Increased reproducibility and comparability of data leak evaluations using ExOT

Increased reproducibility and comparability of data leak evaluations using ExOT

Philipp Miedl[†] miedlp@ethz.ch
Bruno Klopott[†] klopottb@student.ethz.ch
Lothar Thiele[†] thiele@ethz.ch

Abstract

As computing systems are increasingly shared among different users or application domains, researchers have intensified their efforts to detect possible data leaks. In particular, many investigations highlight the vulnerability of systems w. r. t. covert and side channel attacks. However, the effort required to reproduce and compare different results has proven to be high. Therefore, we present a novel methodology for covert channel evaluation. In addition, we introduce the Experiment Orchestration Toolkit ExOT*, which provides software tools to efficiently execute the methodology.*

*Our methodology ensures that the covert channel analysis yields expressive results that can be reproduced and allow the comparison of the threat potential of different data leaks. ExOT is a software bundle that consists of easy to extend C++ libraries and Python packages. These libraries and packages provide tools for the generation and execution of experiments, as well as the analysis of the experimental data. Therefore, ExOT decreases the engineering effort needed to execute our novel methodology. We verify these claims with an extensive evaluation of four different covert channels on an Intel Haswell and an ARMv***8** *based platform. In our evaluation, we derive capacity bounds and show achievable throughputs to compare the threat potential of these different covert channels.*

1. Introduction

As computing devices are getting more powerful, they are often intended for shared use to fully utilise the available resources. For example, multiple users have access to the same cloud computing infrastructure, or mobile devices are used for multiple application domains with different security clearances such as business and private applications. Cloud and mobile computing systems must prevent data leak from one user, or one application domain, to another. Such a data leak may exist in the form of a covert or side channel, which are closely related. This relation is highlighted by Ristenpart et al. [15], stating that "Covert channels provide evidence that exploitable side channels may exist.". In other words, covert channels help to estimate the extend to which side channel attacks might be feasible. This is important as quantifiable metrics often cannot be determined for side channels, but for covert channels it is relatively easy to derive well known metrics.

[†]ETH Zurich, Computer Engineering and Networks Laboratory (TIK), Gloriastrasse 35, Zurich, Switzerland

Figure 1: A data leak where the source emits information via the channel, received by the sink and forwarded to an adversary. The data transfer is either hidden (covert channel), or unintentional (side channel).

We base our analysis of covert channels on the setup illustrated in Figure 1, which is widely used in literature. The main components are (i) the *source* application with access to confidential information, (ii) the *channel*, and (iii) the *sink* application which receives and forwards data to an adversary.

While the system in Figure 1 looks rather simple, an exhaustive analysis can be very costly in terms of time and engineering effort. Hence, an exhaustive analysis is not common practice and examples for limited experimental evidence can be found in recent literature, for example the thermal and cache covert channels. The thermal covert channel was first presented by Masti et al. [8] and seemed to be quite harmless, only allowing data transmission at 1.33 bps with 11% bit error. Yet, Bartolini et al. [3] showed in their analysis that the upper channel capacity bound is beyond 100 bps and outperformed the experimental throughput results of Masti et al. [8] by more than 20 times. This illustrates, how the lack of an exhaustive analysis yields results that are not expressive and may lead to false conclusions. In different work, Gruss et al. [6] stated that the comparison of previously presented throughputs of different cache covert channels was not possible. The authors pointed out, that due to the different implementation strategies all experiments had to be repeated. This might require high engineering effort and highlights the importance of comparable metrics for data leaks, rather than implementations. These examples indicate that there is a need for a well defined *methodology* for the analysis of covert channels.

Such a methodology would allow a confident estimation of the threat potential of a covert channel if it is (i) general applicable to a large class of covert channels, (ii) defines models, metrics and experiments, as well as (iii) ensures that the results are *reproducible*, *comparable* and *expressive*. Based on the international vocabulary of metrology [2] we define reproducibility, comparability and expressiveness as follows.

Reproducibility. An analysis is reproducible if different researchers can repeat it based on the description of the original work, and derive the same conclusions.

Comparability. Comparability of different data leak analyses is assured, if their results are reported using the same metrics.

For example, the threat potential of data leaks cannot be compared if one is quantified using throughputs and another using capacity bounds.

Expressiveness. Expressiveness of metrics describe qualitatively how well they characterise relevant properties of the evaluated system. For example, an implementations might not reveal the full potential of a data leak. Therefore, throughput and the corresponding error rate are only expressive towards an implementation, rather than a data leak. In contrast, the channel capacity bound reports the maximum possible throughput under ideal conditions. Hence, the capacity bound is expressive towards the threat potential of a data leak.

Contributions. To the best of our knowledge, we are the first to propose a widely applicable methodology for covert channel analysis. The methodology meets the requirements for repeatability, comparability and expressiveness. Furthermore, it defines models, metrics and experiments. In addition, we present the supporting *Experiment Orchestration Toolkit ExOT*. It reduces the engineering effort needed to execute the proposed methodology. ExOT is designed to be easily extended or reconfigured, such that a variety of different classes of experiments can be executed. Thus, ExOT substantially exceeds the functionality of previously presented work like the Mastik toolkit [16] or libflush [5, 7]. For example, ExOT allows to validate new as well as previously presented data leak analyses or to quantify the effects of mitigation techniques.

2. Revisiting known covert channels

In this section, we review a selection of covert channels that have been presented in recent years. While the selection is not exhaustive, its purpose is to show the usability of the proposed methodology and the toolkit. We choose two different covert channels families, thermal and cache covert channels.

2.1. Thermal

To establish a thermal covert channel, the source application encodes information into temperature changes. By generating a high utilisation, the cores of the platform heat up. If the source application is idle the platform cores cool down. Bartolini et al. [3] presented upper capacity bounds of 334 bps for an Intel Haswell and 414 bps an ARMv7 architecture, respectively. The determined throughputs are listed in Table 1.

2.2. Cache

The cache covert channels considered in this work rely on timing measurements during operations on data. If data resides in different levels of the memory hierarchy, the time needed for the data access will vary. As multiple cores share the last level cache, the timing of applications on other cores can be influenced by forcing specific cache blocks to be removed from cache. Based on these forced timing variations, a cross-core covert channel can be established.

In this work, we consider three different cache timing covert channels: (i) the *Flush+Flush* [6, 7] channel relying on the execution time of the flush operation, (ii) the *Flush+Reload* [6, 7, 17] channel using the time needed to execute the reload command, and (iii) the *Flush+Prefetch* [5] channel exploiting the timing of the prefetch instruction. We will not provide details of the underlying primitives due to space constraints.

Table 1: Throughputs of known covert channels, which are hard to compare due to varying evaluation methods and error rates.

Covert Channel	Empirical Throughput / Error Rate	
	x86_64	ARM
Thermal	56 bps / 1.74%	49 bps / 1.70%
Flush+Flush	496 KB/s / 0.84%	178.3 Kbps / 0.48%
Flush+Reload	298 KB/s / 0.00%	1.14 Mbps / 1.10%
Flush+Prefetch	146 KB/s / $< 1\%$	N/A

None of the mentioned references present capacity bounds for their covert channels, while all provided throughput experiments. In Table 1, we report the results obtained from the re-implementation of the "Flush+" channels by Gruss et al. [6]. The throughputs were all determined using different implementations and report different error rate metrics. Hence, they cannot be used as a measure of the general threat potential of this class of data leaks. The throughputs are only an expressive metric for a specific implementation. We address the issue of expressiveness and show that the new methodology allows the comparison of threat potentials of fundamentally different covert channels.

3. A covert channel analysis methodology

Figure 2 illustrates the four main steps of our proposed covert channel analysis methodology. In this section, we describe these four steps and how our toolkit implements them.

3.1. Modelling

During the modelling phase, the covert channel needs to be formally described. A model is a necessary prerequisite for comparability and is beneficial when defining the metrics for evaluation. For example in case of covert channels, the formal models enables us to derive an upper capacity bound.

This phase cannot be automated as it relies on the expertise of the researcher. However, the framework provides two ready-to-use models, which can be applied and determine the experimentation flow. The formal models that are implemented and described in ExOT are (i) time-continuous and value-continuous (e. g. thermal [3]), as well as (ii) time-discrete and value-discrete (e. g. power [10]). One of these two models applies to most covert and side channels.

3.2. Capacity Bound Derivation

The capacity bound is a metric that allows to estimate the threat potential of a covert channel. A capacity bound is

independent from implementation artifacts, as it describes the maximum capacity of a channel achievable under ideal conditions. Hence, capacity bounds are expressive regarding the general threat potential of a data leak, rather than a specific implementation. Examples for capacity bound derivations can be found in related work [3, 4, 10, 12].

Depending on the previously established model, ExOT provides an experimentation scaffold to determine the parameters necessary to derive the capacity bounds. This includes experiment definition and configuration, as well as the experiment generation, execution and analysis flow. In addition, ExOT controls environmental factors, if the hardware setup allows it. Furthermore, ExOT provides basic building blocks for the applications generating the channel input and recording the channel output. These building blocks can be used on many different platforms, as ExOT also includes a cross-compilation suite and allows for the integration with other tool chains.

3.3. Experimental Channel Evaluation

Based on the experimental evidence provided by the evaluation, the previously established channel model and the capacity bounds are validated. Furthermore, an expressive metric for the threat potential of a specific implementation is provided in the form of throughputs.

The experiments are conducted under well defined laboratory conditions with a prescribed software flow to support reproducibility. Their purpose is to understand the capabilities of the channel and the effect of external influences. The experiment setup ensures reproducibility as the source and the sink application are well synchronized and external influences are controlled. In this phase, first a simple stream of random bits is transmitted from one application to another. Simple source coding (conversion of bits to symbols) and line coding (conversion of symbols to a channel input trace) techniques are used to establish a rudimentary communication channel with minimal engineering effort. Using this configuration, possible throughputs with corresponding error rates are determined and additional experiments are performed to better understand the covert channel. For example, the effect of external influences or additional noise, e. g., generated by other applications can be quantified. This approach yields useful insight for the development of mitigation techniques.

ExOT provides the prescribed software flow and handles source and sink application synchronisation. Furthermore, ExOT offers a variety of different source and line coding options that can be configured and applied to a randomly generated bit stream. Moreover, ExOT also offers an interface to control external influences and additional applications to investigate external influences.

3.4. Deployment Test

Using the knowledge gained in the previous steps, it is possible to evaluate the covert channel in a real world scenario. An example for a deployment test was presented by Maurice et al. [9]. In contrast to the experimental channel evaluation phase, in the deployment test phase the source and sink applications have to operate fully autonomously. This requires that the source and sink application also implement measures that compensate for external influences, like noise or interference. Furthermore, the application might also need to implement sophisticated protocols with synchronisation methods, packeting, bi-directional communication, error detection or error correction. Therefore, such real world implementation can be designed in many different ways and the throughput will often heavily depend on the invested engineering effort. The main goal of the deployment test is to show that an attack can be deployed outside of a laboratory setup. In addition to the previously determined metrics, the deployment tests can yield additional measures like the attack footprint or the necessary implementation effort. ExOT provides application building blocks and experimentation execution flows to reduce the engineering effort in this phase of the analysis.

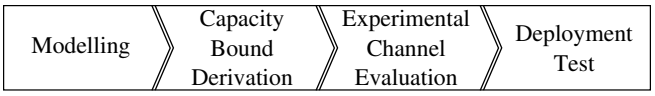

Figure 2: Main steps of the proposed methodology for covert channel analysis.

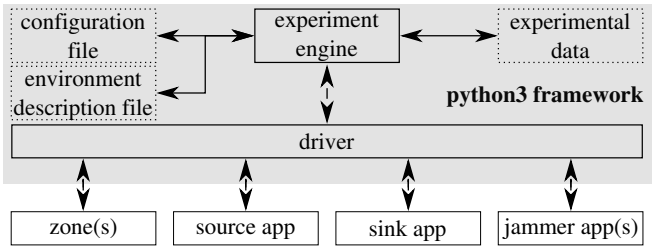

Figure 3: Experimental setup using ExOT. The driver is the interface to the experiment environment consisting of platform zone(s) and the applications.

Our methodology describes which metrics need to be determined and which experiments have to be executed, ensuring reproducible, comparable and expressive results. As we will show in Section 5, it is generally applicable to different kinds of covert channels. Therefore, our proposed methodology fulfils all the requirements defined in Section 1.

4. The Experiment Orchestration Toolkit (ExOT)

In this section, we present implementation details of our *Experiment Orchestration Toolkit ExOT*. Figure 3 shows the structure of ExOT and the interaction of the different components of the setup. Using a *TOML*[1] configuration file and the environment descriptor to capture relevant experiment parameters, ExOT supports repeatability of the experiments.

The experiment environment consists of at least one plat-

[1] https://github.com/toml-lang/toml

Layer Name	*Layer Functions*	
6 - Generate/Verify	generate bits	calculate metrics
5 - Source Coding	bits to symbols	symbols to bits
4 - Line Coding	symbol to trace	trace to symbols
3 - Raw Data Processing	output formatting	raw data to trace
2 - I/O Module	write schedule files	read meas. files
1 - Applications	utilise channel	observe channel
0 - Channel	covert information transmission	

Figure 4: Information flow model. Data travels from the highest to the lowest layer, gets transferred via the channel, and travels up to the highest layer.

form zone, a source and sink application pair and optionally jammer applications. Jammers can be used to simulate disruptive influences on the covert channel, which is useful to either (i) understand the influence of external factors on the covert channel, or to (ii) evaluate possible mitigation strategies. All applications are mapped to a zone, whereas one zone defines a certain platform configuration. For example, a platform may have a secure and an insecure zone.

The experiment engine will setup the environment by configuring the zones(s) and applications as well as copying the necessary data. It controls the experiment execution, fetches the data and cleans up the environment after the experiment execution has finished. The experiment engine also offers a variety of debug outputs in the form of log-messages during execution, or plots for data preview during analysis. The complete flow from generating an experiment to analysis can be written in one Python script, which makes experiments easier to version and maintain.

4.1. Creating sending and receiving applications

We implemented the application library using C++17, taking advantage of modern language features. This includes the use of generic programming, compile-time code generation, templated design and inheritance without complex class hierarchies. These development paradigms ensure that the code base is extendable without too much code duplication. The library provides basic building blocks for the application, whose design is based on the concept of process networks.

The library also contains utilities for (i) a simple and reliable JSON interface for application configuration, (ii) logging and debug output, (iii) file system and Model Specific Register (MSR) handling (iv) execution, exception and signal handlers, as well as (v) time keeping and clocking. In addition to the application library of ExOT, we provide a compilation suite. This compilation suite is based on docker and CMake, allowing easy cross-compilation and integration with other tool-chains, for example the Android NDK. This enables researchers to easily port an analysis to different architectures.

4.2. Information flow

We base our data processing design on a layered information flow model, illustrated in Figure 4. Similar to the well known OSI model, information travels from the highest layer to the lowest, and then up to the highest again. Layers 2 to 6 are implemented as Python packages, which has the following advantages: (i) there is no need for recompilation when a new data processing scheme is tested, (ii) the implementation is platform independent, and (iii) data checks, using for example plots, and debugging are easy to perform.

4.3. Extendability and limitations of ExOT

Due to its design, ExOT can be applied to a wide variety of fields and is not limited to analyses presented in this paper. Using ExOT for additional channels, attacks or other analyses requires an extension of the experiment definitions in the Python framework. If specific deployment applications are necessary, these can be implemented using the building blocks of the C++17 library, or by extending the library.

The applicability of ExOT mostly depends on the platforms the source, sink and jammer applications are run on. This dependency arises as the application building blocks provided by the C++ library often depend on architectural features of the platform. For example, the timing accuracy and maximum sampling period of the applications depends on the timing source provided by a platform.

At the time of initial publication, ExOT supports various platforms that are based on a Linux or Android Operating System (OS) and requires SSH or ADB capabilities of the platforms. However, an extension of ExOT to other OSs or communication interfaces is possible. While including a new communication interface only requires to add a fitting driver to the Python framework, expanding ExOT to new OSs would also call for an extension of the C++17 library.

5. Re-evaluating known covert channels

In this section we show the experimental evaluation of the covert channels presented in Section 2. We use the proposed methodology and ExOT to derive comparable and expressive metrics in a reproducible fashion. We determine a comparable metric, namely the capacity bound, for all of the considered covert channels. To provide experimental evidence for the validity of the models and bounds, we conduct experimental throughput evaluations. These allow a direct comparison of different implementations and platforms. In addition, we analyse the robustness of the implementations towards interference. We perform the evaluation on (i) a Lenovo T440p laptop based on a Intel i7-4700MQ, referred to as *Haswell*, and (ii) a NVIDIA Jetson TX2 based on a dual-core Denver 2 64-bit CPU and quad-core ARM A57 cluster, referred to as *ARMv8*. All information gathered during the experiments is published to ensure reproducibility [13].

Table 2: Time for one covert channel use depending on the cache state.

Covert Channel	Haswell		ARMv8	
	cached	flushed	cached	flushed
Flush+Flush	269.2 ns	265.6 ns	3569.2 ns	3557.8 ns
Flush+Reload	219.9 ns	304.7 ns	3634.4 ns	4078.9 ns
Flush+Prefetch	224.5 ns	303.3 ns	3605.6 ns	4080.1 ns

5.1. Modelling and Capacity Bound Derivation

Thermal. For the thermal covert channel we use the established model from Bartolini et al. [3]. After determining the channel frequency spectrum, we apply the constrained-input water-filling to determine the capacity. In contrast to the original work, where either a single or the sum of all temperature readings was used for data transmission, we will use the maximum of all core temperatures. This conforms to the way readings are processed on the ARMv8 platform [1]. Furthermore, while the original work used channel sub-band splitting to apply the constrained-input water-filling, we use a whitening filter in our calculations. Details on the experiments and the calculations done to determine the capacity bound can be found in Bartolini et al. [3] and the technical report appended to this work [11].

Cache. Typically, there are multiple memory levels in a computing system, i. e., up to three cache levels, the main memory and swap. However, as already indicated in the original work [5–7, 17], our initial experiments have also shown that it is only feasible to reliably control and determine whether data is cached or not. Considering this observation and due to the fact that no models of cache covert channels were given in the original work, we establish a channel model based on a state machine with two states and all possible transitions. We use the method presented by Miedl and Thiele [10] to determine the maximum capacity of 1 bit per channel use. To get a comparable metric, we need to determine the duration of one channel use. We define one channel use as (i) measuring the timing with the defined method, and (ii) resetting the memory state. The channel access times are measured by repeatedly performing the Flush+Flush, Flush+Reload or Flush+Prefetch access on either a cached or not-cached set. The measurement results shown in Table 2 are the average time acquired from a quarter million measurements per channel and cache state. Using the best case timing for each cache channel, we derive the upper channel capacity bounds.

Comparison. The formal models of the considered covert channels enable us to derive capacity bounds, outlined in Figure 5. As expected, the capacity bounds suggest that the threat potential of cache based data leaks is much higher than that of thermal measurement based data leaks. However, the capacities vary by almost 10× when comparing our two platforms Intel and ARMv8. Furthermore, the throughputs reported by Gruss et al. [6] for their ARM based platform are higher than

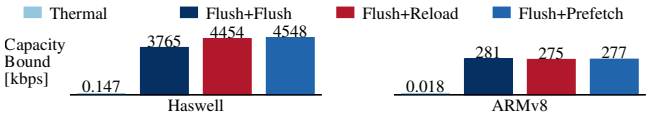

Figure 5: The capacity bounds indicate that cache based data leaks have a higher threat potential than ones that are based on thermal information.

the capacity bounds we derived for the ARMv8 platform. This indicates that the threat potential of cache covert channels is not only architecture, but also platform dependent and therefore highlights the importance of data leak evaluations on different platforms.

5.2. Experimental Channel Evaluation

As a final step, we conduct experiments to validate the model and the capacity bounds. Furthermore, we will assess how jamming applications influence the performance of the covert channels.

To quantify the performance of the covert channels we divide the analysis into two phases, training and evaluation. During the training phases, we transmit a random bitstream with 1.5 kbit, which we use to train the decoder decision device. In the evaluation phase, a 5 kbit bitstream is transmitted and evaluated using the trained decoder decision device. We repeat each phase 5 times and use the mean of the repetitions for further calculations to compensate for variations introduced by the hardware setup. Both phases are then executed for multiple bitrates, such that we perform a bitrate sweep to quantify the throughput to bit error rate relation. Figure 6 illustrates the results for the sweeps without interference in the upper plots.

Thermal. The bitrate sweeps for the thermal covert channel range from 5 bps to 300 bps on Haswell and from 1 bps to 20 bps on ARMv8. On both platforms we use a bitrate step size of 1 bps, a sampling period of 1 ms and Manchester encoding, similar to Bartolini et al. [3]. We show that it is possible to achieve throughputs of almost 100 bps with less than 1% bit errors on the Haswell platform, and up to 5 bps on ARMv8. Hence, we outperform the previous implementation on Intel Haswell [3]. However, our results validate the chosen model and show that the new capacity bounds are tight.

Cache. In the original work, the channel accesses were performed asynchronous accesses at the highest possible rate. Therefore the throughput could not be controlled and it was not possible to show how the transmission rate influences the bit error rate. In contrast, our implementation allows to set the transmission rate and channel accesses are timed. Therefore we can perform a bitrate sweep from 6.4 kbps to 5 Mbps with a step size of 64 kbps to show how the bit error rate changes with increasing throughput. However, the timekeeping necessary to control the transmission rate limits our maximum sampling period. If the sampling rate approaches 5 μs when sampling only one cache set, we start to observe transmis-

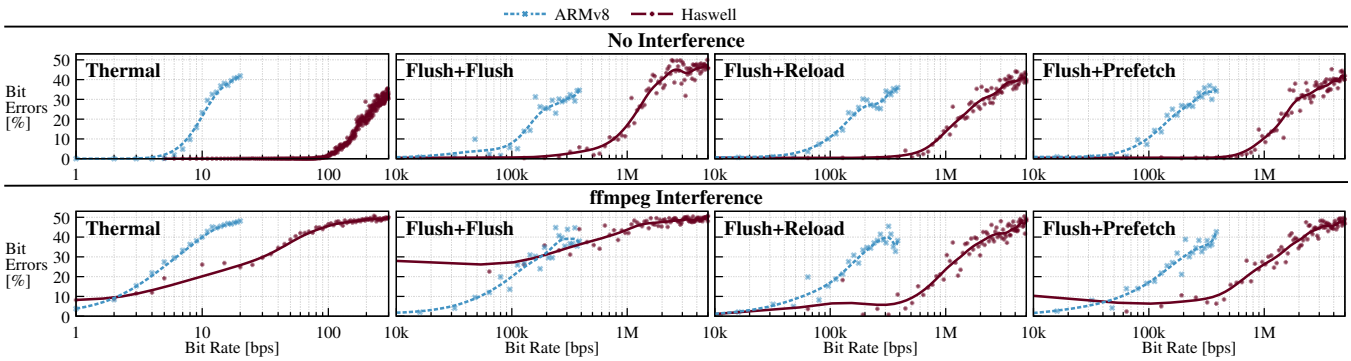

Figure 6: Without interference the bit error increases similarly for all three cache covert channels, whereas higher throughputs can be achieved on Haswell. The throughputs of the thermal and the Flush+Flush covert channel are more deteriorated by the ffmpeg interference, whereas the effect are smaller on ARMv8.

sion errors due to inaccurate timing, i. e., channel accesses are not synchronised. To be able to achieve high bitrates with longer sampling periods, we therefore use 64 cache sets with a spacing of 16 sets in parallel. This allows us to sample every 22 μs on Haswell and 225 μs on ARMv8. We determine the sampling rate on both platforms using the worst case time per channel use for one set from Table 2, plus a small security margin. On Haswell, this configuration allows us to establish transmissions with less than 1% bit errors at almost 200 kbps using Flush+Flush, 300 kbps with Flush+Reload and around 500 kbps for Flush+Prefetch. On ARMv8 the channels allow throughputs of around 15 kbps using Flush+Flush, 40 kbps with Flush+Reload and Flush+Prefetch, for less than 1% bit errors. The discrepancy of the results in comparison with original work (see Table 1) can be pinned to differences of the implementation and platforms. In addition, the experimental results show a higher difference between the capacity bound and the achieved throughputs, compared to the thermal channel. This is caused by limitations of our implementation, namely the limitation of cache sets and sampling rate we can use, rather than channel effects. Moreover, the capacity bounds are not as tight as they are based on the optimistic assumption that all channel accesses can be done with the best case time reported in Table 2. Therefore we consider the experimental evidence sufficient to validate the capacity bounds.

Interference. To evaluate the performance of the covert channels under noisy conditions, we repeated the sweeps while running a jammer application. To cause additional utilisation of the cores and cache operations, we start video encoding using ffmpeg 1 s prior to the transmission. The results illustrated in the lower plots in Figure 6 show that both channels suffer from interference. The influence of interference is lower on the ARMv8 platform, as it is optimised for tasks like video processing. Furthermore, the cache channels would also allow to use multiple lines to increase robustness of the transmission through redundancy, rather than increasing the throughput using parallelism. Therefore we consider the cache covert channel to be more robust and ultimately posing a higher threat than the thermal covert channel.

Remarks. Our results show that throughput results can vary to a great degree for different implementations. This illustrates the necessity of the capacity bound as a metric for comparing data leaks rather than throughput to compare implementations. In addition, our evaluation also shows the importance for experimental evidence to validate the models and metrics.

6. Concluding Remarks and Future Outlook

In this work we proposed a *methodology* for covert channel evaluation and presented the *Experiment Orchestration Toolkit ExOT*. Our methodology ensures that covert channels analysis results are reproducible, comparable and expressive. ExOT reduces the engineering effort needed to execute the presented methodology. The source code and documentation of ExOT is *publicly available* for download on our website [14].

We show the effectiveness of our methodology and ExOT by evaluating a thermal and three variations of cache covert channels. In our evaluation we capture different aspects of the data leaks by determining capacity bounds as well as empirical throughput and error rates. Such comparable metrics are fundamental to determine which data leaks need immediate action and which ones can be tolerated.

Acknowledgements

The authors would like to thank Naomi Stricker for proofreading the paper and the reviewers for the valuable feedback.

References

[1] NVIDIA Tegra Linux Driver Package Development Guide 32.2 Release. https://docs.nvidia.com/jetson/l4t/index.html, 2018.

[2] J. 200. International vocabulary of metrology–basic and general concepts and associated terms, 2012.

[3] D. B. Bartolini, P. Miedl, and L. Thiele. On the Capacity of Thermal Covert Channels in Multicores. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys

'16, pages 24:1–24:16, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4240-7. doi: 10.1145/2901318.2901322. URL http://doi.acm.org/10.1145/2901318.2901322.

[4] D. Evtyushkin and D. Ponomarev. Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 843–857. ACM, 2016.

[5] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard. Prefetch side-channel attacks: Bypassing smap and kernel aslr. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 368–379. ACM, 2016.

[6] D. Gruss, C. Maurice, K. Wagner, and S. Mangard. Flush+ flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299. Springer, 2016.

[7] M. Lipp, D. Gruss, R. Spreitzer, and S. Mangard. Armageddon: Last-level cache attacks on mobile devices. *CoRR abs/1511.04897*, page 169, 2015.

[8] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun. Thermal Covert Channels on Multi-core Platforms. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 865–880, Washington, D.C., Aug. 2015. USENIX Association. ISBN 978-1-931971-232. URL https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/masti.

[9] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, S. Mangard, and K. Römer. Hello from the other side: SSH over robust cache covert channels in the cloud. *NDSS, San Diego, CA, US*, 2017.

[10] P. Miedl and L. Thiele. The Security Risks of Power Measurements in Multicores. In *Proceedings of the 2018 ACM symposium on Applied computing*. ACM, 2018.

[11] P. Miedl and L. Thiele. Capacity calculations in "Increased reproducibility and comparability of data leak evaluations using ExOT". http://hdl.handle.net/20.500.11850/378017, 03 2020.

[12] P. Miedl, X. He, M. Meyer, D. B. Bartolini, and L. Thiele. Frequency Scaling as a Security Threat on Multicore Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2497–2508, 2018.

[13] P. Miedl, B. Klopott, and L. Thiele. Data: Thermal and cache covert channel analysis with ExOT. http://hdl.handle.net/20.500.11850/378872, 03 2020.

[14] P. Miedl, B. Klopott, and L. Thiele. ExOT Website. https://www.exot.ethz.ch/, 03 2020.

[15] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[16] Y. Yarom. Mastik: A Micro-Architectural Side-Channel Toolkit. https://cs.adelaide.edu.au/~yval/Mastik/, 2016. Accessed 21st of May 2019.

[17] Y. Yarom and K. Falkner. FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, 2014.