Comparing Countermeasures against Interrupt-Related Covert Channels in an Information-Theoretic Framework

Heiko Mantel and Henning Sudbrock
RWTH Aachen University
Security Engineering Group
{mantel,sudbrock}@cs.rwth-aachen.de

Abstract

Interrupt-driven communication with hardware devices can be exploited for establishing covert channels. In this article, we propose an information-theoretic framework for analyzing the bandwidth of such interrupt-related channels while taking aspects of noise into account. As countermeasures, we present mechanisms that are already implemented in some operating systems, though for a different purpose. Based on our formal framework, the effectiveness of the mechanisms is evaluated. Despite the large body of work on covert channels, this is the first comprehensive account of interrupt-related covert channel analysis and mitigation.

1. Introduction

Application-level security necessarily relies on some properties of the underlying system layers. The concept of process separation is crucial in this respect as it provides protection between application programs. While separation is often taken for granted, it can be rather non-trivial to separate processes completely from each other. One problem is that one must not only rule out communication via overt channels like shared memory, but also via *covert channels* [Lam73], which are not intended for communication.

Covert channels can be established based on shared resources. For instance, a process can excessively access virtual memory in order to increase the amount of physical memory allocated to it, thereby increasing the likelihood of paging for other processes. These processes measure the delay caused by paging to establish a covert channel where messages are encoded by the paging rate. Hard disks, network cards, and the CPU itself are further examples of resources that are possible sources of covert channels.

Covert channels and covert channel analysis have received much attention by the research community over the last 30 years. Nevertheless, the problem of covert channels is not completely solved – neither in practice nor in theory.

This article focuses on covert channels that involve interrupts. For instance, a process can access a file such that the corresponding interrupt by the hard-disk controller occurs while some other process is running. A covert channel can then be established by encoding messages by the delay caused by such interrupts. While interrupt-related covert channels, in principle, could be avoided by polling devices instead of using interrupt-driven communication, such solutions are impractical in most cases (see Section 2). However, without sacrificing the communication paradigm, it turns out to be difficult to completely eliminate interrupt-related channels. If one cannot eliminate these channels then one at least should be able to assess their dangers. For this purpose, we developed methods for analyzing and limiting the bandwidth of interrupt-related covert channels.

The novel contributions of this article are:

- a collection of mechanisms that can be adopted as countermeasures against interrupt-related covert channels and that are already available in some systems,
- a formal framework for analyzing the bandwidth of interrupt-related channels and of the effectiveness of countermeasures against such covert channels, and
- an evaluation of the mechanisms in our information-theoretic framework, resulting in a classification of the various mechanisms under given conditions.

We found it somewhat surprising that interrupt-related covert channels have received only very little attention so far. A notable exception is [Tro98], where an analysis is presented on how to exploit the duration of keyboard-interrupts in order to deduce information about a secret password while it is typed in. The scope of the current article differs in that we focus on the intentional covert communication between processes and in that we strive for a more formal evaluation for this class of covert channels.

Published in Proceedings of the 20th IEEE Computer Security Foundations Symposium, Venice, Italy, July 6-8, 2007. IEEE Computer Society, pages 326–340.

2. Covert channels

McHugh defines a covert channel as "*a mechanism that can be used to transfer information from one user of a system to another using means not intended for this purpose by the system developers*" [McH95]. Traditionally, one distinguishes timing channels from storage channels. A covert channel is a *timing channel* if its usage involves a global clock or a process's local perception of time. If time is not involved then a channel is referred to as *storage channel*.

2.1. Timing channels and quotas

In this article, we focus on timing channels. The following example illustrates two possibilities for establishing a timing channel based on the CPU usage of a process.

Example 1 *Assume a Round-Robin scheduler and that only two processes are active, the sender and the receiver. The sender encodes messages by how much of a given time quantum it actually uses. For sending the value* 0*, the sender yields the CPU immediately at the beginning of the quantum. For sending a* 1*, the sender uses its entire quantum. The receiver can then re-construct the value based on the starting time of its subsequent time quantum. Alternatively, messages can also be encoded by the wake-up time of the sender: Sender and receiver agree on fixed points in time* t_k*. If the sender sleeps at time* t_i*, the* i*th bit sent is* 0*, otherwise it is* 1*. In [Hus78], these two channels are referred to as* quantum-time channel *and* interquantum-time channel*, respectively.* $\Diamond$

To rule out side effects of using a shared resource, one can assign to each process a fixed quota for using this resource. Within its quota, a process can utilize the resource at its discretion – but not beyond. Effectively, the resource is virtualized, giving each process the impression of having exclusive access to an unshared resource with lower performance.

Example 2 *Assume that the CPU quanta are fixed for the processes from Example 1, including the starting time and the duration of each given time quantum. Then there is no danger of quantum-time channels because yielding the CPU does not cause the subsequent time quantum to start earlier. Interquantum channels are avoided by allowing processes to wake up only during their own time quanta.* $\Diamond$

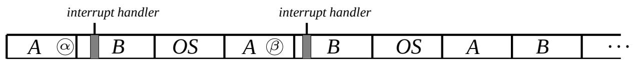
Analogously, quotas can be employed for virtualizing other resources. For instance, the paging-based covert channel from Section 1 can be excluded by assigning a fixed quantum of physical memory to each process.

2.2. Interrupt-related channels

Unfortunately, the solution in Example 2 is not sufficient for eliminating timing channels if the usual interrupt-driven communication is used for interacting with hardware devices. For instance, when a file transmission from the hard disk into RAM is completed then the hard-disk controller issues an interrupt. When the interrupt controller receives the interrupt, it informs the CPU, and the currently running process is stopped in order to handle the interrupt. This behavior can be exploited to establish a timing channel where the sender encodes messages by the number of interrupts to be handled during the receiver's quanta:

Example 3 *Assume a Round-Robin scheduler and three active processes: an operating-system process* OS*, a process* A *(the sender), and a process* B *(the receiver). For sending a* 1*, the sender accesses a file such that the corresponding interrupt occurs during the receiver's subsequent quantum, while, for sending a* 0*, no interrupt is initiated. During its time-slots, the receiver measures the time to detect delays caused by interrupt handling.*

The following diagram illustrates the transmission of the sequence $\langle 1, 1, 0\rangle$ *from* A *to* B*. In the diagram, time progresses from left to right, and the labeled boxes represent the time quanta where the label indicates the process:*



The labeled circles indicate the points in time when file accesses occur. Process A *performs a file access at* α *during its first quantum, and the corresponding interrupt is handled during* B*'s first quantum (indicated by the shaded area in the box representing* B*'s first time quantum). Process* A *issues another file access at time* β*. During its third time quantum,* A *does not perform any file accesses. Process* B *can then reconstruct the sequence by measuring the delay caused by interrupts in its three time-slots.* $\Diamond$

Interrupts are referred to as *asynchronous interrupts* if they originate from a hardware device at some point in time after the access to the device was requested (like, e.g., a keyboard interrupt or the interrupts in Example 3). In contrast, synchronous interrupts (or software interrupts) immediately occur after a given instruction (like, e.g., interrupts caused by a division by zero or a system call). Apparently, interrupt-related channels can only be created via asynchronous interrupts that are not masked by the operating system.[1]

Exploitation scenarios like the one in Example 3 are realistic threats. The access time of contemporary hard disks is around 5–10 milliseconds, while a base time quantum of a process with nice value 0 in the Linux Kernel (version 2.6) has a length of 100 milliseconds [BC06]. This allows a sender to aim precisely at a time quantum of the subsequently scheduled process. If there are more than two active

[1]Note that masking blocks interrupts on a per-device basis, not on a per-process basis. This is why one cannot mitigate interrupt-related covert channels based on masking analogously to the solution in Example 2.

processes and the time quantum of the receiver does not directly follow the sender's time quantum then the sender can increase the amount of data that is transferred in order to delay the interrupt to the next time quantum of the receiver.

Besides hard disks, there are various other devices that provide possibilities for covertly communicating based on asynchronous interrupts. For instance, CD/DVD drives, sound cards, network adapters, and sensors cause interrupts that all can be the source of covert channels. In principle, one could eliminate interrupt-related covert channels by changing the paradigm for communicating with devices. Instead of a device informing a process that it needs attention, each process could actively poll devices for information. However, there are good reasons for not migrating to polling. Namely, delays in handling requests might lead to data being lost, e.g., when no free buffer space is available for a packet that arrived on the network. Moreover, polling increases the communication overhead between the CPU and devices, which may lead to significant performance penalties. This degradation of performance would come on top of the one caused by using quotas. Consequently, mechanisms against interrupt-related covert channels are attractive if they allow one to retain the main advantages of interrupt-driven communication.

As it is not always feasible to eliminate interrupt-related covert channels completely, one might want to select the countermeasure with the best cost-benefit ratio instead. For determining the benefits of a given mechanism one needs to calculate the limits that it imposes on the bandwidth of the given channel. For answering, how dangerous a given covert channel is, one needs a quantitative analysis. A qualitative analysis (answering: Is there a covert channel?) would not be sufficient as it is clear that the countermeasure does not eliminate the channel. In the subsequent section, we propose such a quantitative analysis.

3. The information-theoretic framework

We developed a quantitative covert channel analysis based on Shannon's information theory [Sha48]. We recall the necessary basics from information theory in Section 3.1 (based on [CT91]) before we introduce a formal model of the communication via interrupt-related covert channels. This model provides the information-theoretic framework for our quantitative analysis in later sections.

3.1. A primer on information theory

In this article, we consider discrete random variables, i.e., random variables with a finite or countable range. The probability that a random variable X has value x is denoted by $p(X = x)$ or, if the random variable is clear from the context, by $p(x)$. We use the symbol X also to denote the set of possible values for the random variable X. As a convention, we use X and Y to denote random variables.

The *entropy of* X (denoted by $H(X)$) is a measure for the amount of information contained in X. It is defined by

$$H(X) = -\sum_{x \in X} p(x) * \log(p(x)),$$

where $\log$ denotes the logarithm with base 2 (like in the rest of the article). The entropy $H(X)$ constitutes a lower bound on the average number of bits needed to represent a value of X. Consider, for instance, a representation where $b(x)$ denotes the bit-string representing a value x. Then the number of bits needed to represent a value of X on average can be calculated by $\sum_{x \in X} p(x) * |b(x)|$ where $|b(x)|$ denotes the length of $b(x)$, i.e., the number of bits. In general, we have $H(X) \leq \sum_{x \in X} p(x) * |b(x)|$ and $H(X) \leq \log(|X|)$. One can also determine an upper bound on the length under an optimal encoding: there always is an encoding $(b(x))_{x \in X}$ such that $\sum_{x \in X} p(x) * |b(x)| < H(X) + 1$ (see [CT91]).

The *conditional entropy of* X *given* Y (denoted by $H(X|Y)$) is a measure for the amount of information contained in X if the value of Y is known. It is defined by

$$H(X|Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) * \log(p(x|y)),$$

where $p(x, y)$ is a short-hand notation for $p(X = x \wedge Y = y)$, i.e. the joint probability of X having value x and Y having value y. Moreover, $p(x|y)$ is a short-hand notation for $p(X = x \mid Y = y)$, i.e. the conditional probability of X having value x given that Y has value y. Generally, we have $0 \leq H(X|Y) \leq H(X)$. If the value of Y completely reveals the value of X then $H(X|Y) = 0$. In the other extreme case, the value of Y does not reveal anything about the value of X, and $H(X|Y) = H(X)$ holds.

The *mutual information of* X *and* Y (denoted by $I(X;Y)$) is a measure for the amount of information about the value of X that is revealed by learning the value of Y. It can be calculated by $H(X) - H(X|Y)$. For instance, if X is uniformly distributed and takes values from 0 to 255 then a straightforward binary encoding is optimal, where each value is encoded by a bit-string of length 8. If Y equals the lowest two bits of X then $I(X;Y) = 2$. Note that the notion is symmetric, i.e., $I(X;Y) = I(Y;X)$.

A *memoryless discrete channel* C is a triple (I_C, O_C, P_C) where I_C is the input alphabet (i.e., the set of values that can be sent on C), O_C is the output alphabet (i.e., the set of values that can be received from C), and $P_C = (p(o|i))_{i \in I_C, o \in O_C}$ is the probability matrix where $P_C[i, o] = p(o|i)$ is the probability that the output equals o given that the input equals i.

The *capacity of the channel* C (denoted by $CAP(C)$) is an upper bound on the amount of information (in number

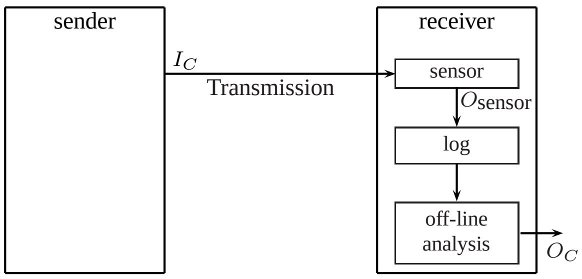


Figure 1. An interrupt-related channel

of bits) that can be transmitted over C on average with an arbitrarily small error probability [CT91]. It is defined by

$$CAP(C) = \max I(X;Y),$$

where X is a random variable with range I_C, Y is a random variable with range O_C, and the maximum is computed over all possible probability distributions for X. Here, the probability distribution for Y is completely determined by the probability distribution for X and P_C:

$$p(Y=o) = \sum_{i \in I_C} p(X=i) * P_C[i,o]$$

For instance, a channel C with $I_C = O_C = \{0,\ldots,255\}$, $P_C[i,o] = 1$ for $i = o$, and $P_C[i,o] = 0$ for $i \neq o$ has the capacity $\mathit{CAP}(C) = 8$. In contrast, if $P_{C'}[i,o] = 1$ for $o = 255$ and $P_{C'}[i,o] = 0$ for $o \neq 255$, then $\mathit{CAP}(C') = 0$. The first channel perfectly transmits the input, while the second does not transmit any information at all.

3.2. Modeling the transmission

In order to transmit information via an interrupt-related channel, the sender performs operations that initiate asynchronous interrupts with the intention of influencing the receiver's behavior such that the message can be decoded. The receiver measures any delays that occur during his time-slots and attempts to reconstruct the original message from these measurements. Obviously, it is in the interest of the receiver, both to maximize the precision of the measurement and to maximize the reliability of the analysis. Unfortunately, these goals are conflicting because any processing time that the receiver dedicates to the analysis is lost for measurements. The solution is to perform the analysis off-line (as indicated in Figure 1). In our bandwidth analysis, we assume this worst-case scenario.

In the remainder of this section, we make a number of simplifying assumptions. We assume that the number of active processes is fixed. Moreover, we assume a Round-Robin scheduler ensuring that the starting time and length for each time-slot is fixed. Consequently, quantum-time and interquantum channels are prevented (see Examples 1 and 2). The input that a sender can provide to the channel are the points in time during a time-slot when he performs interrupt-initiating operations. The output is the delay that the receiver measures during his time-slot. The channel itself is modeled as a memoryless discrete channel. Here, we measure delays, and time in general, in abstract time units, where executing an operation requires at least one time unit. Moreover, we assume the transmission to be noiseless (a proper treatment of noise will be integrated in Section 5).

In the remainder of this section, we detail our model of the transmission via interrupt-related covert channels.

The sender. The input to the channel corresponds to the sender's operations that cause later occurrences of asynchronous interrupts. Formally, an input can be viewed as a set $\{t_1,\ldots,t_k\}$ where each element t_j represents an interrupt-initiating operation, and t_j equals the time when the respective operation is performed. All times are calculated relatively to the starting point of the given time-slot. That is, $0 \leq t_j < s$ holds for all $j \in \{1,\ldots,k\}$, where s equals the duration of the time-slot. The input alphabet is

$$I_C = \{A \mid A \subseteq \{0,\ldots,s-1\}\} .$$

For notational convenience, we alternatively represent the input as a list $[t_1,\ldots,t_k]$, implicitly assuming that the elements occur in ascending order (i.e., $j < j' \Rightarrow t_j < t_{j'}$).

Note that the input alphabet I_C constitutes a worst-case scenario. In reality, the sender might not be able to generate all elements in the input alphabet. For instance, if it requires more than one time unit to execute an interrupt-initiating operation then inputs of the form $[\ldots,t,t+1,\ldots]$ cannot be generated. Moreover, there may be an upper bound on the number of interrupt-initiating operations that can be performed during a time-slot. For instance, the sender's execution is stopped after performing a synchronous/blocking hard-disk access until the corresponding interrupt occurred and was handled. With such operations, the sender can only generate either the empty set or a singleton set as input. In contrast, asynchronous/non-blocking I/O allows the sender to generate input with more than one element [BC06].

The receiver. During the transmission, the sensor repeatedly records the points in time in a log (in RAM) while it is running. The log provides the input to a later off-line analysis, which attempts to re-construct the delays caused by interrupts. A sensor output can be viewed as a set $\{t_1,\ldots,t_k\}$ where each element t_i is a point in time that the sensor recorded. This output alphabet equals the input alphabet:

$$O_{\text{sensor}} = I_C ,$$

but the interpretation differs. For instance, an off-line analysis of the output $[0, 1, 2, 6, 7]$ would reveal the delay two time units after the start of the time-slot. This delay can be explained by the occurrence of interrupts, where the number of interrupts can be calculated from the measured delay and the time required for handling a single interrupt.

Note that our sensor-output alphabet, again, constitutes a worst-case scenario. Typically, many elements of the output alphabet will be infeasible. For instance, if interrupt handling requires 3 time units, the output $[0, 1, 2, 6, 7]$ is possible, while $[0, 1, 2, 4, 6, 7]$ is impossible. Moreover, if the sensor requires more than one time unit for probing the clock and for recording the value in the log then outputs of the form $[\dots, t, t + 1, \dots]$ are impossible. More generally, some interrupts might be disguised if the sensor can measure time only coarsely. In fact, reducing the clock resolution is one possible countermeasure against interrupt-related covert channels (see Section 4).

We assume that the output of the off-line analysis is the cumulative delay during each time-slot (i.e., the sum of all time units during which the receiver was not active):

$$O_C = \{0, \dots, s\} ,$$

where s is the duration of the receiver's time-slot. We needed such a simplification to make the detailed analysis in Sections 5 and 6 technically feasible. However, this is clearly not a worst-case scenario because some information in the sensor's log is not exploited (like, e.g., when the delays occurred during the time-slot). Nevertheless, our analysis of the channel and of the countermeasures (here and in Section 6) reveals a number of interesting insights and, we believe, also gives a fairly realistic impression of the threat and the effectiveness of the mechanisms.

Moreover, if the output of the off-line analysis is, indeed, limited to the cumulative delay then the output alphabet O_C constitutes a worst-case scenario. In reality, some elements of O_C might be impossible (like for O_{sensor}).

The channel. The transmission is modeled by a memoryless discrete channel (see Section 3.1). The probability matrix determines the likelihood that an output $o \in O_C$ occurs given that the sender transmitted an input $i \in I_C$. Conversely, the matrix can be used to determine the probability that $i \in I_C$ was sent given that $o \in O_C$ is received.

Example 4 *If each interrupt-initiating operation in the sender's time-slot results in an interrupt in the receiver's time-slot and each such interrupt induces a delay of c time units, then one obtains the following probability matrix:*

$$p(o|i) = \begin{cases} 1 & \text{, if } o = |i| * c, \\ 0 & \text{, otherwise.} \end{cases}$$

However, if an interrupt occurs only within the receiver's time-slot if the interrupt-initiating operation was performed at most t time units after the start of the sender's time-slot, then one obtains the following probability matrix:

$$p(o|i) = \begin{cases} 1 & \text{, if } o = |\{x \in i \mid x \leq t\}| * c, \\ 0 & \text{, otherwise.} \end{cases} \qquad \Diamond$$

The threat. Interrupt-related covert channels constitute a threat that should not be neglected when constructing and evaluating security-critical systems. The following example illustrates that a 12-bit PIN (i.e. the order of magnitude used in authentication mechanisms for banking machines) can be easily transmitted in approximately one second.

Example 5 *Assume that the Round-Robin scheduler schedules a receiver's time slot always immediately after a time-slot of the sender. We assume a length of 100ms for a time-slot and that one other process is active in addition to the sender and the receiver (e.g., a system-level process). The sender uses asynchronous/non-blocking I/O for accessing the hard disk in order to generate interrupts where a hard-disk access shall require exactly 10ms and handling an interrupt shall take exactly 1ms (see Sections 5.1 and 5.2 on how such strict assumptions can be relaxed).*

For simplicity, we use the following alphabets:

$$I_C = \{A \mid A \subseteq \{0, \dots, 99\}\} \qquad O_C = \{0, \dots, 99\}$$

That is, here, time units correspond to milliseconds. Due to the access time of the hard disk, an access by the sender leads to an interrupt in the receiver's time-slot only if it is performed within the last 10ms of the sender's time-slot. This leads to the following probability matrix:

$$p(o|i) = \begin{cases} 1 & \text{, if } o = |\{x \in i \mid x \geq 90\}|, \\ 0 & \text{, otherwise.} \end{cases}$$

Based on Section 3.1, one can calculate that the bandwidth of the channel is at least 11.5 bits per second.[2] $\Diamond$

4. Possible countermeasures

In the following, we present six mechanisms that can be used against interrupt-related channels. These mechanisms are implemented in some operating systems, but they were not originally intended as a countermeasure against interrupt-related covert channels. Here, we describe how each mechanism works, postponing an evaluation to Section 6.

[2]Let X and Y be random variables on I_C and O_C, respectively. Assume that no interrupt-initiating operations are performed in the first 90ms of the sender's time-slot. Let X capture the number of interrupt-initiating operations in the last 10ms of the sender's timeslot. If X is uniformly distributed then Y is uniformly distributed on $\{0, \dots, 10\}$. Hence, we have $I(X;Y) = I(Y;X) = H(Y) - H(Y|X) = \log(11)$. From $CAP(C) = \max I(X;Y)$, we obtain $CAP(C) \geq \log(11)$, which refers to the capacity for a single time-slot of the receiver. As, on average, there are $3\frac{1}{3}$ time-slots of the receiver per second, at least $3\frac{1}{3} * \log(11) \equiv 11.5$ bits can be transmitted per second.

Fuzzy time and *reduced clock resolution* are mechanisms against covert timing channels, in general. Both mechanisms reduce the bandwidth by modifying how processes perceive time. These mechanisms are effective as all interrupt-related channels are also timing channels. *Polling* can be used to replace interrupt-driven communication, but this is, in general, not acceptable (as discussed in Section 2). However, there are cases where polling is a viable solution, e.g., in some real-time systems polling is applied in order to ensure reliable upper bounds on latency times. *Interrupt-initiated polling*, *interrupt-rate limiting*, and *strict software scheduling* all impose constraints on the number of interrupts that can occur in a time-slot. These three mechanisms are usually employed for preventing receive livelocks.[3]

4.1. Altering the perception of time

The exploitation of covert timing channels closely depends on the ability to perceive time. It is the nature of timing channels that they cannot be exploited without having access to a system clock or some other source of timing information. However, it is usually unrealistic to prevent the access to all possible sources of timing information (including, e.g., access to internal system clocks, clocks attached to the network, and interleavings of event sequences that could, otherwise, serve as relative clocks [Wra91]).

Fuzzy time provides a solution to mitigate timing channels while still giving processes access to the system clock [Hu91]. This system clock, however, measures time imprecisely where the duration of each clock tick is randomly distributed around some average value. If a clock tick takes, e.g., $10\mu s$ on average then the duration of a clock tick, e.g., could be uniformly distributed between $5\mu s$ and $15\mu s$. The VAX security kernel is an example of an operating system that incorporates a fuzzy time mechanism [KZB+90].

One problem with fuzzy time is that, over time, the imprecise system clock could deviate too much from real time. However, this problem can be overcome by synchronizing the imprecise clock at predefined intervals with a precise clock (which is not directly accessible by processes).

There are fundamental limits on how far one can reduce the bandwidth of timing channels by a fuzzy time mechanism (see Section 6 for a detailed analysis). In particular, it cannot eliminate interrupt-related channels completely.

Reduced clock resolution is a mechanism that also alters the perception of time but does not share this limitation. Instead of altering the behavior of the accessible system clock, one reduces the resolution of timing information. This gives processes a coarser perception of the system time and thereby reduces their capabilities to exploit timing channels. Implementing this solution is fairly straightforward as one only needs to provide variants of system calls that return timing information with the given resolution.

[3]A receive livelock occurs if a device generates so many interrupts that interrupt handling takes up all CPU time. For instance, if a network card receives too many packets from a network then it may cause so many interrupts for forwarding the packets into a buffer that no processing time remains for properly processing the packets that are already in the buffer.

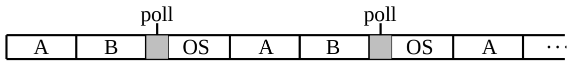
4.2. Change of communication paradigm

Besides mitigating interrupt-related covert channels, our other goal is to retain the main advantages of interrupt-driven communication. However, the latter goal becomes obsolete if there are already other reasons for using polling-based communication with hardware devices.

Polling can be used to ensure maximum latency times for hardware device. Some real-time kernels (e.g., [The07]) offer a polling mode for this purpose. In polling mode, the operating system repeatedly checks whether the various devices need any attention. In order to ensure a maximal latency for a given device, the system must check this device in shorter intervals than the maximal latency time.

Polling fundamentally differs from interrupt-driven communication where devices actively inform the operating system by interrupts that they need attention. If a device is not checked frequently enough then there is a danger of data being lost (e.g., if the buffer on a network card is too small for storing all packets that arrive). If such dangers shall be avoided then the intervals in which the device is checked must not only be shorter than the maximal latency time but also shorter than the minimal time needed to loose data.

Some care is needed in the implementation of polling, if it shall not be the source of other covert channels. In particular, the duration of polling should not have any influence on application processes. This can be achieved, by dedicating fixed parts of the time-slots of operating system processes to polling as illustrated in the following diagram:

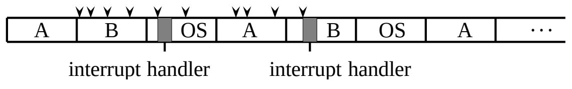


When determining the size of the slots for polling, one must take into account the execution times of the relevant device drivers. Moreover, these slots themselves constitute shared resources (as polling occurs on a per device basis and not on a per process basis) and, hence, one must ensure that they cannot be the source of additional covert channels.

4.3. Selective processing of interrupts

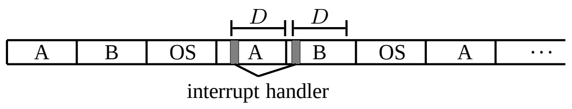
In interrupt-driven communication, the CPU is traditionally interrupted each time some device needs attention. A more selective processing of such situations can reduce the bandwidth of interrupt-related channels. On the implementation level, this can be achieved by creating interrupts more selectively at the devices or, alternatively, by filtering interrupts using the masking mechanism of the operating system.

Interrupt-rate limiting delays interrupts until certain conditions are met. For instance, a network card would delay interrupts until a certain number of packets has arrived (rather than requesting an interrupt for each packet). This is illustrated in the following diagram (where each arc on top of the time-slots indicates the arrival of a packet):

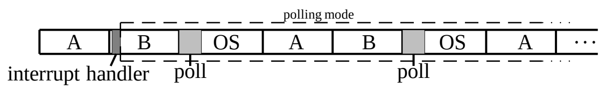


Interrupt-rate limiting is, e.g., implemented in the Intel 82540EM Gigabit Ethernet Controller as a countermeasure against receive livelocks [Cor03]. It indeed protects against this threat because, if too many network packets arrive then the network card overwrites old packets in its buffer, without needing any processing time of the CPU.

Strict software scheduling is another mechanism for avoiding receive livelocks [RD05]. The mechanism enforces a minimal time span D between any two successive interrupts from the same device. This is implemented by masking interrupts from a given device for D time units after this device issued an interrupt request. If the mask has not yet been reset then further interrupt requests from this device are dropped or, alternatively, held back by the interrupt controller as illustrated in the following diagram:



Interrupt-initiated polling masks interrupts from a given device immediately after this device caused an interrupt, and the operating system switches to polling mode for this device. This is illustrated in the following diagram:



To avoid the disadvantages of polling-based communication, the system switches back to interrupt mode after a given time period or when some other condition is met.

In [MR96], interrupt-initiated polling is proposed as a countermeasure against receive livelocks. To be effective, the mechanism must be combined with an implementation of the polling mode that avoids livelocks. A similar mechanism can be found in the Linux kernel (the NAPI, [SO01]).

5. Refining the information-theoretic model

Before evaluating the countermeasures wrt. their effectiveness against interrupt-related channels, we move to a more realistic scenario. In Section 3.2, we introduced an information-theoretic model of the transmission that leaves out of consideration that the communication could be disturbed by noise. In this section, we refine the model considering three possible sources of noise: varying response times of devices, varying duration of interrupt handling, and interrupts caused by other processes than the sender.

Fortunately, the refinement does not require any fundamental changes of the framework. A noisy interrupt-related channel is, again, modeled as a memoryless discrete channel $C = (I_C, O_C, P_C)$. Formally, we define the probability matrix P_C by the multiplication of two matrices Q_C and R_C (i.e. $P_C = Q_C \times R_C$). Consequently, the rows of Q_C are indexed by the input alphabet I_C and the columns of R_C by the output alphabet O_C of the channel. Both alphabets are taken from Section 3.2, i.e., $I_C = \{A \mid A \subseteq \{0, \dots, s-1\}\}$ and $O_C = \{0, \dots, s\}$. For accessing the columns of Q_C as well as the rows of R_C, we use a finite index set K_C.

More concretely, we use Q_C to specify the probability that a certain number $k \in K_C$ of interrupts occurs during the receiver's time-slot given that the sender performed interrupt-initiating operations at the times specified by $i \in I_C$. The matrix R_C specifies the probability that a delay $o \in O_C$ occurs during a time slot of the receiver given that a certain number $k \in K_C$ of interrupts occurred in this time-slot. The probability that a delay $o \in O_C$ is caused given that the input equals $i \in I_C$ can be determined by:

$$\begin{aligned} p(o|i) &= \textstyle\sum_{k=0}^{k=|K|} p(k|i) * p(o|k) \\ &= \textstyle\sum_{k=0}^{k=|K|} Q[i,k] * R[k,o] \\ &= (Q \times R)[i,o] = P[i,o] \end{aligned}$$

The following example illustrates that and how noiseless channels can be represented in this refined framework.

Example 6 *Assume that each interrupt-initiating operation in the sender's time slot causes one interrupt in the receiver's time-slot and that no other interrupts occur. Moreover, let the time needed for handling a single interrupt be a constant c. The resulting probability matrices are:*

$$\begin{aligned} Q[i,k] &= p(k|i) = \begin{cases} 1 & \textit{if } k = |i|, \\ 0 & \textit{if } k \neq |i| \end{cases} \\ R[k,o] &= p(o|k) = \begin{cases} 1 & \textit{if } o = c * k, \\ 0 & \textit{if } o \neq c * k \end{cases} \end{aligned} \qquad \Diamond$$

In the remainder of this section, we explain how aspects of noise can be incorporated into our framework. The three refinements presented are compatible and can be used in combination with each other.

The probabilistic characterization of noise depends on the particular system and on the system load. For a reliable bandwidth analysis, the probability distribution must reflect reality as precisely as possible. However, the precise probability distribution is usually rather difficult to determine analytically for a complex system. Therefore, an experimental element seems unavoidable in the analysis. Our approach is to select some parameters for each possible source of noise. For a concrete system, these parameters can be determined by an experimental evaluation. In our formal treatment of noise, we just assume that the respective parameters are given and base the analysis on the probability distribution with the maximal entropy (among the distributions satisfying the given parameters). This approach is known as the *principle of maximum entropy* [Jay57, CT91].

5.1. Noise: varying response times

The response times of hardware devices are not constant, but rather vary around some average value.[4] We use a random variable ΔT to characterize the duration between performing an interrupt-initiating operation and the corresponding interrupt. Let $\overline{\delta}$ be the average value of ΔT and σ^2 be the variance of ΔT. Given $\overline{\delta}$ and σ^2, the normal distribution (denoted by $N(\overline{\delta}, \sigma)$) maximizes the entropy (see [CT91, chapter 11]). Following the principle of maximal entropy, we assume that ΔT is $N(\overline{\delta}, \sigma)$-distributed.

That is, the probability that performing an interrupt-initiating operation at time t results in an interrupt between time a and time b can be calculated by

$$p(a \leq t + \Delta T \leq b) = \Phi\left(\frac{b - t - \overline{\delta}}{\sigma}\right) - \Phi\left(\frac{a - t - \overline{\delta}}{\sigma}\right),$$

where Φ is the cumulative distribution function of the normal distribution with parameters 0 and 1. The entries of Q_C are calculated as the sum of all probabilities that k interrupts occur in the receiver's time-slot (i.e., between a and b):

$$\begin{aligned} p(k|i) \quad = \quad & \sum_{A \subseteq i, |A| = k} \left(\prod_{t \in A} p(a \leq t + \Delta T \leq b)\right) \\ & * \left(\prod_{t \in i \setminus A} 1 - p(a \leq t + \Delta T \leq b)\right). \end{aligned}$$

For a given system, the parameter $\overline{\delta}$ depends on the particular hardware devices accessed by the interrupt-initiating operation, e.g., on the access time of a hard disk. For hard disks, values between 5 ms and 15 ms are common (e.g., 12.6 ms for the Seagate Barracuda 7200.7 [LLC05]). Access times for CD-ROM drives are much higher (about 200 ms to 400 ms). If access times have been obtained for a device, the variance σ^2 can be obtained, e.g., by a maximum likelihood estimation ([Kay93]).

[4]The variation depends on many parameters. In particular, it can be influenced by the sender. For instance, the sender can select specific disk sectors for the disk accesses to keep the variations small.

5.2. Noise: varying duration of handling

The time needed for handling interrupts is not constant in reality, but is more adequately characterized by a discrete random variable HT. Let the minimal, maximal, and average duration needed for interrupt handling be denoted by t_{min}, t_{max}, and $\overline{t}$, respectively. Following the principle of maximal entropy, the probability distribution is specified by

$$p(HT = t) = \frac{\beta^t}{\sum_{j=t_{min}}^{t_{max}} \beta^j},$$

where $0 < \beta$ (see [CT91]). The value of β is uniquely determined by $\overline{t}$ as $\sum_{t=t_{min}}^{t_{max}} t * p(HT = t)$ is monotonically increasing in β. The entries of R_C can be calculated by

$$p(o|k) = \sum_{\substack{(t_1, \ldots, t_k) \\ t_1 + \ldots + t_k = o}} p(HT = t_1) * \ldots * p(HT = t_k).$$

This source of noise reduces the capacity as it makes it harder for the receiver to guess the number of interrupts in a time-slot based on the cumulative delay that he observes. A delay of, e.g., eight time units could be explained by two interrupts with a duration of four time units as well as by four interrupts with a duration of two time units.

5.3. Noise induced by other processes

In reality, there will usually be further active processes besides sender and receiver, and these processes might also perform interrupt-initiating operations. We refer to the resulting interrupts as *noisy interrupts* because they disturb the covert communication between sender and receiver.

We use a discrete random variable N to characterize the number of noisy interrupts in the receiver's time-slot. Let $\overline{n}$ be the average number of noisy interrupts in a time-slot of the receiver. We assume that the probability that a noisy interrupt occurs at a given point in time is independent from this particular point in time as well as from the history of prior interrupts. Under these assumptions, the distribution of N is approximated by a Poisson distribution with parameter $\overline{n}$ (see [Bul79]), i.e., $P(N = k) = e^{-\overline{n}} * \overline{n}^k / k!$.

The probability that k interrupts occur in the receiver's time-slot, given that $k' \leq k$ of interrupt-initiating operations lead to interrupts in this time-slot of the receiver, equals the probability that $k - k'$ noisy interrupts occur, i.e., $P(N = k - k')$. The entries of the matrix Q_C can be calculated by (where $p'(k|i)$ refers to the entries of the original matrix Q_C and $p(k|i)$ to the entries of the modified matrix taking noisy interrupts into account):

$$p(k|i) = \sum_{k'=0}^{k} p'(k'|i) * P(N = k - k').$$

Note that this treatment of noisy interrupts can be used in combination with our treatment of varying response times (see Section 5.1) as it is specified as a transformation on the matrix Q_C (rather than as an absolute definition of Q_C).

We measured the average number of interrupts on a Debian Linux System under normal system load. Clock interrupts are requested regularly, and their frequency is therefore known to the sender and receiver. For this reason we did not count them in our measurement. We obtained an average number of four interrupts per 100 milliseconds.

6. Evaluation of countermeasures

We evaluate the mechanisms from Section 4 with respect to their potential to mitigate interrupt-related channels. Our information-theoretic framework serves as the basis of this quantitative analysis. Before analyzing the mechanisms, we show how each mechanism can be characterized in the framework from Section 5 .

6.1. Embedding into the framework

Fuzzy time causes deviations of the system clock from real time. We use a random variable *ERR* to capture the deviation of a given clock tick. Given a maximal deviation K, *ERR* is uniformly distributed in the interval $\{-K, \dots, K\}$. Since the deviation for a given clock tick is chosen independently from the deviation for previous clock ticks, the maximal deviation increases over time (e.g., after 5 clock ticks, the maximal deviation would be $5 * K$). For limiting the maximal deviation, the fuzzy clock is periodically synchronized with a precise clock. Obviously, these re-synchronizations reduce the effectiveness of the fuzzy time mechanism against timing channels. For analyzing the effectiveness of the mechanism against interrupt-related channels, we make the worst-case assumption that a re-synchronization occurs immediately before each clock tick that is relevant for the analysis. The delay caused by fuzzy time in the receiver's time-slot can then be characterized as the sum of two random variables (one for the starting point and one for the endpoint of the time-slot) that are each uniformly distributed on $\{-K, \dots, K\}$, i.e. by $\mathit{ERR}+(-\mathit{ERR})$. We characterize the fuzzy time mechanism in the framework from Section 5 by specifying the matrix R_C (where $p(o|k)$ refers to the entries of R_C without fuzzy time and $p_{fuz}(o|k)$ refers to the entries with fuzzy time):

$$p_{fuz}(o|k) = \sum_{j_1, j_2 = -K}^{+K} \begin{pmatrix} p(\mathit{ERR}{=}j_1) * p(\mathit{ERR}{=}j_2) \\ * \, p(o - j_1 - j_2 | k), \end{pmatrix}$$

where we assume $p(o{-}j_1{-}j_2|k) = 0$ if $o{-}j_1{-}j_2 < 0$.

Reduced clock resolution has the effect of making time measurements less accurate than with full clock resolution. E.g., if the time resolution is ten time units, the measured cumulative delay is a multiple of ten. In this case we assume that, instead of time t, the receiver measures the time $10 * \lfloor t/10 \rfloor$. I.e., the measured time is rounded off to the next multiple of ten. We characterize the mechanism in the framework by modifying the matrix R_C (where $p(o|k)$ denotes the probabilities with full clock resolution and $p_{red}(o|k)$ denotes them with reduced clock resolution):

$$p_{red}(o|k) = \begin{cases} \sum_{j=0}^{9} p((o{+}j)|k) & \text{if } o \equiv 0 \mod 10, \\ 0 & \text{otherwise.} \end{cases}$$

Other clock resolutions can be characterized analogously.

Polling prevents occurrences of asynchronous interrupts, which is characterized by changing the matrix Q_C:

$$p(k|i) = \begin{cases} 0 & \text{if } k > 0 \\ 1 & \text{if } k = 0 \end{cases}$$

Interrupt-rate limiting reduces the number of interrupts that a device generates. For our analysis, we assume a mechanism that generates one interrupt for every chunk of v interrupts that would otherwise be requested, where v is some constant. The handling of a given interrupt might take longer with this mechanism (e.g., when a network card issues an interrupt, v packets instead of 1 packet must be handled), however, we ignore such aspects in our analysis.

A subtlety that our analysis addresses in more detail is that there is some uncertainty about the number of interrupts that are already pending at the given device. This number determines how many further interrupt situations are needed before an actual interrupt is forwarded to the CPU. We assume that the number of pending interrupts is captured by a random variable PEN that is uniformly distributed on $\{0, \dots, v{-}1\}$ (i.e., $p(\text{PEN}{=}w) = \frac{1}{v}$ for $w \in \{0, \dots, v{-}1\}$).

The mechanism is characterized in the framework from Section 5 by modifying the matrix Q_C (where $p(k|i)$ refers to the entries of Q_C without interrupt-rate limiting and $p_{irl}(k|i)$ refers to the entries with interrupt-rate limiting):

$$p_{irl}(0|i) = \sum_{j=0}^{v-1} \frac{1}{v} * \left(\sum_{l=0}^{v-j-1} p(l|i) \right)$$

and (for $k > 0$)

$$p_{irl}(k|i) = \sum_{j=0}^{v-1} \frac{1}{v} * \left(\sum_{l=k*v-j}^{(k+1)*v-j-1} p(l|i) \right).$$

Strict software scheduling enforces a gap of D time units between two successive interrupts. Consequently, if a time-slot's length is s then at most $\lfloor s/D \rfloor + 1$ interrupts can occur in it. Note that the number of interrupts that occur with strict software scheduling might be strictly below this upper bound, even if more than $\lfloor s/D \rfloor + 1$ interrupts would occur without strict software scheduler. For instance, if all interrupts would occur in the second half of the time-slot then $\lceil \frac{\lfloor s/D \rfloor + 1}{2} \rceil$ would be an upper bound on the number of interrupts that occur with a strict software scheduler. For analyzing the effectiveness of the mechanism, however, we make the worst-case assumption that, if less than $\lfloor s/D \rfloor + 1$ interrupts would occur without strict software scheduling then all these interrupts also occur with strict software scheduling.

We characterize the mechanism in our framework by modifying the matrix Q_C (where $p(k|i)$ refers to the entries of Q_C without strict software scheduling and $p_{sss}(k|i)$ refers to the entries with strict software scheduling):

$$p_{sss}(k|i) = \begin{cases} p(k|i) & \text{if } k < \lfloor s/D \rfloor + 1, \\ \sum_{k=\lfloor s/D \rfloor+1}^{K} p(k|i) & \text{if } k = \lfloor s/D \rfloor + 1, \\ 0 & \text{otherwise.} \end{cases}$$

Interrupt-initiated polling causes the system to switch to polling mode after an interrupt occurred. In polling mode, interrupts from the given device are masked, and the operating system is now responsible for repeatedly checking whether the device needs attention. After remaining in polling mode for some time, the system may switch back to interrupt mode in a later time-slot. For analyzing the effectiveness of the mechanism, we make the worst-case assumption that the system switches to interrupt mode before each time-slot of the receiver.

We characterize the mechanism in our framework by modifying the matrix Q_C (where $p(k|i)$ refers to the matrix without interrupt-initiated polling and $p_{iip}(k|i)$ to the matrix with interrupt-initiated polling):

$$p_{iip}(k|i) = \begin{cases} p(0|i) & \text{if } k = 0 \\ \sum_{k>0} p(k|i) \, (= 1 - p(0|i)) & \text{if } k = 1 \\ 0 & \text{if } k > 1 \end{cases}$$

Note that the probability is 0 for $k > 1$ as the first interrupt in the receiver's slot causes a switch to polling mode.

Combinations of different mechanisms can be characterized by combining the transformations presented in this section. However, such a straightforward combination relies on that mechanisms do not disturb each other. One needs to check to which extent this assumption is satisfied in reality before applying this solution. In comparison, we expect an analysis to be much more involved if one investigates combinations of mechanisms that are not independent.

6.2. Numerical evaluation

We evaluate the mechanisms by computing the capacity of the channel for each of them. For such a computation, one needs to fix some parameters of the channel. We consider two scenarios, one without noise and one with noise.

In the first scenario, we use the values from Example 5. The second scenario is as follows: the receiver's time-slots are scheduled directly after the sender's time-slots, where all time-slots have a length of 100ms. An interrupt is handled on average 10ms after the corresponding interrupt-initiating operation was performed, while approximately 80% of the requests are handled between 5ms and 15ms after the interrupt-initiating operation occurred. The handling of an interrupt requires between 1ms and 4ms, with an average of 2.5ms. Additionally, on average 2 noisy interrupts occur in a time-slot of the receiver. In summary, we obtain $a = 100$, $b = 200$, $\overline{\delta} = 10$, $t_{min} = 1$, $t_{max} = 4$, $\overline{t} = 2.5$, and $\overline{n} = 2$. Moreover, $\sigma = 4$ follows from the assumption that 80% of the interrupts are handled between 5ms and 15ms after their initiation.

We analyze both scenarios for two senders. Sender A performs up to four interrupt-initiating operations in a time-slot. Hence, his input alphabet consists of the subsets of $\{0, \ldots, 99\}$ that contain at most four elements. Sender B performs up to one interrupt-initiating operation. Hence, his input alphabet is $\{\{\}, \{0\}, \ldots, \{99\}\}$. The receiver behaves as described in Section 3.2.

Due to the complex probability matrices arising in this context an analytical derivation of the capacity is no longer tractable. More precisely, the problem is the maximization over all probability distributions in the definition of $\mathit{CAP}(C)$ (see Section 3.1). Therefore we employ an approximative method. For our numerical computations, we use an algorithm due to Blahut ([Bla72]) and Arimoto ([Ari72]). Computing the approximative capacities for the channel with this algorithm for sender A, we obtain 2.32 bits and 0.33 bits in the first and second scenario, respectively. For sender B we obtain 1 bit and 0.1 bits, respectively.

Figure 2 illustrates the results of our numerical computations. The diagrams indicate how the choice of a mechanism's parameter affects the channel bandwidth. In each column, the upper diagram refers to Scenario 1 while the lower diagram refers to Scenario 2. The solid line refers to sender A and the dashed line to sender B. In the following, we discuss the insights gained from our numerical analysis.

Fuzzy time reduces the capacity towards 0 for high values of K, without reaching 0. A given non-zero capacity can be enforced by an appropriate value for K. For sender B, e.g., a maximal deviation of $K = 3$ suffices to push the capacity below 10^{-5} bits. To achieve the same reduction for sender A, the value of K must be three times as high.

Reduced clock resolution lowers the capacity to zero for sufficiently coarse clock resolutions. However, reducing the clock resolution decreases the capacity only slowly. To reduce the capacity by 50% the clock resolution must be lowered by a factor of ten. The discontinuities in the noiseless

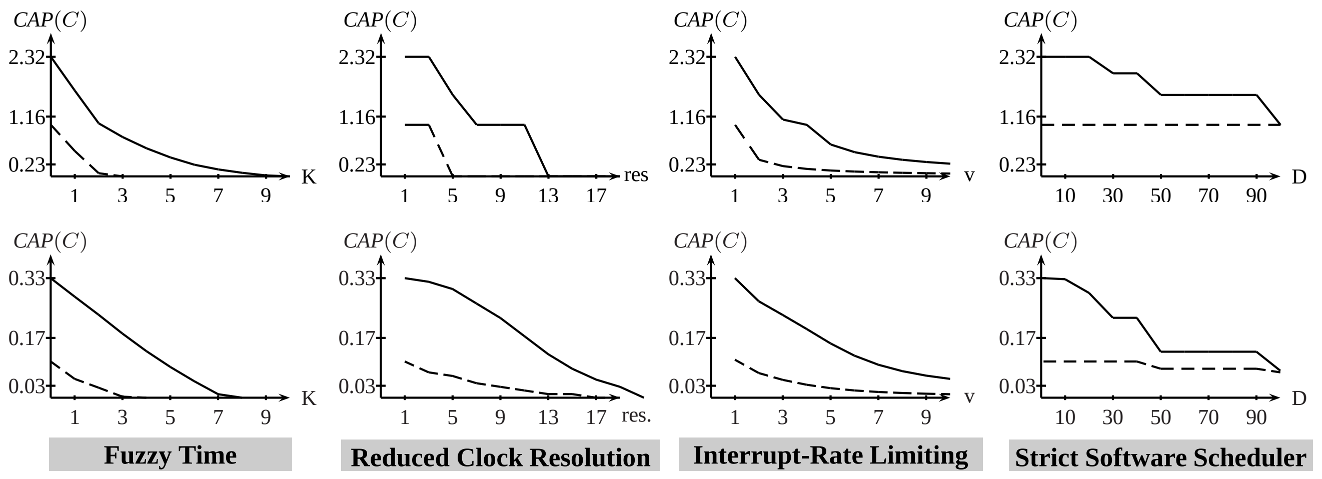


Figure 2. Results of numerical computation

case (see Figure 2) might stem from the phenomenon that, for certain clock resolutions, differences between some delays can no longer be distinguished. Moreover, by numerically evaluating some further scenarios, we learned that the longer the handling of a single interrupt takes, the more one needs to lower the clock resolution for obtaining the same reduction of the capacity.

Reduced clock resolution and fuzzy time, both make timing data imprecise. Time measurements with fuzzy time and a maximal deviation of K time units are as imprecise as time measurements with a clock whose reduced resolution is $2K$ time units. We found it surprising that, for small values of K, the capacity with fuzzy time is much lower than the capacity with a clock resolution of $2K$ time units.

Polling obviously reduces the capacity to 0.

Interrupt-rate limiting decreases the capacity towards zero for high values of v. The diagrams indicate that if the value of v is large enough then one can obtain a capacity arbitrarily close to 0. However, when increasing the value of v the capacity is decreasing only slowly. If $v = 6$ interrupts are merged then the capacity decreases by 50%. To decrease the capacity by 90% one needs to merge at least $v = 11$ interrupt requests.

Strict software scheduling cannot push the capacity below some threshold. In our example, the threshold is 1 bit in the first scenario and and 0.075 bits in the second scenario (see Figure 2). This also shows that the value of the threshold depends on the amount of noise. Furthermore, the maximal capacity reduction is attained only slowly. For instance, a delay of more than 90 milliseconds between adjacent interrupts is needed for sender A.

Interrupt-initiated polling reduces the capacity to 1 bit in the first scenario and to 0.07 bits in the second scenario for sender A. For sender B, we obtain the same results. This is not by coincidence as at most one interrupt can occur in the receiver's time-slot. Therefore, it is irrelevant whether the sender can initiate one or four interrupt requests in a time-slot. Moreover, if the CPU stays in polling mode for several scheduler rounds before using interrupts again, symbols cannot be transmitted in some scheduling rounds. This is the reason why it becomes possible to obtain a capacity arbitrarily close to zero with this mechanism.

6.3. Recommendations

The numerical analysis in Section 6.2 illustrates the benefits of the various mechanisms with respect to the mitigation of interrupt-related channels. Two mechanisms, polling and reduced clock resolution, make it possible to close such channels completely. Our analysis reveals that three other mechanisms (fuzzy time, interrupt-rate limiting, and interrupt-initiated polling) allow one to lower the capacity arbitrarily close to zero by setting the respective parameter to a particular value. Finally, strict software scheduling cannot reduce the capacity below some given threshold.

Some mechanisms have additional benefits besides mitigating interrupt-related covert channels, but they also create costs (e.g., a performance overhead or a more complex system architecture). Reduced clock resolution and fuzzy time impair the perception of time. They are, therefore, not applicable if applications need precise timing information. An additional benefit of these mechanisms is, however, that they mitigate not only interrupt-related channels, but covert timing channels in general. Interrupt-rate limiting, interrupt-initiated polling, and the strict software scheduler have the additional benefit of also being effective against receive livelocks. From all the mechanisms, only interrupt-rate limiting requires a modification of hardware.

None of the mechanisms is superior to all others. Hence, when choosing a mechanism, one should take the particular scenario into account:

- If the channel shall be closed completely, the alternatives are reduced clock resolution and polling. The former allows one to keep the communication paradigm (whose benefits are discussed in Section 2), while the latter does not disturb time measurements.

- If the capacity need not be reduced to zero, fuzzy time, interrupt-rate limiting, or interrupt-initiated polling are preferable. All of them are adjustable to obtain a given non-zero capacity. Among these mechanisms, fuzzy time has the advantage that interrupt communication need not be changed. However, if accurate timing information is needed, it is no alternative.

- Due to its limited ability to lower the capacity, strict software scheduling is not the first choice to mitigate the channel. However, if it is already used for receive livelock mitigation then it might still be a natural candidate for mitigating interrupt-related channels. However, one needs to verify that the reduction of the channel's capacity suffices one's needs.

7. Related work

Covert channels, firstly identified in [Lam73] , have been researched extensively during the last 30 years. The three main goals (and also research directions) are covert channel identification, bandwidth estimation, and mitigation.

An overview of methods for covert channel identification is given in [Gli93]: syntactic information flow analysis studies specifications or programs equipped with an information-flow semantics (e.g., [Den76, DD77, Mil76]). A more recent technology for controlling information flow are security type systems (see, e.g., [VSI96, SM03]). The shared resource matrix method proposed in [Kem83] identifies attributes of shared resources that can be used for covert communication (see also [HKMY87, Kem02]). A third approach is a noninterference analysis (e.g, [Fei80, GM82]).

A variety of covert channels has been identified, exploiting, e.g., shared CPUs [Hus78], shared hard disks [SGLS77, KW91], or the shared system bus [Hu91]. A scenario in which interrupts are used for covert information transmission is discussed in [Tro98]. In contrast to our scenario, the information is not transmitted intentionally, in fact, information about entered passwords is obtained by measuring the duration of keyboard interrupts.

After a covert channel has been identified, one needs to assess its danger. This is usually achieved by estimating its bandwidth. If the bandwidth is too high, one needs to decrease it or to even eliminate the channel completely. Besides informal methods for estimating the bandwidth (e.g., [TG88]), the focus has been on deriving the capacity of covert channels based on information theory. A number of articles study this problem conceptually, without investigating concrete channels. Millen and Moskowitz [Mil87, Mos90] study the connection between notions of noninterference [GM82, McC88] with the capacity of a channel. Various other articles (e.g., [Mil89, MM94, MGK96]) investigate how the capacity of certain classes of channels can be computed, while [Mos91] and [MM92] study the effects of noise on simple timing channels. In contrast to the types of noise considered in this article, the noise effects in [Mos91] and [MM92] affect the time it takes to transmit a symbol, not the output of the channel itself.

Besides the analysis of abstract channels, concrete channels have been investigated, and mechanisms mitigating the channels have been proposed and analyzed. A reason for this were the Department of Defense Trusted Computer System Evaluation Criteria [TCS85], requiring covert channel analysis from level B2 upwards (the more recent Common Criteria [CC05] require a covert channel analysis from level EAL5 upwards as well). In [Hu91], the so-called bus-contention channel is identified. It allows one to transmit information by modulating the usage of a shared system bus. Fuzzy time is proposed as a countermeasure and evaluated wrt. its effectiveness by measurements on a system. In [Gra93b], Gray proposes another countermeasure based on partitioning the system bus, and performs a rigorous information-theoretical analysis of its effects on the channel capacity. In [Gra93a], the bus-contention channel is formally analyzed in combination with fuzzy time. Another example in which a concrete channel and a countermeasure are analyzed is the NRL-Pump [KM93, KMC05].

To our knowledge there are no articles in which different countermeasures mitigating the same covert channel are analyzed and compared in a uniform formal framework.

8. Conclusion

We presented a collection of countermeasures against interrupt-related covert channels and analyzed their effectiveness in a novel information-theoretic framework. To our knowledge, this is the first general investigation of interrupt-related covert channel analysis and mitigation. It revealed some interesting insights about the threat as well as the countermeasures and also created some open questions and tasks that would be interesting to address in the future.

We were somewhat conservative by focusing on mechanisms for which implementations are available. During our project, also some more speculative solutions came to our mind that have not been implemented yet. One example is to modify interrupt-driven communication such that interrupts can be filtered based on which process performed the interrupt-initiating operation (rather than masking only on a per-device basis). Another possibility would be to introduce a co-processor that performs timing-critical parts of interrupt handling without involving the CPU.

It will be interesting to see how well our information-theoretic framework can be adapted for the analysis of other covert channels and of countermeasures against them. It would also be desirable to verify the analytical results presented here in experiments. The outcome of such an experiment is not completely determined by our analysis results because we made some choices in parameters that seemed plausible to us (often making worst-case assumptions), but that are not the only possible choices. These include, for instance, the choice of the output alphabet (i.e., that the receiver can only see the accumulated delay), the choice of maximal entropy distributions (in Sections 5.1 and 5.2), and the choice of the Poisson distribution (in Section 5.3). Refining the instantiation is another interesting direction for future work, for instance, by changing the input alphabet I_C (e.g., to distinguish different operations depending on the hardware devices involved and on other aspect of the access) or the output alphabet O_C (e.g., to allow the analysis to exploit information about when delays occurred within a time-slot).

Acknowledgments. We thank the anonymous reviewers for their suggestions.

This work was funded by the German Research Association (DFG) in the Computer Science Action Program.

References

[Ari72] S. Arimoto. An Algorithm for Computing the Capacity of Arbitrary Discrete Memoryless Channels. *IEEE Transactions on Information Theory*, 18(1):14–20, 1972.

[BC06] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel*. O'Reilly, 2006.

[Bla72] R. Blahut. Computation of Channel Capacity and Rate-Distortion Functions. *IEEE Transactions on Information Theory*, 18(4):460–473, 1972.

[Bul79] M. G. Bulmer. *Principles of Statistics*. Dover Publications, 1979.

[CC05] Common Criteria for Information Technology Security Evaluation Version 2.3, 2005.

[Cor03] Intel Corporation. Interrupt Moderation Using Intel Gigabit Ethernet Controllers, Application Note (AP-450), Revision 1.1, 2003.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications, 1991.

[DD77] D. E. Denning and P. J. Denning. Certification of Programs for Secure Information Flow. *Communications of the ACM*, 20(7):504–513, 1977.

[Den76] D. E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, 1976.

[Fei80] R. Feiertag. A Technique for Proving Specifications are Multilevel Secure. Technical Report CSL-109, Computer Science Laboratory, SRI International, 1980.

[Gli93] V. Gligor. A Guide to Understanding Covert Channel Analysis of Trusted Systems. CSC-TG-030, Rainbow Series (Light Pink Book), 1993.

[GM82] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[Gra93a] J. W. Gray III. On Analyzing the Bus-Contention Channel Under Fuzzy Time. In *Proceedings of the Computer Security Foundations Workshop*, pages 3–9, 1993.

[Gra93b] J. W. Gray III. On Introducing Noise into the Bus-Contention Channel. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 90–98, 1993.

[HKMY87] 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):157–168, 1987.

[Hu91] W.-M. Hu. Reducing Timing Channels with Fuzzy Time. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 8–20, 1991.

[Hus78] J. C. Huskamp. *Covert Communication Channels in Timesharing Systems*. Technical Report UCB-CS-78-02, University of California, Berkeley, 1978.

[Jay57] E. T. Jaynes. Information Theory and Statistical Mechanics. *Physical Review*, (106):620–630, 1957.

[Kay93] S. M. Kay. *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*. Prentice Hall, 1993.

[Kem83] R. A. Kemmerer. Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels. *ACM Transactions on Computer Systems*, 1(3):256–277, 1983.

[Kem02] R. A. Kemmerer. A Practical Approach to Identifying Storage and Timing Channels: Twenty Years Later. In *18th Annual Computer Security Applications Conference*, pages 109–118, 2002.

[KM93] M. H. Kang and I. S. Moskowitz. A Pump for Rapid, Reliable, Secure Communication. In *ACM Conference on Computer and Communications Security*, pages 119–129, 1993.

[KMC05] M. H. Kang, I. S. Moskowitz, and S. Chincheck. The Pump: A Decade of Covert Fun. In *Proceedings of the 21st Annual Computer Security Applications Conference*, pages 352–360, 2005.

[KW91] P. A. Karger and J. C. Wray. Storage Channels in Disk Arm Optimization. In *IEEE Symposium on Security and Privacy*, pages 52–63, 1991.

[KZB$^+$90] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn. A VMM Security Kernel for the VAX Architecture. In *IEEE Symposium on Security and Privacy*, pages 2–19, 1990.

[Lam73] B. W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615, 1973.

[LLC05] Seagate Technology LLC. Product Manual Barracuda 7200.7 Serial ATA, 2005.

[McC88] D. McCullough. Covert Channels and Degrees of Insecurity. In *Proceedings of the Computer Security Foundations Workshop*, pages 1–33, 1988.

[McH95] J. McHugh. Covert Channel Analysis: A Chapter of the Handbook for the Computer Security Certification of Trusted Systems. NRL Technical Memorandum 5540:080A, 1995.

[MGK96] I. S. Moskowitz, S. J. Greenwald, and M. H. Kang. An Analysis of the Timed Z-channel. In *IEEE Symposium on Security and Privacy*, pages 2–11, 1996.

[Mil76] J. K. Millen. Security Kernel Validation in Practice. *Communications of the ACM*, 19(5):243–250, 1976.

[Mil87] J. K. Millen. Covert Channel Capacity. In *IEEE Symposium on Security and Privacy*, pages 60–66, 1987.

[Mil89] J. K. Millen. Finite-State Noiseless Covert Channels. In *Proceedings of the Computer Security Foundations Workshop*, pages 81–86, 1989.

[MM92] I. S. Moskowitz and A. R. Miller. The Channel Capacity of a Certain Noisy Timing Channel. *IEEE Transactions on Information Theory*, 38(4):1339–1344, 1992.

[MM94] I. S. Moskowitz and A. R. Miller. Simple Timing Channels. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 56–64, 1994.

[Mos90] I. S. Moskowitz. Quotient States and Probabilistic Channels. In *Proceedings of the Computer Security Foundations Workshop*, pages 74–83, 1990.

[Mos91] I. S. Moskowitz. Variable Noise Effects Upon a Simple Timing Channel. In *IEEE Symposium on Security and Privacy*, pages 362–372, 1991.

[MR96] J. C. Mogul and K. K. Ramakrishnan. Eliminating Receive Livelock in an Interrupt-driven Kernel. In *USENIX Annual Technical Conference*, pages 99–112, 1996.

[RD05] J. Regehr and U. Duongsaa. Preventing Interrupt Overload. In *Proceedings of the ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 50–58, 2005.

[SGLS77] M. Schaefer, B. Gold, R. Linde, and J. Scheid. Program Confinement in KVM/370. In *Proceedings of the ACM Annual Conference*, pages 404–410, 1977.

[Sha48] C. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[SM03] A. Sabelfeld and A. C. Myers. Language-based Information-Flow Security. *IEEE Journal on Selected Areas in Communication*, 21(1):5–19, 2003.

[SO01] J. H. Salim and R. Olsson. Beyond Softnet. In *Proceedings of the 5th Annual Linux Showcase and Conference*, pages 165–172. USENIX, 2001.

[TCS85] Department of Defense Trusted Computer System Evaluation Criteria (TCSEC), DOD 5200.28-STD, 1985.

[TG88] C.-R. Tsai and V. D. Gligor. A Bandwidth Computation Model for Covert Storage Channels and its Applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 108–121, 1988.

[The07] The MathWorks. xPC Target Realtime Kernel `http://www.mathworks.com/access/helpdesk/help/toolbox/xpc/ug/f5-15620.html`, April 12th 2007.

[Tro98] J. Trostle. Timing Attacks Against Trusted Path. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 125–135, 1998.

[VSI96] D. Volpano, G. Smith, and C. Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, 4(3):1–21, 1996.

[Wra91] J. C. Wray. An Analysis of Covert Timing Channels. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 2–7, 1991.