. All processes on a computer have internal memory, and therefore it would appear that all processes can act as clocks. In practice, a timing loop may be used, each distinguished event being the increase in value of a loop-counter.

The above argument leads to an interesting symmetry: The passage of time is simply the occurrence of sequences of distinguished events. One sequence of events may be used as a measure, with which a contemporary sequence may be compared, to determine the relative ordering of the events in the two sequences. In other words, one sequence of events can serve as a clock which may be used to measure the (relative) timing of other event sequences.

The above description of a clock includes event sequences that are irregular. This is intentional, since the only way to determine whether a particular event sequence is "regular" is to measure the time between successive events using another clock. This argues that there is no absolute scale against which time may be measured; rather time may only be measured in terms of event orders.

Given the absence of an absolute time scale, the *relative* qualifier used above may be dropped as superfluous, which results in a complete equivalence of measured property and measuring device.

Using this equivalence between a clock and an event stream, we can re-state the timing channel definition as a channel in which information is conveyed by the relative timings of two clocks or event sequences visible to the observer. The transmitter injects information into the channel by modulating the relative frequency or phase of the two clocks.

In many simple timing channels, a single clock is modulated by the transmitter, and the receiver compares this clock with a reference clock to detect the modulation. In more complicated channel exploitations, both clocks are modulated (in anti-phase) by the transmitter.

4 The disk channel as a pair of clocks

Re-examining the four variants of the disk arm channel discussed above, we find that two clocks are involved in each. A sequence of pairs of (***issue-seek***, ***complete-seek***) events forms a clock, and there are two such clocks operating simultaneously in the first and third variants. The second and fourth variant use a single (***issue-seek***, ***complete-seek***) sequence, and require the use of an additional clock by the receiving process to measure the timing of this sequence. The receiving process may make use of a timing loop to synthesize this additional clock.

5 Dual-clock analysis

Since a timing channel exploitation requires the simultaneous observation of two clocks by the receiver, a method of constructing timing channels is suggested. The analyst first enumerates all clocks in the system, and then examines them pair-wise to determine

[2] Whether or not a given event can be distinguished from another may depend on the observer. For example, if an event consists of a boolean variable changing state, then there are only two intrinsically distinguished events (**true** $\rightarrow$ **false**) and (**false** $\rightarrow$ **true**). However, if an observer is incrementing a count of how many times the variable has changed state, then there are an infinite number of possible distinguished events for that observer, each event being the change in the value of the counter from N to $N+1$. The presence of sufficient internal memory allows an observer to distinguish arbitrary numbers of otherwise indistinguishable events

whether each pair of clocks can be exploited as a channel. This approach is best visualized as the construction of a matrix, whose rows list clocks that are modulated in the channel exploitation, and whose columns list clocks used as reference clocks in the exploitation. Each element of the matrix then indicates a potential channel. The analyst must determine whether each of these N^2 potential channels (where N is the number of clocks in the system) is exploitable, and if so, what its bandwidth is.

This is the approach that was taken in the VAX VMM covert channel analysis, and it successfully exposed many timing channels, some of which are used as examples throughout this paper. It also provides a justification for the validity of the counter-measures that were applied to reduce the bandwidths of all timing channels in the system (section 9 and [2]).

In a computer system, some clocks may be intrinsically impossible to modulate. Such clocks can only be used as reference clocks, against which modulated clocks may be measured, reducing the number of potential channels that must be investigated from the N^2 elements of the above matrix to $N \times M$, where M is the number of clocks capable of being modulated. However, practical experience with VAX VMM has shown that almost all clocks can be modulated by a sufficiently determined attacker, as discussed in section 8, and thus this approach may offer little to simplify the analyst's task.

Given that most clocks can be modulated, it may be better to abandon any attempt to distinguish between modulated and reference clocks. Viewed this way, the matrix lists potential channels (other than than those on the leading diagonal) twice. The analyst can therefore ignore half of these channels, resulting in $N \cdot (N+1)/2$ potential channels that must be investigated. Each such potential channel must be examined to determine whether the modulation of either clock is feasible.

This analysis would be of little practical value if clocks were as difficult to locate as covert channels. Fortunately, in a conventional computer system there are only a small number of places where clocks may arise. This can be seen to be true by recognizing that all computer systems offer a process the ability to construct a clock by executing a timing loop, using the CPU instruction-cycle clock. All other distinct clocks must operate asynchronously to this. Conventional computers confine asynchronous behavior to a few areas resulting in the following list of possible clock sources:

- The CPU instruction-cycle clock, which allows processes to construct accurate timing loops. In a multiprocessor system whose CPU's do not operate in lock-step, there is one such clock per CPU.

- Real-time clocks or time-of-day registers, which are provided by the hardware designers as intentional software-readable clocks.

- The IO subsystem (completion interrupts, DMA data arrival rate, etc)

- The memory subsystem (data/instruction fetch, interlocks, etc)

Examining each of these areas uncovers several clocks, most of which may be modulated. For example, the memory subsystem provides a clock based on the time take to perform a memory fetch. This time depends on whether the target data was in the cache (and, for a system with a multi-level cache, which cache it was in), and on the level of traffic on the memory bus and through the memory controller. Most of these factors can in principal be adjusted by another process.

Many of these clocks occur in large numbers. For example, the clock formed from the (***issue-seek***, ***complete-seek***) sequence can be replicated on each disk-drive (assuming that the disk controllers have sufficient throughput). This can be used either to replicate entire covert channels (thus increasing the aggregate throughput of the channels), or to simply construct additional reference clocks.

Some of these clocks are not independent. As an example, consider a process using DMA data arrival to construct a clock. The process issues a disk read request into a buffer, and then polls the first byte of the buffer. When the data in that byte changes, the process knows the DMA transfer has started. It then polls another location, further along in the buffer, and when that data changes the process knows that the transfer has reached that point in the buffer. This method gives the process an accurate and adjustable clock. However, DMA may cause a high level of memory activity, disturbing the operation of clocks constructed from the memory subsystem.

The VAX VMM experience has shown that it is much easier to discover clocks than to directly discover timing channels. This DMA clock described above was present in an early prototype of VAX VMM, but was eliminated from the product-quality version by the techniques described in [3].

6 Storage or timing?

Given that the defining feature of a timing channel is that two independent clocks are required by the receiver, all four of the earlier disk-arm channel variants appear to be timing channels. This was indeed the category into which the authors of [1] placed this channel. However, the data is stored in the channel in the position and direction of motion of a physical disk arm, which can be viewed as a storage object, and this argues that the channel is a storage channel.

This illustrates that in many cases there is no sharp distinction between a storage channel and a timing channel. In the case of the disk arm channel, the transmitter injects data into the channel by setting the position and direction of the disk arm, employing it as a storage object, whereas the receiver extracts that information by generating two clocks whose relative rates are affected by the state of the arm. If one considers the injection of information, then the channel is a storage channel, whereas if one considers the method used to extract information, then the channel appears to be a timing channel. In practice, it seems simpler to say that there is a covert channel which has both storage and timing characteristics. This means that the channel may be detected either by a method that detects storage channel, or by a method that detects timing channels.

Kemmerer's shared-resource matrix methodology has produced good results when used to detect storage channels, and in principle allows all storage channels in a system to be located. The method operates by listing possible storage objects that may be shared, and identifies operations that may inject or extract data from these storage objects. In principal, all storage channels (or all covert channels which employ a shared storage object) may be detected. However, in practical use, the SRM analysis is not normally applied to the entire software and hardware of a system, and therefore some storage channels may evade detection.

The dual-clock construction allows channels which have a timing component to be constructed, much as the SRM analysis constructs possible storage channels. As with the SRM analysis, while theoretically capable of producing all covert channels that have a timing component, the fraction of channels that are detected in practice will depend on the depth to which the system is analyzed, in particular on the depth of the search for clocks.

The approach outlined here was used in the timing channel analysis in the VAX VMM. It detected many timing channels in the prototype, many of which were shown by subsequent testing to be exploitable at high speed. These channels were subsequently closed or reduced in bandwidth in the product-quality version of the kernel.

7 Direct channels

So far, the channels described have been examples of what we shall term *process channels*, where the receiver is a process executing on the system. A second class of channels will be termed *direct channels*. In these channels, the information being leaked through the channel appears directly on an output device. As an example, consider a variant of the above disk-arm channel. This requires five tracks, named **A** through **E** in order of increasing distance from the center spindle. Tracks **B**, **C** and **D** are adjacent. To transmit a single bit, the transmitter issues a seek to place the arm on either track **A** or track **E**. The low secrecy process is assumed to have write-access to the disk drive, and issues two simultaneous write requests, one to an area that includes a block on track **B** and all blocks on track **C**, and the other to an area that includes all blocks on track **C** and one block on track **D**. The order in which the write requests complete will be determined by which of tracks **A** and **E** the transmitter placed the arm. Assuming different data was written by each write request, the final value stored in track **C** will indicate the value of the transmitted bit. The low secrecy process does not need to determine the completion order in order to exploit this channel – the information is sent directly to a disk device, where it remains as low-secrecy data.

There are still two clocks operating in this version of the channel, formed from (*seek*,*write*) pairs. The difference is that the clocks are constructed in such a way that the comparison occurs inside the disk-drive, rather than in an running program.

Another example of a direct channel uses a serial terminal line, and requires a multiprocessor system. A low-secrecy process, **LOW**, initiates a large (many-character) asynchronous write operation. It then enters a loop, making memory references that avoid any caches available to **LOW**. A high-secrecy process, running simultaneously on another processor, makes either similar memory references (to send a one), or avoids accessing the memory bus (to send a zero). The time taken for **LOW** to complete its series of memory references is thus modulated by the actions of **HIGH**. **LOW** then issues a *cancel* request to the serial line. The length of time taken for **LOW** to complete its loop (and therefore the bit transmitted by **HIGH**) can be inferred from the number of characters that were sent down the serial line before the *cancel* aborted the write.

The fundamental feature of channels that have timing characteristics is that they employ two clocks, whose relative rates may be varied by the activity of a transmitting process, and which may be compared to detect the variations in the rates. This treatment of timing channels bears a close resemblance to the way in which frequency modulated radio signals may be demodulated, by comparing them with a locally generated signal of fixed frequency in such a way that the difference between the information-carrying signal and the reference signal (the information itself) is revealed.

8 Modulation Scenarios

The experience of VAX Virtual Machine Monitor indicates that almost every clock can be modulated by a sophisticated transmitter. Consider a clock provided by the regular interrupt-stream produced by a dedicated hardware clock-generator. It might be thought that this clock could not be modulated by software. However, the event-stream that is visible to a user of the clock is not the stream of interrupt requests themselves, but instead the regular diversions of the processor into interrupt-handling code. If a transmitter can introduce a variable interrupt latency into the system (for example, by performing I/O in such a way as to generate significant device interrupt activity), then the effect is to delay the visibility of the interrupts to the receiving process.

Even the clock produced by an instruction timing loop may be modulated on most multiprocessor systems, as discussed above in section 7. Provided that the timing loop is implemented by instructions that access main memory[3] and that a single common asynchronous memory bus is used, then a process executing on a different processor from the one which is running the timing loop may flood the memory bus with transfer requests, causing each cycle of the timing loop to take longer due to the bus contention. An exploitation of a channel based on this technique was demonstrated on an early prototype of VAX VMM and found to allow high speed information leakage. The bandwidth of this channel was reduced to an acceptable level in the final version of the system through the use of techniques described briefly in section 9 and in detail in [2].

While this exploitation takes pains to avoid cache hits in order that the system memory bus will become a performance bottleneck, a different modulation scenario can exploit cache hits. Consider a uniprocessor with a direct-mapped cache, running two processes containing covert channel exploitation programs. One, at low secrecy, reads sufficient memory locations to fill the cache with low secrecy data, and then relinquishes the CPU. The other process, running at a high secrecy level, read certain memory locations, causing some cache slots to be re-filled with high secrecy data, and then relinquishes the CPU. Finally, the first process rereads the data it read earlier, but measuring the time of each read attempt against a reference clock. Those memory locations which correspond to cache locations that were filled with high secrecy data will take significantly longer to read than the locations that still contain low secrecy data, as the displaced low-secrecy data must be re-fetched from main memory. This exploitation allows the timing of individual instructions to be modulated with precision. It is a direct analogue of the Multics page-fault channel, but as it operates at a level much closer to the hardware, it is significantly faster.

9 Fuzzy Time

The measures described in [2], collectively known as "fuzzy time", may be justified from the previous discussion. Information is carried in a timing channel in the difference in rates of two clocks. Therefore, if all clocks were removed from a computer system, then that computer system would exhibit no timing channels. Such a computer would be useless, since (as discussed above) removal of all clocks would entail removal of all memory. However, if we accept that any process may construct an accurate clock from a timing loop, we can still eliminate timing channels by eliminating all other clocks, since a timing channel requires a pair of clocks. While it is theoretically possible to completely remove visibility of asynchronous events from a computer system, in practice this would result in very poor performance. It is, however, possible to add mechanisms which disrupt attempts to compare clocks by introducing random variations into the visible timings of events. Simply adding random delays to events is equivalent to adding noise into the system, which is likely to severely degrade system performance and which may be countered by appropriate encoding techniques. The techniques used in VAX VMM effectively notify processes of asynchronous events only at pre-computed times, such that the only information that may be deduced from the event delivery is that the event actually occured previously. The experience of VAX VMM has been that the impact of such measures on system performance can be made small in comparison with the dramatic reduction in the exploitable bandwidth of the system's timing channels that results from their adoption.

[3] This may require careful processor-specific design in order to avoid per-processor cache memory preventing the access from reaching main memory.

10 Summary

The analysis method described above is a distillation of the informal methods used to locate timing channels in VAX VMM. Avoidance of covert channels had been an early design goal, and this was addressed to a large degree by eliminating dynamic resource allocation wherever possible. This technique was successful in that the number of storage channels found in the system was small. However, many timing channels were found using the methodology described in this paper; moreover, many of these resulted from operations which were outside the control of the software portion of the TCB. These results lead to the development and adoption of the specific bandwidth reduction techniques described in [2] and [3].

Concentrating on identification of clocks, categorized as asynchronous event-streams, quickly identifies the channels and suggests their exploitations. The most obvious clocks tend to be discovered early, and these are usually those that are easiest to use in an exploitation. This means that the channels which are easiest to exploit are identified early in the analysis process.

Acknowledgments

The ideas presented in this paper are the result of discussions with many people, but especially Wei Hu, Clifford Kahn, Paul Karger and Andrew Mason.

References

[1] B. D. Gold, R. R. Linde, R. J. Peeler, M. Schaefer, J. F. Scheid, and P. D. Ward. A security retrofit of VM/370. In *AFIPS Conference Proceedings, Volume 48, 1979 National Computer Conference*, pages 335–344, AFIPS Press, Montvale, NJ, 1979.

[2] Wei-Ming Hu. Reducing timing channels by fuzzy time. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society, Oakland, CA, 20–22 May 1991.

[3] Paul A. Karger and John C. Wray. Storage channels in disk arm optimization. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society, Oakland, CA, 20–22 May 1991.

[4] Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn. A VMM security kernel for the VAX architecture. In *1990 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society, Oakland, CA, 7–9 May 1990.

[5] Richard A. Kemmerer. A practical approach to identifying storage and timing channels. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 66–73, IEEE Computer Society, Oakland, CA, 26 - 28 April 1982.

[6] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[7] Steven B. Lipner. A comment on the confinement problem. *Operating Systems Review*, 9(5):192–196, November 1975. Proceedings of the Fifth Symposium on Operating Systems Principles, Unversity of Texas at Austin, Austin, TX, USA, 19–21 November 1975.

[8] John C Wray. A methodology for the detection of timing channels. June 1990. Presented at The Computer Security Foundations Workshop III, Franconia; To appear in Cipher.