An Empirical Bandwidth Analysis of Interrupt-Related Covert Channels

Richard Gay, Heiko Mantel, and Henning Sudbrock

Department of Computer Science, TU Darmstadt, Germany
{gay,mantel,sudbrock}@mais.informatik.tu-darmstadt.de

Abstract We empirically evaluate interrupt-related covert channels, in short IRCCs, a type of covert channel that leverages hardware interrupts for communication. The evaluation is based on an exploit of IRCCs that we implemented as a proof-of-concept. We use a combination of experimental evaluation and information-theoretic analysis to compute the bandwidth of the channel on a concrete system. Our analysis shows a channel bandwidth of IRCCs based on interrupts of network interface cards (NICs) of approximately 5 bit/s. Besides the channel bandwidth, our experiments revealed previously unnoticed properties of IRCCs based on interrupts of NICs. While side channels based on hardware interrupts have been discussed before, this is the first experimental evaluation of covert channels based on hardware interrupts.

1 Introduction

A main goal of computer architectures is an efficient use of limited resources such as computation time and storage capacity. Hardware and operating systems therefore implement a sharing of hardware resources such as the CPU or main memory among running processes. Regarding the security of a system, this efficiency however comes at a cost: Shared resources can be used by malicious computer programs to establish covert communication channels [Lam73] that circumvent the system's security mechanisms.

When assessing a system with respect to the security it provides, covert channels must be taken into account. Completely eliminating covert channels is often impracticable when a certain level of efficiency shall be maintained. Instead, one may aim at finding an acceptable compromise in the trade-off between the efficiency of the system and the severity of covert channels by quantitatively assessing covert channels.

In this paper, we investigate *interrupt-related covert channels* [MS07, MS09], which exploit that operating system and userland processes typically share CPUs. These channels leverage that hardware devices communicate with the operating system via asynchronous interrupts, hence the channel's name. A sender of the channel can instruct the operating system to invoke an operation on a hardware device that, upon completion, triggers an interrupt to the operating system. A receiver of the channel, that executes on the same CPU as the operating system, can notice the interruption and thereby obtain information from the sender.

Interrupt-related covert channels (IRCCs) have been studied with respect to their theoretically achievable bandwidth and countermeasures against them [MS07, MS09, MSS+08]. IRCCs build on the interplay between hardware devices, operating systems, and userland processes. In recognition of this complexity, we find an empirical evaluation crucial for assessing the significance of IRCCs in real systems.

In this paper, we quantitatively assess IRCCs based on experimental evaluations. The main contributions of this paper are

- a proof-of-concept exploit of interrupt-related covert channels based on interrupts of network interface cards (NICs),
- an evaluation of the IRCC exploit combining experiments with a bandwidth analysis based on Shannon's information theory [Sha48], and
- two previously unnoticed properties of IRCCs based on NIC interrupts.

In particular, our evaluation combines empirical and analytic elements to compute channel bandwidths. Empirical results provide lower bounds for the achievable channel performance. In contrast, the analytic evaluation assumes the worst case for the attacker's capabilities of exploiting the channel in the sense that he may choose an ideal information encoding for transmissions. Overall, our results indicate that IRCCs can constitute a realistic threat to the confidentiality of secret information like passwords and secret keys.

2 Background

Covert channels. The consideration of covert channels in information systems can be traced back to Lampson [Lam73]. Since then, the identification, analysis, mitigation, and implementation has received great attention in computer security [Gli93, McH95, Kem83]. Among others, channels based on shared CPUs [Hus78], shared caches [KMO12], shared buses [Hu91], shared hard disks [KW91], have been identified and studied.

Interrupt-related covert channels. IRCCs are covert channels based on asynchronous hardware interrupts [MS07, MS09, Gay08]. These interrupts are raised by hardware devices such as network interface cards (NICs) or hard disks. Upon occurrence, they interrupt the CPU in executing userland or operating system processes. The interruption signals to the operating system that it can take care of the changed state of the respective hardware device.

Figure 1 illustrates how IRCCs exploit hardware interrupts for covert communication. When the sender wants to transmit the binary digit **1**, it requests the operating system to initiate an operation of a hardware device and yields the CPU immediately afterwards. While the operation is still ongoing, the operating system then switches the execution to the receiver. Once the hardware device finishes its task and triggers a hardware interrupt, the operating system stops executing the receiver, performs some device-specific operations and finally resumes the receiver. The induced delay of its execution can be detected by the receiver and its existence could be evaluated as a **1** being transmitted.

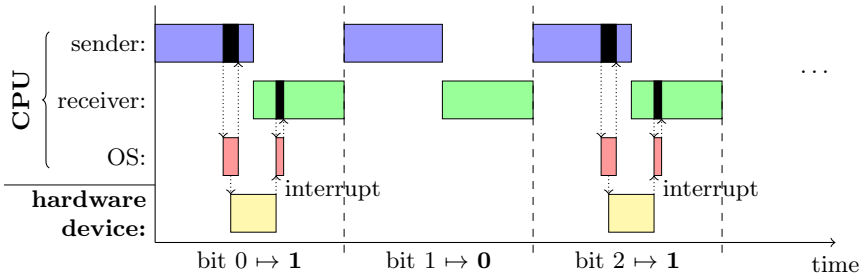

Figure 1: IRCC transmission schema

When the sender wants to transmit the digit **0**, it simply does not perform an operation that would cause an interrupt. The receiver can then determine the absence of an interruption.

Analytic evaluation of IRCCs. Mantel and Sudbrock [MS07, MS09] develop a formal information-theoretic framework for analyzing the bandwidth of IRCCs. They use the framework to derive upper bounds on the bandwidth of IRCCs. Moreover, using such upper bounds they evaluate the effectiveness of countermeasures that are intended to mitigate the capacity of IRCCs. They also investigate refinements of the information-theoretic model that take into account peculiarities of the environment in which an IRCC is exploited.

Experimental evaluation of IRCCs. We are aware of only two evaluations of interrupt-related channels in the literature. Molter et al. [MSS+08] empirically evaluate a hardware component as a countermeasure against IRCCs. The evaluation focuses on changes of measured interrupt durations due to the countermeasure. An evaluation of an IRCC channel is beyond the scope of the paper.

Trostle [Tro98] experimentally evaluates interrupt durations of a side channel based on keyboard interrupts. In this setting, the "sender" is a victim typing, e.g., her password and is, thus, not deliberately participating in the transmission over the channel. In contrast, the sender in our setting is actively participating in the transmission.

Countermeasures against IRCCs. The mitigation of IRCCs has been studied in the literature. Proposed countermeasures include changes in the time granularity available to processes, polling of hardware devices [MS07], and special-purpose hardware devices for interrupt handling [MSS+08]. In the current paper, we instead focus on the evaluation of IRCCs based on practical experiments as we consider this a gap in the literature's treatment of IRCCs.

3 The IRCC Exploit

We developed an exploit of IRCCs that consists of a sender program and a receiver program. The exploit transmits bit sequences from the sender to the

receiver under the assumption that both processes are executed on the same computer and that this computer as a single CPU. The transmission is unidirectional from the sender to the receiver. In absence of feedback from the receiver, the exploit transmits a bit sequence in consecutive time intervals of a fixed duration t_{bit} for each bit.

For establishing the channel, the exploit uses hardware interrupts of a wired network interface card (NIC). A NIC triggers a hardware interrupt when a packet has successfully been sent over the wire in order to inform the operating system that waiting queued packets can be sent.

Attacker model. We consider an attacker who is interested in confidential information stored on a computer system. The attacker is able to deploy the IRCC sender as a Trojan horse that can access the data. He is able to deploy the IRCC receiver on the same computer system. We assume that the deployed sender is unable to communicate the confidential information directly to the attacker or to the receiver by means of overt communication channels. The receiver does not have access to the confidential information but is able to send all information that it gets to the attacker. Finally, the attacker is able to encode and decode the input and output to the IRCC in an efficient way that allows to compensate transmission errors. That is, we assume that sender and receiver are already running. How to inject them into the target system is outside the scope of this paper. For instance, sender and receiver could be Trojan horses.

3.1 Fixed-Timeslice Scheduling

In this paper, we assume a setting, where side channels and other channels with substantial bandwidth have been addressed. In particular, we assume that *quantum-time channels* [Hus78] are ruled out by a scheduler that (a) assigns time slots, so called *timeslices*, of fixed length to processes and (b) does not permit a process to yield the CPU to another process prematurely. In the following, we refer to this concept as *fixed-timeslice scheduling*.

Remark 1. A *quantum-time channel* (QTC) is a covert channel that relies on a sender and a receiver process being executed on the same CPU. The sender of a QTC transfers information to a corresponding receiver by varying its time of CPU usage until it yields the CPU. The receiver detects this amount of time by measuring the time that it has not been running in favor of the sender. ◊

We implemented fixed-timeslice scheduling as a modification of the Linux scheduler. A vanilla Linux 2.6.22.9 kernel served as the basis for the modification. We make three modifications to the O(1) scheduler [BC05, Ch. 6] of this kernel. Firstly, we add code for distinguishing IRCC sender processes, IRCC receiver processes, and other processes by their effective group ID. Secondly, we enforce priority-independent timeslice lengths of $t_{\text{slice}} = 100\,\text{ms}$ for all IRCC processes. This includes blocking potential IRCC receiver processes from being scheduled as long as the preceding sender's timeslice has not elapsed. Thirdly, we disable

```
1 void send(data[]) {
2   wait_until(t0);
3   while (gettime() < t0 + length(data) * tbit) {
4     sched_yield();
5     wait_until(gettime() + tslice − δ);
6     idx = (gettime() − t0) / tbit;
7     if (data[idx] == 1)
8       generate_interrupt();
9 } }
```

Listing 1: Pseudocode of the main IRCC sender routine

dynamic process priorities, i.e., priorities varying due to a process's recent CPU consumption, for all IRCC processes.

Fixed-timeslice scheduling enables a stable number of sender-receiver rounds within multiples of $2\,t_{\text{slice}}$ through the second modification. Furthermore, the time made available to a sender or receiver process is independent from the process's previous time consumption due to the third modification. Both aspects simplify the later model and analysis of the channel in Section 4.2.

3.2 The IRCC Sender

The sender of the exploit uses an IRCC channel to transmit a given sequence of bits. Listing 1 shows a pseudocode version of the IRCC sender. The `data` parameter holds the bit sequence to be transmitted. The code consists of three main parts. The first part determines the start and end of the transmission interval for the bit sequence (lines 2–3). The second part synchronizes the execution of the sender with the end of every timeslice (lines 4–5). The third part generates an interrupt if, when executing the code, a **1** is to be transmitted (lines 6–8).

For establishing the IRCC, sender and receiver must execute at the same time. We synchronize their execution by first waiting for an agreed starting time t_0 of the transmission (line 2). The starting time could, e.g., be a full hour. The sender terminates its execution when all bits are transmitted. Line 3 checks this condition by querying the system clock.

The sender has to generate the interrupt at a point in time that makes the interrupt occur while the receiver is executing. With fixed-timeslice scheduling, we therefore generate interrupts always a certain amount of duration δ before the end of timeslices. For this, line 4 instructs the operating system to yield the CPU[1] such that when the execution of the sender is resumed in line 5, then this is within a fresh timeslice. Line 5 then waits for an amount δ less than the timeslice length t_{slice}. The following code is therefore executed an approximate duration δ before the end of the sender's timeslice.

Whether an interrupt has to be generated or not at some point in time depends on whether that time falls into the interval of a **0** bit or a **1** bit. This

[1] Note that due to fixed-timeslice scheduling, the CPU remains unused until the full timeslice has elapsed and the next process is only resumed afterwards. For our purposes here, it is only important that line 5 starts at the beginning of a timeslice.

```
1 void receive(counts[]) {
2   wait_until(t_0);
3   while (gettime() < t_0 + length(counts) * t_bit) {
4     dt = measure_duration(iters);
5     if (dt >= t_int^- && dt <= t_int^+) {
6       idx = (gettime() - σ - t_0) / t_bit;
7       counts[idx] += 1;
8 } } }
```

Listing 2: Pseudocode of the main IRCC receiver routine

is determined in lines 6–7. Finally, the generate_interrupt function generates the actual interrupt. This is done by the sendto system call with which we send a UDP packet with 1450 B of zeros.

3.3 The IRCC Receiver

The receiver of the exploit detects and records interruptions of its executions that are supposed to be caused by hardware interrupts generated by the IRCC sender. Listing 2 shows a pseudocode version of the IRCC receiver. The code returns an array holding the number of counted interrupts for every bit interval. The code consists of three main parts. The first part determines the start and end time of the transmission interval for the bit sequence (lines 2–3) and is the same as for the IRCC sender. The second part measures whether the execution time of a fixed portion of code was interrupted for a relevant duration (lines 4–5). The third part counts a relevant interrupt for the respective bit interval during which the interrupt occurred (lines 6–7).

The key element of the IRCC receiver is the second part. The actual measurings are performed by measure_duration, whose idea is to measure the execution time of a constant portion of code. The function computes and returns the difference dt between measured and uninterrupted duration (t_u). That is, if a hardware interrupt occurs during a measuring, then the measuring function returns the approximate duration of the interruption. If dt falls into the interval $[t_{\text{int}}^-, t_{\text{int}}^+]$, then it is considered an interrupt generated by the sender. This interval thus captures possible durations for interrupts generated by the sender.

A notable difference of the third part compared to the IRCC sender is the σ in line 6. This parameter introduces an offset between the bit intervals in the as seen by the sender and by the receiver. This offset captures that a certain amount of time elapses between the generation of an interrupt by the sender and the occurrence of the interrupt.

4 Experimental and Analytical Evaluation

The performance of our IRCC exploit depends on many aspects such as the configuration of the IRCC exploit itself, the compilation of the exploit into executable code, the scheduling of processes by the operating system, the behavior

experiment		*number of counted interrupts*					
		send ***0***		*send* ***1***			
#	t_u [μs]	0	1	0	1	2	3
1-1	0.183	8000	0	166	1929	5904	1
1-2		8000	0	161	1965	5874	0
2-1	1.823	8000	0	0	288	7712	0
2-2		8000	0	10	317	7673	0
3-1	181.2	8000	0	0	16	7984	0
3-2		8000	0	4	47	7949	0
4-1	724.8	8000	0	6	39	7955	0
4-2		7999	1	6	37	7957	0
5-1	1105	8000	0	1	161	7838	0
5-2		8000	0	9	553	7438	0

Table 1: Counted interrupts when transmitting $(\mathbf{10})^{8000}$ with the IRCC exploit

of other processes in the system, the behavior of the NIC in sending out network packets and in triggering hardware interrupts, and interrupts generated by other hardware devices such as timers or input devices. Given these complex dependencies, we aim for an experimental evaluation of the IRCC exploit to determine the performance of the exploit quantitatively.

In the following, we show the results of the experimental evaluation and analyze the channel's bandwidth using an information-theoretic model.

Experimental setup. We conducted all shown experiments on a desktop computer with a 1400 MHz CPU, 512 MB of main memory, and a 10 Mbit/s NIC. As operating system, we employed the modified Linux 2.6.22.9 kernel from Section 3.1 as part of a Gentoo Linux system. Running software besides the IRCC exploit was configured to default kernel threads, the udev device file system (version 119), and bash (version 3.2). For the compilation of the IRCC exploit from the source code, we used gcc 4.1.2 and binutils 2.18.

4.1 Experimental Determination of Channel Properties

We experimentally determine how many interrupts are detected by the IRCC receiver when the IRCC sender sends a **1** or, respectively, **0**. More precisely, we use the exploit to transmit the sequence $(\mathbf{10})^k$ for $k = 8000$ samples. The time interval for each bit is $t_{\text{bit}} = 4 \cdot t_{\text{slice}} = 400\,\text{ms}$. That is, both sender and receiver can execute for at most two timeslices per bit and the sender can generate at most $N = 2$ interrupts for every bit. Concerning the remaining parameters of the exploit, we experimentally determined the following values to yield good results: $\delta = 40\,\mu\text{s} + t_g$ where t_g is the runtime of `generate_interrupt`, $t_{\text{int}}^{-} \approx 15.56\,\mu\text{s}$, $t_{\text{int}}^{+} \approx 16.34\,\mu\text{s}$, and $\sigma = 2\,\text{ms}$. Based on these parameters, we compare the channel properties for different settings of the receiver's `iters` parameter, which yields different uninterrupted durations t_m of the receiver's measuring function `measure_duration` (Listing 2). For each setting, we conducted two transmissions.

If the channel was flawless, then sending a **0** would cause 0 interrupts to be counted for all 8000 **0**s. Sending a **1** would cause $N = 2$ interrupts to be counted for all 8000 **1**s. Table 1 shows the actual results of our experiments.

The experiments show that in overall only two cases (experiments 1-1 and 4-2), more interrupts were counted than were actually generated. In contrast, all of the experiments show that some generated interrupts were not counted by the receiver. The concrete number of missed interrupts turned out to depend on the measuring duration. Note that experiment 3-1 shows a particularly low interrupt miss ratio of 0.2%.

By using an appropriate channel encoding, an attacker could run the exploit and compensate transmission errors caused by missed interrupts. In the following, we therefore analyze the maximum bandwidth that an attacker could achieve for the empirically determined channel of Table 1.

4.2 Analysis of IRCC Channel Bandwidth

For the analysis of the bandwidth, we employ a model of a discrete memoryless channel [CT06], which we introduce first. Afterwards, we adapt the model for our concrete setting based on the experimentally determined transition matrices.

A communication channel can be represented by a triple of possible inputs, possible outputs, and a model of the relationship between inputs and outputs. A *discrete* channel is a communication channel with discrete input alphabet I and discrete output alphabet O. The channel is *memoryless* if the relationship between inputs and outputs can be expressed by a *transition matrix* $P = (p(y|x))_{x \in I,\, y \in O}$, where $p(y|x)$ is the conditional probability of obtaining output $y \in O$ given input $x \in I$. That is, an output depends only on the respective current input but not on previous or future inputs or outputs of the channel.

A model for the exploit's channel. We model the channel established by the IRCC exploit as a discrete memoryless channel where the input alphabet is $I = \{\mathbf{0}, \mathbf{1}\}$ and the output alphabet is $O = \{0, 1, \ldots, N\}$. Inputs capture single bits while outputs capture the number of counted interrupts, which is bounded by the maximum number N of interrupts generated by the sender when sending a **1**. We model the transition matrix P such that $p(Y{=}0|X{=}\mathbf{0}) = 1$ and $p(y|X{=}\mathbf{1}) = \binom{N}{y}(1-\lambda)^y \lambda^{N-y}$ for $y \in O$. That is, the number of counted interrupts follows a binomial distribution parametric in λ, which models the probability of a single interrupt being missed by the receiver. The binomial distribution models stochastic independence of pairs of missed interrupts.

Lemma 1. *Let $\lambda \in [0, 1)$ and $N \in \mathbb{N}$. Then the capacity of the discrete memoryless channel (I, O, P) is*

$$Cap(I, O, P) = -(1 - \pi\bar{\ell}) \log(1 - \pi\bar{\ell}) + \pi\ell \log \ell + \pi\bar{\ell} \log \pi$$

where $\ell = p(Y{=}0|X{=}\mathbf{1}) = \lambda^N$, $\bar{\ell} = 1 - \ell$, and $\pi = (\bar{\ell} + \ell^{-\ell/\bar{\ell}})^{-1}$. ◊

A proof of the lemma can be found in the appendix.

			capacity [bit]		*bandwidth [bit/s]*	
experiment	λ	G	N=1	N=2	N=1	N=2
1-1	14.13%	0.05	0.698	0.930	3.491	2.324
1-2	14.29%	0.05	0.696	0.928	3.478	2.321
2-1	1.80%	5.28	0.935	0.998	4.676	2.495
2-2	2.11%	8.33	0.927	0.997	4.633	2.493
3-1	0.10%	0.02	0.994	1.000	4.972	2.500
3-2	0.34%	23.32	0.983	1.000	4.917	2.500
4-1	0.32%	42.79	0.985	1.000	4.923	2.500
4-2	0.31%	43.87	0.985	1.000	4.925	2.500
5-1	1.02%	0.03	0.959	0.999	4.796	2.948
5-2	3.57%	0.15	0.889	0.993	4.447	2.482

Table 2: IRCC channel properties under discrete memoryless channel model

The model in the experiments. Using the maximum-likelihood estimator of the binomial distribution [Hoe66, 3.3], we instantiate the model by computing $\lambda = \bar{y}/N$, where $N = 2$ and where $\bar{y}$ is the average number of interrupts counted for input **1** in Table 1. Based on Lemma 1, we compute the capacity of the channel. The bandwidth of the channel can then be computed as $B = Cap\,/(N \cdot t_{\text{slice}})$.

Table 2 shows the results of the analysis for the different experiments. The estimated probability λ of missing an interrupt in the receiver ranges from about 0.1% to about 14.29%. The corresponding channel bandwidths peak at approximately 2.5 bit/s for N=2, which we used in the experiments. When we transfer the values of λ to N=1, then computed bandwidths peak at 4.972 bit/s.

Adequacy of the model. The model makes two assumptions: first, at most as many interrupts can be counted as are generated by the sender. Second, misses of interrupts are stochastically independent. We see the first assumption as valid, given that the experiments only showed very rare cases in which one additional interrupt was counted.

The second assumption simplifies the transition matrix to a binomial distribution of counted interrupts when **1** is sent. Goodness of fit tests [Hoe66, Wik13] support this assumption for half of the experiments (where $G < 3.84$).

4.3 Findings

Besides the probabilities of interrupts being missed, our experiments with the IRCC exploit revealed further properties of IRCCs that use NIC interrupts.

Interrupt time frames. Depending on properties of the NIC, the operating system's scheduling, and the precision of the IRCC sender, we speculated that interrupts might occur only within a limited time frame of the receiver's timeslices. We conducted experiments with a modified IRCC receiver, in which we measured at what times interrupts occurred relatively to the beginning of the receiver's timeslices. Our experiments show that the interrupts generated by the

sender occurred only during a relatively short time frame of about 33 μs within the succeeding receiver timeslice. The standard deviation of occurrence times from the mean time turned out to be only 0.92 μs.

Based on the knowledge about the narrow time frame, we assume that an augmented exploit could reach significantly higher bandwidths by encoding information in the time at which the interrupt is generated by the sender (i.e., by varying the δ parameter in Listing 1).

Multiple interrupts. Depending on the interrupt source chosen for the channel it may be possible to generate multiple interrupts at once. With our IRCC exploit based on NIC interrupts, we discovered that sending a payload that would exceed the maximum transmission unit leads to multiple packets being sent and one interrupt being caused for each packet. Experiments involving the generation of four NIC interrupts showed very low standard deviations for the delay until the interrupt occurrences (less than 5 μs) and for the respective interruption durations (less than 0.3 μs). Delays and durations of all four interrupts furthermore assumed values from mutually disjoint ranges.

We assume that an augmented exploit could leverage the possibility of sending several interrupts to transmit more information within a single pair of sender and receiver timeslices.

5 Conclusion

Previous works on IRCCs provide worst-case results for the bandwidth of IRCCs based on formal models of the channel [MS07, MS09]. To substantiate the relevance of IRCCs as a non-negligible threat for information systems, we show based on an implemented exploit that IRCCs are feasible in practice. Our experimental and analytical evaluation shows bandwidths of approximately 5 bit/s, which suffice to disclose data such as passwords or private keys within an acceptable amount of time.

In this paper, we use NIC interrupts for establishing an IRCC. Typical systems contain other devices such as hard disks that use hardware interrupts. IRCC exploits could use these devices instead or in addition. As such, our results constitute a lower bound on the bandwidth of IRCCs. In our setup, the few running processes do not create a large amount of noise. The presence of noise caused by other processes would reduce the bandwidth but we expect it to not completely eliminate the channel. Such noise can be countered by tailored encodings from information theory [CT06]. For our setup, we used fixed-timeslice scheduling. In a setting without fixed-timeslice scheduling, higher bandwidths of IRCCs are achievable due to shorter timeslices. In addition, quantum-time channels could be used for transmitting information.

Our findings in Section 4.3 reveal two properties of IRCCs based on NIC interrupts: IRCCs can exploit the point in time at which an interrupt occurs as well as how an interrupt relates to preceding or following interrupts. This allows an IRCC to encode more information in a single interrupt and thereby increase

the bandwidth. To our knowledge, these properties have not previously been noticed for IRCCs.

Acknowledgments. This work was funded by the DFG under the project FM-SecEng in the Computer Science Action Program (MA 3326/1-3).

References

[BC05] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel.* O'Reilly & Associates, third edition, 2005.

[CT06] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley-Interscience, second edition, 2006.

[Gay08] R. Gay. Interrupt-Related Covert Channels from an Attacker's Perspective. Diploma thesis, RWTH Aachen, December 2008.

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

[Hoe66] P. G. Hoel. *Introduction to Mathematical Statistics.* John Wiley & Sons, Inc, third edition, 1966.

[Hu91] W.-M. Hu. Reducing Timing Channels with Fuzzy Time. In *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, 1978.

[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.

[KMO12] B. Köpf, L. Mauborgne, and M. Ochoa. Automatic Quantification of Cache Side-Channels. In *24th International Conference on Computer Aided Verification*, LNCS 7358, pages 564–580. Springer, 2012.

[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.

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

[McH95] J. McHugh. Chapter 8: Covert channel analysis from handbook for the computer security certification of trusted systems. Technical Memorandum 5540:080A, Naval Research Laboratory, 1995.

[MS07] H. Mantel and H. Sudbrock. Comparing Countermeasures against Interrupt-Related Covert Channels in an Information-Theoretic Framework. In *20th IEEE Computer Security Foundations Symposium*, pages 326–340, 2007.

[MS09] H. Mantel and H. Sudbrock. Information-Theoretic Modeling and Analysis of Interrupt-Related Covert Channels. In *Workshop on Formal Aspects in Security and Trust*, Springer, LNCS 5491, pages 67–81, 2009.

[MSS+08] H. G. Molter, H. Shao, H. Sudbrock, S. A. Huss, and H. Mantel. Designing a Coprocessor for Interrupt Handling on an FPGA. Technical report, TUD-CS-2008-1103 (TU Darmstadt), 2008.

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

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

[Wik13] Wikipedia. G-test — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/G-test`, 2013. [Online; accessed 21-June-2013].

A Proofs

The following proof is an adapted version of so far unpublished work by one of the authors [Gay08].

Proof (Lemma 1). Let $p(X)$ be a probability distribution over $x \in I$. Let $\pi := p(X{=}\mathbf{1})$. Then $p(X{=}\mathbf{0}) = 1 - \pi$. The definition of mutual information gives

$$\begin{aligned} I(X;Y) &= \sum_{x,y} p(x,y) \cdot \log \frac{p(x,y)}{p(x)p(y)} = \sum_{x,y} p(y|x)p(x) \cdot \log \frac{p(y|x)}{p(y)} \\ &= (1-\pi) \sum_{y} p(y|X{=}\mathbf{0}) \log \frac{p(y|X{=}\mathbf{0})}{p(y)} + \pi \sum_{y} p(y|X{=}\mathbf{1}) \log \frac{p(y|X{=}\mathbf{1})}{p(y)} \\ &\overset{p(Y=0|X=\mathbf{0})=1}{=} -(1-\pi) \log p(Y{=}0) + \pi \sum_{y} p(y|X{=}\mathbf{1}) \log \frac{p(y|X{=}\mathbf{1})}{p(y)}. \end{aligned}$$

The second sum can now be split into the cases $y = 0$ and $0 < y \leq N$. Let $\ell := p(Y{=}0|X{=}\mathbf{1})$ and $\bar{\ell} := 1 - \ell$. Then

$$\begin{aligned} I(X;Y) &= -(1-\pi) \log p(Y{=}0) + \pi\ell \log \frac{\ell}{p(Y{=}0)} + \pi \sum_{0<y\leq N} p(y|X{=}\mathbf{1}) \log \frac{p(y|X{=}\mathbf{1})}{p(y)} \\ &= -(1-\pi+\pi\ell) \log p(Y{=}0) + \pi\ell \log \ell + \pi \sum_{0<y\leq N} p(y|X{=}\mathbf{1}) \log \frac{p(y|X{=}\mathbf{1})}{p(y)}. \end{aligned}$$

From $p(y) = \sum_x p(y|x)p(x)$, according to the law of total probability, we obtain $p(Y{=}0) = 1 - \pi(1-\ell) = 1 - \pi\bar{\ell}$ and, for $y > 0$, $p(y) = \pi p(y|X{=}\mathbf{1})$. This gives

$$\begin{aligned} I(X;Y) &= -(1-\pi\bar{\ell}) \log(1-\pi\bar{\ell}) + \pi\ell \log \ell - \pi \log \pi \sum_{0<y\leq N} p(y|X{=}\mathbf{1}) \\ &= -(1-\pi\bar{\ell}) \log(1-\pi\bar{\ell}) + \pi\ell \log \ell - \bar{\ell}\pi \log \pi. \end{aligned} \tag{1}$$

The capacity $Cap(I,O,P)$ is defined as the maximum of the mutual information $I(X;Y)$ between input and output, with respect to the input distribution. This input distribution is represented by the single parameter $\pi = p(X{=}\mathbf{1})$. The maximum[2] can be computed by solving $\frac{d}{d\pi} I(X;Y) = 0$ for π. For this, we first differentiate $f(\pi) \cdot \log f(\pi)$ wrt. π for any differentiable function f:

$$\begin{aligned} \frac{d}{d\pi} f(\pi) \log_2 f(\pi) &= f'(\pi) \log_2 f(\pi) + f(\pi) \cdot \frac{f'(\pi)}{f(\pi)\ln(2)} \\ &= f'(\pi) \cdot (\log_2 f(\pi) + \log_2(e)) = f'(\pi) \cdot \log_2(ef(\pi)). \end{aligned}$$

By utilizing the above result, we obtain

$$\begin{aligned} \frac{d}{d\pi} I(X;Y) &= \bar{\ell} \log(e(1-\pi\bar{\ell})) + \ell \log \ell - \bar{\ell} \log(e\pi) \\ &= \ell \log \ell + \bar{\ell} \log\left(1/\pi - \bar{\ell}\right) \overset{!}{=} 0. \end{aligned}$$

From $\lambda \in [0,1)$, we know $\ell < 1$. This allows to solve the previous equation for π, resulting in $\pi = \left(\bar{\ell} + \ell^{-\ell/\bar{\ell}}\right)^{-1}$. For this value of π, we get the capacity by computing the mutual information as in (1). Hence the lemma holds. □

[2] The extremal value is a maximum since $I(X;Y)$ is concave [CT06, Theorem 2.7.4].