Analysis and Detection of Cache-Based Exploits

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieur (Dr.-Ing.)
von

Tsvetoslava Vateva-Gurova, M.Sc., geb. Vateva,
aus Botevgrad, Bulgarien

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Referenten:	Prof. Dr. Guido Salvaneschi
	Prof. Neeraj Suri, Ph.D.
	Prof. Dr. Stefan Katzenbeisser

Datum der Einreichung: 29. Oktober 2019
Datum der mündlichen Prüfung: 11. Dezember 2019

Darmstadt 2019
D17

Analysis and Detection of Cache-Based Exploits

by

Tsvetoslava Vateva-Gurova

ERKLÄRUNG

Hiermit versichere ich, die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 29. Oktober 2019

Tsvetoslava Vateva-Gurova

ABSTRACT

Over the last decades the digitalization has become an integral part of daily life. Computer systems complexity has also grown at a rapid pace. New business models have emerged to optimize utilization and maintenance cost of these complex systems, but neglecting the introduction of new security threats. Cloud computing, for instance, has been established as an important part of the modern IT infrastructure ignoring the potential security risks entailed in its pervasive usage.

A popular threat in the Cloud and other complex systems that are reliant on the usage of shared resources stems from the exploitation of side channels. In the context of co-location of mutually untrusted users on the same hardware, the confidentiality of user data has to be guaranteed. However, the class of side-channel and covert-channel attacks has been demonstrated to circumvent the secure isolation between co-located users both in the Cloud and in a native environment by exploiting hardware side effects, e.g., through timing analyses of accesses to CPU caches. The threat related to these exploits has been known for decades, but its relevance has grown with the increasing popularity of Cloud services. In this context, the cache is leveraged as an illegal channel to convey information either from one adversary to another in a covert-channel attack or to leak information from a victim to an attacker in a side-channel attack. As cache usage does not require any privileges, addressing the threat resulting from such an exploit turns out to be a challenging task.

On this background, this thesis aims at enhancing systems security by considering the cache-based covert-channel and side-channel attacks. We develop a classification of existing attacks by exploring their feasibility depending on the execution environment context, and construct an information leakage model which includes the CPU scheduling effect on the core-private cache exploitability. To delve into the specifics of detecting cache exploits, we define a set of indicators of compromise and investigate their correlation with the success of a core-private cache exploit. To account for the effect of the hypervisor scheduling configuration on the exploitability of the core-private cache, we empirically assess the success of a covert-channel attack while varying hypervisor scheduling parameters. We employ software events and performance counters to develop a reliable detection mechanism tailored to find contemporary side-channel attacks.

The results presented in the thesis demonstrate that by utilizing deliberately selected indicators of compromise along with a comprehensive analysis, systems security can be significantly enhanced with respect to the cache exploitability.

KURZFASSUNG

In den letzten Jahrzehnten hat sich die Digitalisierung zu einem festen Bestandteil des täglichen Lebens entwickelt. Die Komplexität von Computersystemen hat ebenfalls zugenommen. Neue Geschäftsmodelle sind entstanden, um die Auslastung und die Wartungskosten zu optimieren, aber die damit verbundene Einführung neuer Sicherheitsbedrohungen wird häufig vernachlässigt. Cloud Computing hat sich als wichtiger Bestandteil der modernen IT-Infrastruktur etabliert trotz der potenziellen Sicherheitsrisiken, die mit seiner allgegenwärtigen Nutzung verbunden sind.

Eine verbreitete Bedrohung in der Cloud und anderen komplexen Systemen, die auf die Verwendung gemeinsam genutzter Ressourcen angewiesen sind, resultiert aus der Ausnutzung von Seitenkanälen. Diese Bedrohung ist seit Jahrzehnten bekannt, aber ihre Relevanz hat mit der zunehmenden Popularität von Cloud-Diensten zugenommen. In diesem Kontext kann der Cache als Seitenkanal genutzt werden, um die Datenübertragung zwischen Gegnern bei einem verdeckten Kanalangriff oder Datenexfiltration von einem Opfer bei einem Seitenkanalangriff zu ermöglichen. Da für die Verwendung des Caches keine Berechtigungen erforderlich sind, gestaltet sich die Bekämpfung der Bedrohung durch einen solchen Exploit als herausfordernd.

Vor diesem Hintergrund zielt diese Dissertation darauf ab, die Systemsicherheit zu verbessern, indem cache-basierte Seitenkanalangriffe berücksichtigt werden. Wir entwickeln eine Klassifizierung bestehender Angriffe, indem wir ihre Durchführbarkeit in Abhängigkeit vom Kontext der Ausführungsumgebung untersuchen und ein Modell erstellen, das den CPU-Schedulingeffekt auf die Ausnutzbarkeit des Core-Private-Cache einschließt. Um die Besonderheiten der Erkennung von Cache-Exploits zu untersuchen, definieren wir Gefährdungsindikatoren und untersuchen deren Korrelation mit dem Erfolg eines Core-Private-Cache-Angriff. Um die Auswirkung der Hypervisor-Scheduling-Konfiguration auf die Ausnutzbarkeit des Core-Private-Cache zu berücksichtigen, wird der Angriffserfolg unter Variation der Hypervisor-Scheduling-Parameter empirisch bewertet. Mithilfe von Software-Events und Performance Counters entwickeln wir einen zuverlässigen Erkennungsmechanismus, der auf das Auffinden aktueller Seitenkanalangriffe zugeschnitten ist.

Die in der Dissertation vorgestellten Ergebnisse zeigen, dass durch die Verwendung bewusst ausgewählter Gefährdungsindikatoren zusammen mit einer umfassenden Analyse die Systemsicherheit in Bezug auf die Cache-Ausnutzbarkeit erheblich verbessert werden kann.

ACKNOWLEDGMENTS

Pursuing the PhD was a long and intriguing journey. I was happy to spend my PhD years at the DEEDS group and to meet nice people along the way. I would like to thank my advisor Neeraj Suri for giving me this opportunity and the freedom to develop and explore my own research interests. I am grateful for your advises and your guidance throughout the whole PhD endeavour! I am also very grateful to Guido Salvaneschi for being my internal referent and to Stefan Katzenbeisser for accepting to be my external reviewer. I am thankful to Kirstin Peters, Christian Reuter and Thomas Schneider for being on my committee.

Pursuing the PhD requires an active research environment. Therefore, I would like to thank all my colleagues in the DEEDS group for being always open for discussions. I am thankful to my officemates Salman, Ruben and Jesus for the nice atmosphere in the office. I consider myself lucky in this regard. Although my Spanish did not improve tremendously, I learned a lot about Barcelona, Mallorca, football, the Pakistani culture and cuisine throughout the years! I am happy to have had discussions on formal modeling and diverse other technical and non-technical stuff with Habib and to get Nico's opinion on various topics related to our joint work and far beyond. I am glad for having supervised theses jointly with Olli and for our fruitful technical and non-technical discussions. I am thankful to Sabine for having a friendly ear for all my problems and being always helpful with the paperwork. I am also thankful to Ute for dealing with HotCRP and the website and for being ready to help. I am grateful for the feedback I have received throughout the years, for the nice lunch/coffee breaks and the insights I gained about karma, mustard and other important things in life. It's been a nice time! I am also happy that I met my former colleagues Heng, Thorsten, Ahmed, Hatem, Hamza, Daniel, Reza, Kubilay and Stefan. Each of them has contributed to the DEEDS spirit in their own way.

Last but foremost, I would like to thank to my family for being loving, supportive, patient and encouraging throughout all the years. Mostly, I am grateful for having my little daughter who taught me what is important in life, how to be more flexible and open-minded, and that everything is possible if you "just give it a try". Thanks Iliya, for your patience, love and advises. It has always been a great pleasure for me to have you by my side in stressful and less stressful times and I do know how lucky I am. Special thanks go to my parents Genka and Georgi for teaching me everything in life, being supportive and caring every time I needed it. I owe them a lot! Thanks also to my brother Nikolai for having an idea how to tackle whatever problem I had. I am grateful to my grandmothers and grandfathers for their love and positive energy and to my parents-in-law Anka and Pano for helping out any time it was needed. I feel extremely lucky for the family I got and appreciate their contribution to finish the PhD.

PUBLICATIONS

The following publications have, in parts verbatim, been included in this thesis.

[Vat+14a] T. Vateva-Gurova, J. Luna, G. Pellegrino, and N. Suri. "Towards a Framework for Assessing the Feasibility of Side-channel Attacks in Virtualized Environments". In: *Proc. of the 11th International Conference on Security and Cryptography - Volume 1: SECRYPT 2014, Vienna, Austria*. INSTICC. SciTePress, 2014, pp. 113–124. doi: 10.5220/0005052101130124

[VSM15] T. Vateva-Gurova, N. Suri, and A. Mendelson. "The Impact of Hypervisor Scheduling on Compromising Virtualized Environments". In: *Proc. of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, DASC 2015, Liverpool, United Kingdom, October 26-28*. 2015, pp. 1910–1917. doi: 10.1109/CIT/IUCC/DASC/PICOM.2015.283

[Vat+18] T. Vateva-Gurova, S. Manzoor, Y. Huang, and N. Suri. "InfoLeak: Scheduling-Based Information Leakage". In: *Proc. of the 23rd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC) PRDC'18, Taipei, Taiwan, December 4-7, 2018*. 2018, pp. 44–53. doi: 10.1109/PRDC.2018.00015

[VG+19] Tsvetoslava Vateva-Gurova, Salman Manzoor, Ruben Trapero, and Neeraj Suri. "Protecting Cloud-Based CIs: Covert Channel Vulnerabilities at the Resource Level". In: *Proc. of Information and Operational Technology Security Systems, Heraklion, Crete, Greece, September 13, 2018*. Ed. by Apostolos P. Fournaris, Konstantinos Lampropoulos, and Eva Marín Tordera. Cham: Springer International Publishing, 2019, pp. 27–38

[VGCS] T. Vateva-Gurova, N. Coppik, and N. Suri. "SpyAlarm: Be the Spy and Spy the Attacker". In: *Transactions on Dependable and Secure Computing* (). [under submission]

The following publications are related to different aspects covered in this thesis and have been published during my doctoral studies, but have not been included in the thesis.

[Hei+19] Markus Heinrich, Tsvetoslava Vateva-Gurova, Tolga Arul, Stefan Katzenbeisser, Neeraj Suri, Henk Birkholz, Andreas Fuchs, Christoph Krauß, Maria Zhdanova, Don Kuzhiyelil, Sergey Tverdyshev, and Christian Schlehuber. "Security Requirements Engineering in Safety-Critical Railway Signalling Networks". In: *Security and Communication Networks* 2019 (2019). doi: 10.1155/2019/8348925

[VS18] Tsvetoslava Vateva-Gurova and Neeraj Suri. "On the Detection of Side-Channel Attacks". In: *Proc. of the 23rd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC) PRDC'18, Taipei, Taiwan, December 4-7, 2018*. 2018, pp. 185–186

[Bir+18] Henk Birkholz, Christoph Krauß, Maria Zhdanova, Don Kuzhiyelil, Tolga Arul, Markus Heinrich, Stefan Katzenbeisser, Neeraj Suri, Tsvetoslava Vateva-Gurova, and Christian Schlehuber. "A Reference Architecture for Integrating Safety and Security Applications on Railway Command and Control Systems". In: *International Workshop on MILS: Architecture and Assurance for Secure Systems, Luxembourg, 25.06.2018*. Zenodo, 2018. doi: 10.5281/zenodo.1314095

[Man+18] Salman Manzoor, Tsvetoslava Vateva-Gurova, Rubén Trapero, and Neeraj Suri. "Threat Modeling the Cloud: An Ontology Based Approach". In: *Proc. of Information and Operational Technology Security Systems, Heraklion, Crete, Greece, September 13, 2018*. 2018, pp. 61–72

[Sch+17b] Christian Schlehuber, Markus Heinrich, Tsvetoslava Vateva-Gurova, Stefan Katzenbeisser, and Neeraj Suri. "Challenges and Approaches in Securing Safety-Relevant Railway Signalling". In: *Proc. of the 2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017*. 2017, pp. 139–145. doi: 10.1109/eurospw.2017.63

[Sch+17a] Christian Schlehuber, Markus Heinrich, Tsvetoslava Vateva-Gurova, Stefan Katzenbeisser, and Neeraj Suri. "A Security Architecture for Railway Signalling". In: *Proc. of the 36th International Conference on Computer Safety, Reliability, and Security, SAFECOMP 2017, Trento, Italy, September 12*. IEEE, 2017, pp. 320–328

[Vat+14b] Tsvetoslava Vateva-Gurova, Jesus Luna, Giancarlo Pellegrino, and Neeraj Suri. "On the Feasibility of Side-Channel Attacks in a Virtualized Environment". In: *E-Business and Telecommunications, ICETE 2014, Vienna, Austria, August 28-30, 2014, Revised Selected Papers*. Ed. by Mohammad S. Obaidat, Andreas Holzinger, and Joaquim Filipe. Springer International Publishing, 2014, pp. 319–339. doi: 10.1007/978-3-319-25915-4_17

[Ves+14] Fatbardh Veseli, Tsvetoslava Vateva-Gurova, Ioannis Krontiris, Kai Rannenberg, and Neeraj Suri. "Towards a Framework for Benchmarking Privacy-ABC Technologies". In: *Proc. of the 29th IFIP TC 11 International Conference on ICT Systems Security and Privacy Protection, SEC 2014, Marrakech, Morocco, June 2-4, 2014*. 2014, pp. 197–204. doi: 10.1007/978-3-642-55415-5_16

[Lun+13] Jesus Luna, Tsvetoslava Vateva-Gurova, Neeraj Suri, Massimiliano Rak, and Alessandra De Benedictis. "SecLA-Based Negotiation and Brokering of Cloud Resources". In: *Cloud Computing and Services Science–Third International Conference, CLOSER 2013, Aachen, Germany, May 8-10, 2013, Revised Selected Papers*. Springer International Publishing, 2013, pp. 1–18

[Gar+13] Jesus Luna Garcia, Tsvetoslava Vateva-Gurova, Neeraj Suri, Massimiliano Rak, and Loredana Liccardo. "Negotiating and Brokering Cloud Resources based on Security Level Agreements". In: *Proc. of the 3rd International Conference on Cloud Computing and Services Science - Volume 1: CloudSecGov, (CLOSER 2013)*. SciTePress, 2013, pp. 533–541

[Gar+12] Jesus Luna Garcia, Hamza Ghani, Tsvetoslava Vateva, and Neeraj Suri. "Quantitative Assessment of Cloud Security Level Agreements - A Case Study". In: *Proc. of the International Conference on Security and Cryptography - Volume 1: SECRYPT 2012, Rome, Italy*. 2012, pp. 64–73

[Dem+18] Kubilay Demir, Hatem Ismail, Tsvetoslava Vateva-Gurova, and Neeraj Suri. "Securing the Cloud-Assisted Smart Grid". In: *International Journal of Critical Infrastructure Protection (IJCIP)* 23 (2018), pp. 100–111. doi: 10.1016/j.ijcip.2018.08.004

CONTENTS

1 INTRODUCTION

Security is always excessive until it is not enough

Robbie Sinclair

Computer systems have become a significant part of our lives while evolving at a rapid pace. Over the last decades they have continuously grown in complexity both in terms of hardware and software. For years hardware has undergone a considerable improvement in speed and space utilization complying with Moore's law [Mo000], which has predicted a doubling of the number of transistors every second year. Although Moore's law does not necessarily characterize the current hardware development trend anymore [Hea18], hardware is still becoming more complex to compensate, e.g., through massively parallel architectures. Due to this rapid development, the existing hardware has been able to easily solve general purpose tasks for decades now which enables the ubiquitous usage of computing power.

As a result, almost every area of nowadays life is dependent on digitalisation or computing. Business, administration, medical care and research are just a few examples of domains that rely on modern computing. Consequently, more and more personal data such as medical records or browsing history is collected in a digital form for various purposes with or without user consent. As an example, advertisement services have become reliant on targeting their audience by analyzing data focused on customer interests. The increase in data gathering practices has become so ubiquitous that legislation was introduced to rein the collection of personal data. The European Union adopted the General Data Protection Regulation (GDPR) [Gdp] in 2016 to regulate the gathering of personal data related to individuals in the European Union. A similar proposal, known as the California Consumer Privacy Act [Ccp], was enacted on the 28th of July, 2018 in California to impose new data protection requirements on companies while granting new rights to the consumers with respect to the collection of their personal data.

The software that enables this rapid development and the processing of huge amounts of data also grows in size and complexity. Spinellis et al. show in their work [SA19] that the codebase of the modern operating systems has drastically grown. For instance, the number of lines of kernel code for Research PDP7 has evolved from 2489 in 1970 to about 8518968 lines of kernel code in FreeBSD 11.0.0 in 2016 [SA19]. Similarly, the number of lines of code of OpenSSL [Ope] has increased from less than 200000 in 2000 to more than 500000 in 2019. In addition, the interconnectivity of various types of devices (cars, mobile phones, etc.) has also increased. Concepts such as Internet of Things and Artificial Intelligence dealing with huge amounts of data, and creating sophisticated prediction models are now well-established. The systems are getting more complex in every thinkable dimension.

The increased usage of and reliance on the Internet and digitalization has emphasized the need for solutions that facilitate the optimized utilization of resources, and concepts focused on the shared usage of resources have emerged. Cloud computing services, for instance, have gained much popularity for their cost efficiency, high availability and on-demand up and down scaling. This is why Cloud computing has become prevalent for numerous applications. According to Gartner, Inc., the Cloud services industry will grow exponentially through 2022 at nearly three times the growth of overall IT services [Gar19]. Moreover, the Healthcare Information and Management Systems Society (HIMMS) Analytics conducted a study in 2014 [HIM14] which confirmed the increasingly wide-spread adoption of Cloud services in critical sectors such as healthcare which deal with highly sensitive private information. Being widely used even in critical sectors, Cloud computing providers need to guarantee secure environments for their customers. Relying intrinsically on shared resources, the Cloud customers have to be securely isolated from each other without the possibility to leak information from the co-located customers.

Along with the benefits and advantages the technological innovations brought along, the risk of loosing control over personal data and the unpredictability of software behavior, especially in the context of increasing system complexity and resource sharing, has risen. The growth in software complexity leads to potential increases in the attack surface and can open up new vulnerabilities. The National Vulnerability Database (NVD) [Nis] maintained by the US National Institute for Standards and Technology (NIST) contains more than 16000 vulnerabilities (i.e., flaws in software which can be exploited by an attacker) reported in 2018 compared to less than 2000 vulnerabilities reported in 2003 and approximately 2500 vulnerabilities reported in 2004 [Nis]. The increased number of vulnerabilities demonstrates the relevance of security nowadays in the complex IT world and indicates the need for rigorous methods to address the existing security issues.

1.1 SECURITY IN A COMPLEX IT WORLD

In the context of wide-spread data collection and sharing of resources across security boundaries, the importance of security and in particular the confidentiality of data only increases. Data confidentiality "deals with protecting against the disclosure of information by ensuring that the data is limited to those authorized or by representing the data in such a way that its semantics remain accessible only to those who possess some critical information (e.g., a key for decrypting the enciphered data)" according to the definition provided by NIST [Infa]. The data confidentiality property guarantees that the data is not made available or disclosed to unauthorized entities, individuals or processes, and the confidential information has to be kept securely disallowing unauthorized access. It is usually achieved by the means of encryption.

Data confidentiality is vital to Cloud computing systems due the usage of shared resources across security boundaries. In fact, the requirement of secure isolation between co-located customers dates back decades. Commonly, the underlying technology in Cloud computing is virtualization which enables the co-location of users across security domains encapsulated in virtual machines. Among the early requirements on virtualization, as defined by Popek [PG74], is the isolation of a virtual machine. This property includes the requirement that the co-located virtual machines cannot compromise the confidentiality of each other. Today, with the increased complexity of computer systems and software, the relevance of this requirement has increased.

Although security is a necessary property for a variety of application scenarios, addressing security requirements can be challenging. For already well-established legacy systems enhancing security entails an increase in terms of cost, and might introduce performance overhead or usability issues. Depending on the system, fixing security issues or adapting to changing threats may even require renewing the standardization of the system, and performing a whole standardization procedure from the beginning. This is, for example, often the case in critical infrastructures such as railway systems. Such issues make system administrators sometimes reluctant to incorporate proper security solutions into their systems unless it is urgently necessary. Furthermore, the adoption of new security solutions and replacing of old ones is often cumbersome and slow.

This is a wide-spread issue in cryptographic software, where legacy ciphers are frequently supported long after their deprecation, which can lead to exploiting security flaws. In 2016 the Sweet32 vulnerability (CVE-2016-2183 and CVE-2016-6329) has been reported to exploit the 3DES encryption algorithm [BL16]. The 3DES algorithm has been considered a legacy cipher for quite some time, and the reported attack emphasizes the need to retire weaker algorithms such as 3DES. Despite this fact, according to the authors of the reported vulnerability, 3DES had been used for about 1-2% of all HTTPS connections between mainstream browsers and web servers. About a year after the Sweet32 vulnerability report, in July 2017, NIST has withdrawn the approval for using 3DES in protocols such as TLS and IPSec. NIST has declared the algorithm as officially retired in a draft guidance published in July, 2018 [BR19]. Similarly, Microsoft announced in April 2019 that 3DES algorithm will be retired beginning of July 2019, about three years after the Sweet32 vulnerability report [Ari19]. This example confirms that the adoption of contemporary security solutions is commonly slow and cumbersome, and compromised solutions might be still in use.

The human factor is often an additional reason for the slow changes related to systems security. User frustration and lack of knowledge often pose a challenge when incorporating security measures to existing systems. Recently, Stark et al. [Sta+19] reported on their experience on deploying a security solution that tackles the danger of improper certificate issuance and can provide desirable security benefits, but its full deployment would represent a huge change to the HTTPS ecosystem. They also studied user behavior in the cases of breakage and came to the conclusion that user behavior is unsafe. Another experiment based on a cyber-physical systems game described in [Fre+17] demonstrated that even the security experts involved in the experiment tended to neglect intelligence gathering. Being that confident in their expertise, the security experts often skipped threat assessment which had implications on their performance within the study. Facing such challenges contributes to the cumbersome and slow addressing of security requirements and issues, and despite the pervasive deployment of cryptographic solutions to enhance system security, confidential data often remains insecure.

While addressing security issues in software and protocols is cumbersome and slow, current architectures are intrinsically insecure, and the vulnerabilities resulting from hardware properties are even harder to tackle. The employed hardware optimizations have resulted in a tremendous performance upgrade over the years, but are often related to security implications confirmed by the wide range of covert-channel attacks and side-channel attacks. These classes of attacks with their different manifestations are often a result of the hardware design exhibiting side effects. The existence of such powerful attacks that can compromise system's confidentiality is theoretically known for more than two decades. However, the system's performance is often at odds with the system's security and, in such cases, when security comes at a price, it is often being neglected.

1.2 EXPLOITING SIDE CHANNELS

A side-channel attack is defined as an attack that is enabled by information leakage stemming from a physical ecosystem [Infb]. These attacks are based on characteristics resulting from the shared usage of or access to resources such as timing, power consumption, acoustic and electromagnetic emissions, etc. Side-channel attacks pose a threat to confidentiality and are hard to detect or prevent. In a side-channel scenario, an adversary observes victim's usage of a shared component and can extract victim's sensitive data based on analysis of the collected observations.

A similar exploit represents the covert-channel attack. A covert channel, as defined in [Infc], is a channel which is not intended or authorized for communication, but enables the information transfer between two cooperating entities. This happens in a way that violates the system's security policy but the involved entities do not exceed their access authorizations. Similar to the side-channel attack, a covert-channel attack exploits hardware side effects such as timing information.

Nowadays, top security conferences are concerned with the investigation of the covert-channel and side-channel exploits, and devote a significant part of their programs to the threat related to the misuse of side effects [Ccs; Spp]. At the same time, major providers such as VMWare or Amazon provide solutions that exclude different side-channel exploit possibilities [Ama14; VMwb]. This trend emphasizes the relevance of the covert-channel and side-channel exploits. Side-channel and covert-channel attacks abusing the side effects of the cache are in the focus of the thesis and are discussed in more detail in Part I. Within the thesis, the term "side channel" is used to denote the leakage channel in the context of both side-channel attacks and covert-channel attacks.

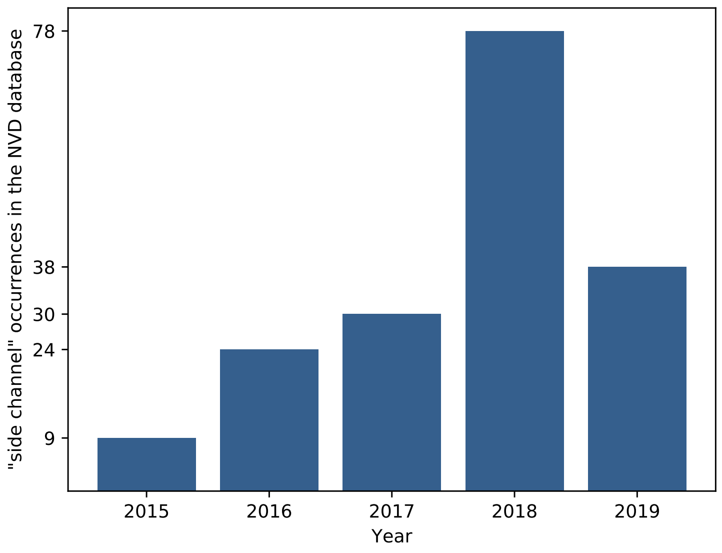


Figure 1: Occurrences of Side-Channel Vulnerabilities in the NIST National Vulnerability Database [Nis].

Currently, the research focuses on (i) proposals of new ways to exploit varied features of the computer architectures which emanate side effects; (ii) novel measures to combat the proposed exploits; and (iii) ways to detect them. In the course of the last decade security gaps stemming from the abuse of side effects have been closed, but new gaps have been opened up. This trend shows that the side-channel problem has not been solved. The number of the officially published vulnerabilities in NVD [Nis] emanating from side channels and exhibiting from low to critical severity has risen over the last years, as shown in Figure 1. On this background, what is currently clear is that side-channel and covert-channel attacks are hard to tackle despite representing a serious threat to the complex computer world and despite the extensive research being conducted on the topic.

The side-channel threat is basically related to all the thinkable components of a device, ranging from a power adapter (cf., [MDS99]), monitor (cf., [Gen+19]), keyboard (cf., [GST14; SWT01]) to cache memory (e.g., [Per05; Yan+19]). Commonly, each risky component is considered separately, and often mitigation strategies tailored to combat specific attacks are developed and proposed. Then, new manifestations of, theoretically, the same attack are discovered which might focus on a different component. A recent example for this development is the work conducted by Yan et al. [Yan+19] which demonstrates the feasibility of side-channel attacks on the sliced non-inclusive last-level cache of Intel Skylake-X architecture which has been considered more secure against last-level cache-based exploits. This work reconfirms that side-channel attacks are not an elusive threat, and research on them is expected to continue at least at the same pace.

1.3 RESEARCH QUESTIONS AND CONTRIBUTIONS

This thesis investigates the research questions stated below, and extends the field of research by making the contributions summarized under the respective research questions. The overall goal of this work and the common line in the research questions is the investigation of cache-based side-channel and covert-channel attacks, referred to as cache-based exploits, and their interplay with the execution environment. In the focus of the thesis are both the effect of the execution environment on the cache-based exploits, and the effect of these exploits on the execution environment, i.e., their possible traces. The thesis aims at enhancing the security of systems relying on shared resources by providing a side-channel attacks detection mechanism, and assessing their interaction with the execution environment. The individual research questions are discussed below.

Research Question 1 (RQ1) Can we classify and model side-channel exploits considering the impact of the execution environment on their feasibility?

Side-channel and covert-channel exploits have been a known threat for decades, and they continue to grow in popularity. Such exploits are especially relevant in Cloud scenarios where resource sharing across security boundaries is commonplace. It has been demonstrated that side-channel attacks can enable data exfiltration and allow attackers to extract confidential information, such as secret keys. Currently, there is no universal solution for coping with this threat, and this is aggravated by the fact that the diversity of side-channel attacks continues to grow. Still, not every reported side-channel attack is feasible under any condition. While side-channel exploitation in the real-world commonly depends on particular properties of the execution environment (e.g., scheduling), the explicit consideration of these properties is often neglected.

Therefore, it will be beneficial to classify the side-channel attacks while considering the factors and properties which characterize the assumed execution environment. Formal modeling of the side channels that include the execution environment's impact is necessary to understand the risk of a particular cache-based exploit in a specific system.

Contribution 1 (C1): A classification approach of side-channel exploits and a side-channel attack model that take execution environment factors into account

A panoply of resources can be exploited to leak information as a side channel. To better understand the risk associated with the exploitation of these resources, this thesis proposes a classification approach for the side-channel exploits. Depending on the context, these attacks can be fairly difficult to conduct in a real-world setting, or their execution can be facilitated, e.g., due to the scheduling configuration.

We provide a classification framework that aims to describe the side-channel attacks based on their execution context to aid the analysis and assessment of their specific characteristics resulting in possible detection or mitigation mechanisms. Based on the proposed classification framework, we focus on the feasibility of side-channel attacks that exploit cache memory in a virtualized environment. The goal of this contribution is to facilitate the systematic reasoning in regard to the side-channel threat along the feasibility assessment and modeling with respect to exploits of cache side effects.

Additionally, we introduce InfoLeak, an information leakage model that ascertains the essential role of the CPU scheduler for exploiting core-private caches as side channels. InfoLeak illustrates the impact of the CPU scheduling on the availability of the core-private cache to the adversary that exploits it as a side channel. Our model enables security experts to consider the associated threat stemming from the core-private cache exploits by analyzing the available scheduling information. We provide an example on InfoLeak usage to demonstrate its applicability for studying the scheduling related logs for possible information leakage. This first contribution of the thesis is presented in Part II and is based, partly verbatim, on material from [Vat+14a; Vat+14b; Vat+18].

Research Question 2 (RQ2): What is the relationship between execution environment properties and the success and feasibility of side-channel attacks?

As side-channel mitigation approaches against particular attacks have been proposed to reduce the information leakage by tweaking the execution environment, it would be beneficial to analyze which of the execution environment properties affect the feasibility of the side-channel exploits and to what extent. Although various components can be exploited to leak information, certain classes of attacks might exhibit the same characteristics. For example, most of the cache-based approaches rely on timing of memory accesses to derive confidential information. Additionally, in a variety of scenarios the adversary needs frequent access to the side channel to be able to obtain sufficient observations of good quality or high granularity. This is related to the synchronization between the attacker and the victim which is of crucial importance for a number of cache-based adversary scenarios.

To this end, employing indicators or metrics which characterize the execution environment in regard to the synchronization possibility between a potential adversary and a victim in the context of core-private cache-based attacks is an enabler for finding potential side-channel attack traces. It is also important to investigate how the scheduling configuration affects the synchronization and consequently the chance of success of a cache-based side-channel exploit.

Contribution 2 (C2): Empirical evaluation of the impact of scheduling on side-channel attacks and their potential traces

The feasibility of side-channel attacks strongly depends on the context of the execution environment including the availability of the exploited shared resource. However, the characteristics of the specific attack must also be considered in order to evaluate the exploitability of an execution environment.

We characterize the context of the execution environment by proposing three attack indicators or metrics and study their correlation with the success of a core-private cache-based attack post-mortem. The proposed metrics are related to the means an attacker often employs to gain sufficient observations of the side channel when exploiting the core-private cache. We also discuss the applicability of the proposed feasibility metrics in a case study.

Additionally, we analyze the impact of the hypervisor scheduling configuration on the exploitability of the core-private cache as a side channel in a virtualized environment. For this purpose, we identify the relevant hypervisor scheduling parameters and conduct an empirical study on the success of the attack depending on different configurations. This contribution of the thesis is presented in Part III and is based, partly verbatim, on material from [VG+19; VSM15; Vat+18].

Research Question 3 (RQ3): Can side-channel attacks be reliably detected?

Although attacks at the architectural level represent a serious threat to data confidentiality, and enable information exfiltration, they are usually neglected by techniques, such as intrusion detection, which commonly focus on high-level network or middleware threats. This is magnified by the fact that side channels usually fall outside the scope of any security policies or access rules. Cache-based side channels, for instance, are accessed without any special privileges.

In this context and given the relevance of cache-based side-channel attacks, investigating the possibilities of detecting side-channel exploits is important for the security community. This is reinforced by the lack of a universal remedy against the side-channel threat posed by the current vulnerable cache design.

Contribution 3 (C3): A reliable side-channel attack detection approach using performance counters and software events

Existing approaches to mitigate or prevent covert-channel and side-channel attacks are usually limited by performance overhead, or they are too costly. Thus, incidents and anomalies related to the usage of the cache as a covert communication channel need to be detected, ideally without requiring new hardware. Traditional intrusion detection systems are not suitable for this purpose due to the lack of policies or access rights for the usage of the side channel.

In this contribution, we focus particularly on the detection of side-channel and covert-channel attacks that abuse the cache. We propose a side-channel detection approach, called SpyAlarm, based on the usage of metrics that indicate the feasibility of such attacks given the execution environment properties and their traces. SpyAlarm leverages a combination of performance counters and software events to reliably detect cache-based exploits while exhibiting a low false positive rate. In this contribution, we also present the SpyAlarm architecture and demonstrate its utility with a prototype implementation. To evaluate our approach, we apply it in several case studies including two state-of-the-art cache-based attacks previously unseen by the detector. Our results demonstrate the utility of SpyAlarm as a reliable side-channel detector, which enables triggering further actions to contain the damage caused by an attacker. This last contribution of the thesis is presented in Part IV and is based, partly verbatim, on material from [VGCS].

1.4 THESIS OUTLINE

The rest of the thesis is structured as follows: In Part I various side-channel exploit strategies are covered, and the system model on which this thesis relies is presented. In Part II, the first research question is discussed, and a side-channel attacks classification approach which focuses on the feasibility factors affecting the attack's success is presented. Part II also includes a leakage model that advocates the scheduling effect on the success of the attack. In Part III the thesis continues with the analysis of the impact of different scheduling parameters, and the footprint cache-based exploits leave on a set of measurable system events. The third research question, which tackles the detectability of cache-based covert-channel and side-channel attacks, is investigated in Part IV. Finally, Part V concludes the thesis with a summary of the contributions and key insights.

Part I

PRELIMINARIES

2

SIDE-CHANNEL AND COVERT-CHANNEL ATTACKS

2.1 OVERVIEW

Lampson is among the first researchers to mention the term "covert channel" as an illegal transmission channel which has not been designed to transmit information [Lam73]. Covert channels are leveraged in covert-channel attacks that involve two or more processes which collaborate to communicate with each other. The communication takes place through a shared resource which can be manipulated and measured by all the involved processes. There are various application scenarios for these attacks. For example, they can be used for secret transmission of information. The hardly detectable existence of such covert communication makes these transmission channels quite elusive. Among others, there are proposals to leverage covert channels to bypass censorship [Fea+02] or to use them to transmit authentication data [dAJ05].

Side-channel attacks can be seen as a variant of the covert-channel problem. Contrary to a covert-channel attack, where the cooperation between processes is the main goal of the collaborating attackers, a side-channel attack is a purely adversarial problem. A side-channel attacker spies on the actions of a co-located victim process and tries to extract secret data from the collected observations. This is possible due to the nature of the commonly used hardware exhibiting measurable side effects which often depend on the data or instructions being processed in various computations.

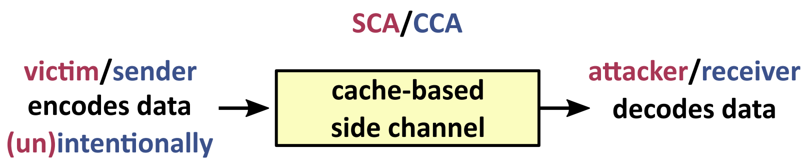


Figure 2: Covert-Channel Attack and Side-Channel Attack – Interaction with the Channel.

As can be seen in Figure 2, common for the covert-channel and the side-channel attacks is the usage of a shared resource as a transmission medium. If we consider the notion introduced by Shannon in his theory on communication [Sha01], the victim plays the role of the information source in a side-channel attack, whereas this role is taken over by the sender in a covert-channel attack. The channel is in both cases the shared medium. The transmitter that encode the message into the channel is the process of the victim that leaves traces in a side-channel attack, or it is the sender process in a covert-channel attack. The message itself can be a secret key in a side-channel attack and some other meaningful information in a covert-channel attack which might be encoded, e.g., as a cache footprint. In such a case, the cache footprint represents the signal. The attacker process, spying on the victim, or the receiver process, trying to decode the message, is the receiver of the communication.

A major distinction between a side-channel attack and a covert-channel attacker lies in the fact that the side-channel attacker has to conduct a thorough analysis of the gathered data which is more complicated than the analysis a covert-channel attacker has to conduct. The covert-channel adversary decrypts a message that has been sent to him/her over the shared resource intentionally by a collaborating attacker. Commonly, the covert-channel receivers are reliant on a predefined shared usage pattern of the shared resource with the sender.

To avoid confusion, the shared resource which can be exploited as a covert or side channel is in both cases referred to as a side channel from now on in the thesis. It has been demonstrated already that the cache can be exploited as a side channel also in complex systems such as the Cloud. As it is easily accessible without the need of any special privileges in various execution environments, the cache represents a powerful and convenient side channel.

2.2 THE CACHE AS A SIDE CHANNEL

As already discussed, the processors used to become faster complying with Moore's law. However, the memory speed, though increasing, did not develop at the same pace. To close the performance gap between the processors and the memory, caches were introduced as an important optimization, deployed universally. Depending on the applied caching algorithm, recently or frequently used data or instructions are fetched from the main memory into the cache and can be accessed much faster if they are needed repeatedly. If the requested data is already in the cache, a cache hit takes place and the latency for accessing the data is low. The smaller the cache containing the data, the faster the access to the requested data is. If the data is not in the cache, a cache miss takes place and the data has to be fetched into the cache from the main memory. This is, logically, associated with a longer latency and accessing the data takes more time than in the case of a cache hit.

Most computer architectures organize the caches in a hierarchical manner. Typically, there are two or three cache levels. The core-private caches, i.e., level 1 caches (L1), are faster and smaller and are located closer to the CPU. L1 cache is private per core and is usually divided into L1 instruction cache which stores instructions and L1 data cache which stores data. The last-level cache (LLC) is commonly shared among all cores and is unified, i.e., it stores both instructions and data.

Within the cache, the data is stored into units called cache lines or cache blocks. They are organized into cache sets. Each cache set consists of w cache lines, where w is referred to as cache associativity. A part of the memory address, often called set index, usually determines the cache set a cache line is mapped to.

In addition to that, the LLC of most modern Intel processors is further divided into slices mainly to reduce congestion [Int]. A hash function is applied to determine the slice of a cache line, but this hash function has not been published officially by the time of writing this thesis. Yarom et al. developed a technique to reverse-engineer the hash function for certain architectures and published their work in [Yar+15].

Furthermore, modern LLCs are often inclusive which means that they contain a strict superset of the contents of the data contained in the lower cache levels [Int].

The access times to the different cache levels and the memory are measurable and, practically, almost each process running on a system can collect observations on the access times to the cache. This variance in the access times to the caches depending on whether a cache hit or a cache miss takes place, is a hardware side effect which can be abused by an attacker and enables powerful side-channel and covert-channel attacks.

2.3 STRATEGIES TO EXPLOIT THE CACHE AS A SIDE CHANNEL

The cache was introduced to close the performance gap between the processors and the memory by overcoming the high latency of the memory. These universally deployed optimizations can be exploited by an attacker to leak confidential information from a process using the same cache.

On this background, there exist different strategies to abuse the cache to leak information. The existing attack strategies can be basically divided into attacks relying on the usage of shared memory between the attacker and the victim, and approaches that do not require shared memory. Among the well-known side-channel attack strategies are Evict+Time and Prime+Probe, described, e.g., [Liu+15; OST06; Per05], Flush+Reload demonstrated in [YF14], and Flush+Flush proposed in [Gru+16].

2.3.1 *Side-channel Attack Strategies which do not Require Shared Memory*

Unlike other strategies, Evict+Time and Prime+Probe do not rely on shared memory. They are characterized by a lower bandwidth compared to the approaches leveraging shared memory, but are still considered more powerful, as they do not depend on restrictions such as shared memory and therefore, can be applied on more systems.

2.3.1.1 *Evict+Time.*

Among the first strategies to exploit the cache as a side channel is Evict+Time, proposed by Osvik et al. [OST06]. Evict+Time assumes the ability to trigger an encryption of known plaintext $\mathbf{p}$ on victim's side and to be able to determine when the encryption starts and when the encryption process ends. Before the encryption, the adversary manipulates the cache state. Then, the attacker lets the victim encrypt the plaintext. After the encryption, the attacker accesses selectively chosen addresses to evict specific cache lines. Then, the adversary triggers another encryption of the same plaintext $\mathbf{p}$ on the victim's side and measures the time the encryption takes. Longer encryption times indicate that the selectively evicted cache lines had to be fetched from the main memory suggesting that the victim has accessed these specific cache lines. The collected timing data is analyzed to extract secret data. Bernstein applied a similar approach to extract an AES key, as described in [Ber05]. The main steps of the approach are summarized below.

1. Trigger encryption of plaintext **p**.

2. Evict specific cache lines by accessing appropriate addresses.

3. Trigger another encryption of plaintext **p** and measure the time the encryption takes.

Evict+Time is a time-driven side-channel attack which is reliant on measuring the overall execution time of victim's code. Evict+Time attackers typically observe the aggregated number of cache hits and cache misses which might result in coarse observation granularity. Despite that this approach still represents a security threat and was recently applied by Gras et al to demonstrate the insecurity of the Address space layout randomisation (ASLR) in [Gra+17].

2.3.1.2 *Prime and Probe.*

Another approach exploiting the cache as a side channel, called Prime+Probe, was described by both Osvik et al. in [OST06] and by Percival in [Per05]. Prime+Probe comprises three main steps. In the first step, the entire cache is filled with attacker's data. This step is often referred to as priming the cache. Then, in the second step, the victim encrypts some plaintext **p** which causes partial eviction of the data the adversary has filled in the cache with. In the third step, the adversary analyzes which cache parts have been evicted by the victim's encryption by measuring its own access times to the respective cache parts. This measurement step is called a "probe" step. To conduct the timing measurements the attacker reloads the same data into the cache as in the first step, but carefully observes how much time it takes. Analogously to Evict+Time, longer access times indicate cache evictions whereas shorter access times suggest that the victim has not touched the corresponding cache parts. Having collected the cache access time data, the attacker correlates them to cryptographic algorithms structure and tries to extract confidential information.

Prime+Probe comprises three main steps, as summarized below.

1. Attacker primes parts of the cache.

2. Victim process accesses or does not access the respective cache parts.

3. Attacker probes the same parts of the cache and measures the time for retrieving the data.

Prime+Probe was originally leveraged to abuse the core-private cache to leak confidential information, but in the last decade also approaches targeting at exploiting the LLC were proposed ,e.g., [Liu+15]. These approaches significantly increase the relevance of Prime+Probe, as they enable the cross-core exploits which makes their success less dependent of the CPU scheduling.

2.3.2 *Shared Memory and Side-Channel Attack Approaches*

Another class of cache-based side-channel attacks relies on the usage of memory pages shared between the attacker and the victim. Although such a requirement is restricting the applicability of these attacks, it is often fulfilled. The operating systems and virtual machine monitors strive for efficient memory management, and apply different optimization techniques. Among them is memory deduplication, also referred to as content-based page sharing, which is applied to reduce the overall memory footprint. This is achieved by merging memory pages with identical content within the same physical machine or across virtual machines. For example, virtual machines running on the same physical host might use identical operating system and/or libraries which increases their potential for memory deduplication. This technique has the potential to achieve significant memory optimization. Gupta et al., for instance, built an extension for the Xen virtual machine monitor to make use of different memory deduplication mechanisms and reported substantial memory savings in [Gup+08].

Due to its potential for optimization, memory deduplication is applied by a number of hypervisors and operating systems. For instance, Linux leveraged a module called KSM to find equal pages in the system [AEW09]. This module allows for sharing pages across different processes and KVM virtual machines. VMWare also applied a set of techniques to eliminate redundancy in memory and reduce the copying overheads [Wal02]. Due to its advantages most known operating systems and hypervisors make use of deduplicating memory.

However, along with the benefits it provides, page sharing among mutually untrusted processes exposes the involved processes to security risks and reveals new security threats [Bos+16; Suz+11]. Thus, it creates just another trade-off between memory optimization and security. Moreover, according to an empirical study conducted by Chang et al. and described in [CWL11], memory deduplication provides much less optimizations as previously reported. Chang et al. argue that the absolute sharing levels (which exclude zero pages) typically remain under 15%. At the same time, the memory sharing poses a serious security threat to the systems. Bosman et al. conducted a powerful attack against the Microsoft Edge browser [Bos+16], which was reported as a vulnerability CVE-2016-3272, and addressed by Microsoft by changing the defaults regarding the memory deduplication usage. VMWare also reacted to the security threat related to memory deduplication and changed the default settings for Transparent Memory Sharing [VMwb]. Among the strategies that exploit page sharing are Flush+Reload and Flush+Flush.

2.3.2.1 *Flush+Reload.*

Flush+Reload is an attack strategy proposed by Yarom et al. [YF14] in which an adversary exploits the ability to monitor victim's accesses to shared memory pages, which can be applied in a virtualized environment. The attack targets at the LLC cache which makes it a powerful mechanism to leak information cross-cores, i.e., it is independent of the co-location of the attacker and the victim on the same CPU core.

Flush+Reload is based on the observation that whenever a process accesses a shared page in memory, the accessed data is cached. Once the data has been cached, the attacker evicts deliberately chosen and monitored memory locations from the cache using the flush instruction (e.g., clflush). By repeatedly reloading the same data and measuring the time it takes for accessing them, the attacker infers whether the victim has accessed it in the meanwhile or not.

The abstract steps involved in a Flush+Reload attack are summarized below.

1. The adversary flushes a deliberately chosen memory location.

2. The victim performs operations.

3. The adversary access the same memory location again and measures the time.

The applicability of the attack proposed by Yarom et al. to the LLC increases its significance, as it enables an attacker to monitor the victim from another CPU core.

2.3.2.2 *Flush+Flush.*

Flush+Flush, proposed by Gruss et al. in [Gru+16], is an extension of the Flush+Reload attack, but it leverages only the timing variation of the flush instruction itself and is not reliant on the memory accesses.

Hence, Flush+Flush attacks are harder to detect by mechanisms analyzing the memory accesses and are considered more robust against detection mechanisms.

2.4 SYSTEM AND ATTACKER MODELS

This section details the system model that underlies this thesis. The presented system model is an abstraction of the system models applied in the individual contributions of the thesis. All of the contributions are focused on the cache as a side channel and consist of a hardware part comprising the CPU cores along with the different cache levels. It includes the core-private cache (i.e., L1 cache) and shared caches (e.g., LLC).

Depending on the contribution, either a hypervisor, called also virtual machine monitor (VMM), or an operating system (OS) runs on top of the hardware. The hypervisor is a software layer, which is a part of each virtualization solution and enables the multiplexing of tenants or users encapsulated in virtual machines on the same physical machine. A virtualized environment is characterized by providing a low-level abstraction from the hardware state. Both in the cases of an underlying hypervisor or OS, a secure isolation between the running virtual machines (in the case of a hypervisor) or between the processes (in the case of an OS) is presumed. The abstract system model is depicted in Figure 3.

Any attacker processes considered in this thesis is assumed to have control over non-privileged processes or virtual machines running on the system. The attacker(s) can indirectly manipulate the cache in the way any user-land process can do by accessing data or instructions. Attackers with privileged access to the system, such as the one described in [Bra+17], which can manipulate the

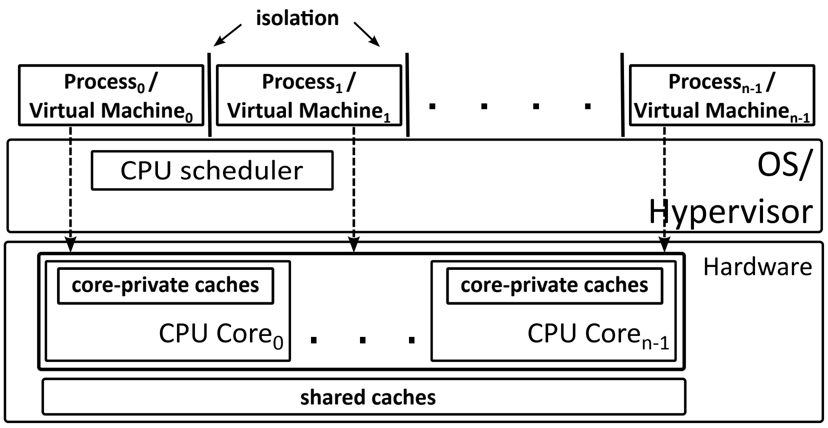


Figure 3: Abstract System Model.

performance counters, but cannot read victim's memory directly, are out of the scope of the thesis.

Part II

ON THE CLASSIFICATION, FEASIBILITY AND MODELING OF SIDE-CHANNEL ATTACKS

3 TOWARDS THE SYSTEMATIC REASONING ABOUT SIDE-CHANNEL ATTACKS

Millen defined four major research areas considering the side channels: (i) explaining, (ii) finding, (iii) measuring and (iv) mitigating side channels in [Mil99]. The objective of this chapter is to enable systematic reasoning about the side-channel attacks considering their execution environment which covers the explanation field, but is also connected to the rest of the defined research areas. The content of this chapter is, partly verbatim, based on material from [Vat+14a; Vat+14b] and tackles the first contribution of the thesis (cf., Section 1.3).

This chapter is organized as follows. Section 3.1 presents the motivation for this contribution. Section 3.2 gives an overview on a proposed generic classification approach. Section 3.3 focuses on feasibility factors that affect the exploits of the cache side effects in a virtualized environment. The related work is presented in Section 3.4, and the concluding remarks are given in Section 3.5.

3.1 CLASSIFICATION AND FEASIBILITY ENDEAVOURS

The term side channel denotes a communication channel that stems from the usage of shared resources and can be exploited, e.g., through manipulations or observations. As the traditional intrusion detection systems are not tailored to protect from such exploits, mainly due to the nature of the side channels, side-channel attacks are considered a serious threat for compromising the confidentiality within diverse execution environments. At the same time, with the advent of Cloud computing, the side-channel attacks relevance has even increased due to the intrinsic sharing of resources across security boundaries, which has led to a higher number of reported side-channel attacks (cf., Figure 1).

While these attacks represent a real threat to the security of any system, almost each manifestation of the attacks is commonly only successful given certain prerequisites or fulfilling certain assumptions. To better understand the side-channel threat and the feasibility of a specific attack, given the huge number of existing side-channel exploits, a generic classification that takes into account the specifity of the execution environment and the attack assumptions is needed. The lack of a generic side-channel attacks classification approach that encompasses both the type of exploit and the execution environment factors affecting its success impedes the systematic analysis of the side-channel attacks threat and the assessment of their feasibility. Such a classification approach has to take into account not only native, but also virtualized environments.

The term virtualization has been present in the IT community for a long time starting from the late sixties [PG74] until now, and has undergone periods of less popularity to gain significance again during the last decades [FDF05]. Despite the presumed strong logical isolation that virtualization provides, virtualized environments and systems offering shared resources or relying on virtualization (e.g., the Cloud) are vulnerable to side-channel exploits due to the inherent resource sharing among mutually untrusted customers. To address the security concerns regarding the side-channel attack threat, VMWare changed their hypervisor default settings following the demonstration of side-channel attacks which exploit memory deduplication [VMwb]. Amazon EC2 Dedicated Instances model also provides services dedicated explicitly to a single customer [Ama14]. This strongly emphasizes the efforts of not only the academic, but also the industrial world to address the concerns related to the side-channel attacks, especially in virtualized environments.

However, the security provided by such solutions comes at a price. Not using memory deduplication or using dedicated instances for a single customer potentially results in a worse utilization of resources which can increase the price of the services for the end users. Therefore, it would be beneficial to assess the feasibility of side-channel attacks by considering the specific execution context of a virtualized environment. Such an environment, multiplexing users on the same resource, exhibits characteristics specific to the virtualization technology. Hence, the side-channel exploits feasibility in a virtualized environment does not necessarily overlap with their feasibility in a native environment. Therefore, considering feasibility aspects in a virtualized environment is needed, as it can aid research on the virtualization isolation strength, and the actual countermeasures to mitigate specific side-channel attacks based on the execution environment and their assumptions.

Among the objectives of the thesis is to investigate the side-channel attack types and to determine the conditions under which these exploits are feasible considering the specific execution environment. For this purpose, we propose a generic classification to serve as a basis for the comparison and analysis of side-channel attacks (SCAs) with respect to diverse characteristics. An extensive classification approach comprising various attack aspects can facilitate the easier identification of mitigation paths for a specific context and of protection mechanisms which can be applied to decrease the probability of such an exploit. This extends the research on side-channel attacks classifications which typically does not focus on the execution environment impact and does not investigate under which conditions a specific attack can be conducted and is feasible.

The feasibility of a certain attack always depends on the adopted means,and we do not claim to provide absolute statements about attacks feasibility, but we aim to provide information regarding the conditions under which specific types of attacks are more or less probable. For this purpose, a selection of demonstrated side-channel exploits in a virtualized environment is considered.

3.2 SIDE-CHANNEL ATTACKS CLASSIFICATION APPROACH

We classify existing side-channel attacks to facilitate the assessment of the security-related properties of the execution environment in which these exploits can be conducted. For this, we define three main categories to characterize an attack according to the: (i) approach, (ii) effect and (iii) limitations. An overview on the proposed classification is given in Figure 4, and the subsequent paragraphs provide a detailed explanation regarding the different classes. In this section, we argue that attack's important characteristics are derived from the way it is conducted with a focus on the effect it has on the exploited system and its potential for success. However, the contextual limitations of the attack in terms of assumptions and challenges may not be neglected to truly excel in studying the details of the exploit.

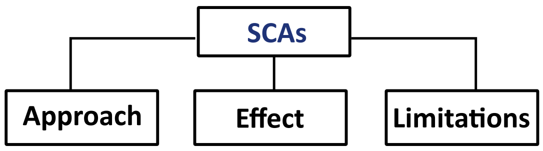


Figure 4: Overview of the Classification Approach.

3.2.1 *Approach*

The approach category describes the adversary strategy, i.e., the way the adversary compromises the security of a system by exploiting a side channel. We differentiate the side-channel attacks further depending on the leakage source or the shared medium being used for conducting the attack, on the intrusiveness of the attack, on the type of the collected and analyzed measurements and the method applied for analyzing the gathered data. This categorization is shown in Figure 5.

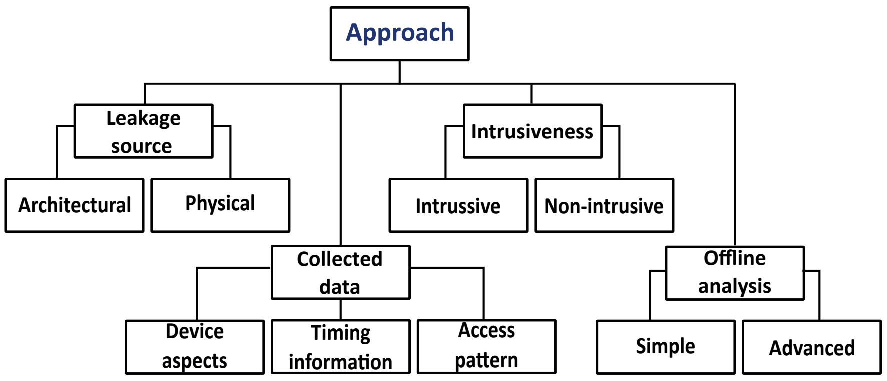


Figure 5: Detailed Overview of "Approach".

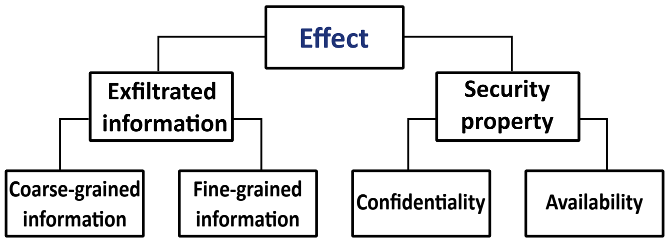


Figure 6: Detailed Overview of "Effect".

3.2.1.1 *Leakage source*

We further refine the classification by categorizing the side-channel attacks into physical attacks and architectural attacks in terms of the leakage medium or leakage source that has been leveraged for conducting the attack. The architectural side-channel attacks make use of an architectural system component. Such examples include the L1 cache, exploited as a side channel [Zha+12], the L2 cache [Xu+11], the L3 cache [YF14], virtual memory paging [Per05], etc. The physical side-channel attacks, in turn, exploit device components to conduct the attack, e.g., keyboard, monitor, power supply unit, etc. Examples of attacks that leverage the power supply unit are given in [Hla+11; MDS99].

3.2.1.2 *Collected data*

Depending on the type of the collected data, we distinguish between exploits leveraging physical device aspects, exploits measuring timing information and exploits relying on the characteristics of the access patterns to the side channel.

PHYSICAL DEVICE ASPECTS. This category encompasses attacks observing physical device aspects, such as power consumption, needed for the conduct of the attacks [Hla+11; MDS99], electromagnetic emanations, used for the attacks [Agr+03; Car+04], acoustic emanations, analyzed for the attacks in [GST14; SWT01]. These device characteristics are monitored and collected while the respective physical device performs a sensitive operation, for instance during cryptographic encryption.

TIMING INFORMATION. Timing information provides the basis for conducting attacks known as time-driven attacks. Sometimes, time-driven attacks have coarse granularity of the observations and are more successful if the time needed for the sensitive operation of attacker's interest (e.g., encryption process) is known in advance. To enable the statistical inference of information, the measurements might have to be conducted repeatedly. An attack relying on timing information is described in [Koc96].

ACCESS PATTERN. Information regarding the accesses to the side channel is leveraged to conduct access-driven attacks. A representative example for this kind of attacks are exploits leveraging the cache as a side channel, but the access patterns also to other shared architectural assets can be used to leak information. Very often this attack type involves collecting time measurements, but they are only used to deduce the access pattern to the observed shared resource. Depending on the specific attack, the involved timing measurements might not necessarily be of very high granularity. Examples of this category are the attacks described in [Per05; Ris+09; WXW12; Xu+11; YF14; Zha+12].

3.2.1.3 *Intrusiveness*

Another important characteristic of the side-channel exploits is their intrusiveness. We differentiate between intrusive and non-intrusive attacks.

INTRUSIVE. A prerequisite for the intrusive side-channel attacks is the direct access to the internal components of the observed device. Adversaries performing intrusive side-channel attacks intervene with device operation.

NON-INTRUSIVE. The non-intrusive side-channel attacks, on the contrary, are passive attacks and only monitor the device operation without intervening with it. Only externally available information, which has not been intentionally leaked, is leveraged in this case. Representative examples for this type of attacks are given in [Agr+03; GST14; Koc96; Ris+09; SWT01; YF14].

3.2.2 *Offline analysis*

The way the collected data is analyzed after the measurements have been conducted (i.e., offline), is used to further refine the proposed classification. We differentiate between advanced side-channel attacks and simple side-channel attacks, as already proposed in [ZD05].

ADVANCED. Commonly, the advanced side-channel attacks are characterized by offline analysis that involves a series of measurements, and sometimes the application of statistical methods to derive information. The offline analysis is also commonly dependent on the attacker capabilities. An example for this category is given in [Xu+11].

SIMPLE. For conducting simple side-channel attacks, it usually suffices to have a single trace to obtain the targeted information (e.g., to extract a secret key). Such an attack is usually easier to deploy in systems characterized by lower noise levels.

3.2.3 *Effect*

Another high-level classification of the side-channel exploits we propose is based on their effect on the system. For this, we focus on the security property that can be violated through the exploit and the asset under attack. To this end, we differentiate the side-channel exploits in terms of the exfiltrated information and the security property, as shown in Figure 6 and detailed in the following paragraphs.

3.2.3.1 *Exfiltrated information*

The exfiltrated information class refers to the assets under attack or to the granularity of the information an adversary is able to compromise. We focus on whether the adversary can gain fine-grained data such as a secret key or, coarse-grained information. For example, Ristenpart et al. managed to obtain coarse-grained data regarding activity spikes in a real Cloud scenario, as reported in their work [Ris+09]. Zhang et al, on the other hand, demonstrated a successful side-channel attack which managed to exfiltrate fine-grained information [Zha+12]. Through their attack Zhang et al. deduced an ElGamal secret key, as described in [Zha+12]. The border between fine-grained and coarse-grained information can be hard to establish. We consider fine-grained information data that enables direct access to confidential assets such as a secret key. Coarse-grained information, on the other hand, is data which can be used to conduct a subsequent attack. Such information is typically not very specific. Examples include probable virtual machine location, activity spikes of a specific machine, etc.

3.2.3.2 *Security property*

This category takes into consideration the security property being violated by the side-channel adversary.

CONFIDENTIALITY. Side-channel attacks primary goal is to compromise the confidentiality of a system. In this case the adversary exfiltrates information regarding the victim such as a secret key.

AVAILABILITY. Availability can be indirectly affected when conducting a typical side-channel attack which aims at leaking confidential information. The attack proposed by Zhang et al. and described in [Zha+12] is an example for that, as resources such as CPU time are frequently taken away from the victim to conduct the attack.

3.2.4 *Limitations*

The limitations class encompasses factors that might restrict the applicability of a side-channel attack to certain scenarios. Such factors include the assumption regarding the execution environment or other prerequisites of the attack which have to be fulfilled. Hereby, we classify side-channel attacks depending on the assumptions regarding the environment and the challenges that have to be considered when executing the attack. This classification can assist security experts in estimating the potential for success of a specific attack considering its details and specifity. The limitations class is consists of the assumptions and challenges subclasses. The following paragraphs elaborate more on the Limitations class, represented in more detail in Figure 7.

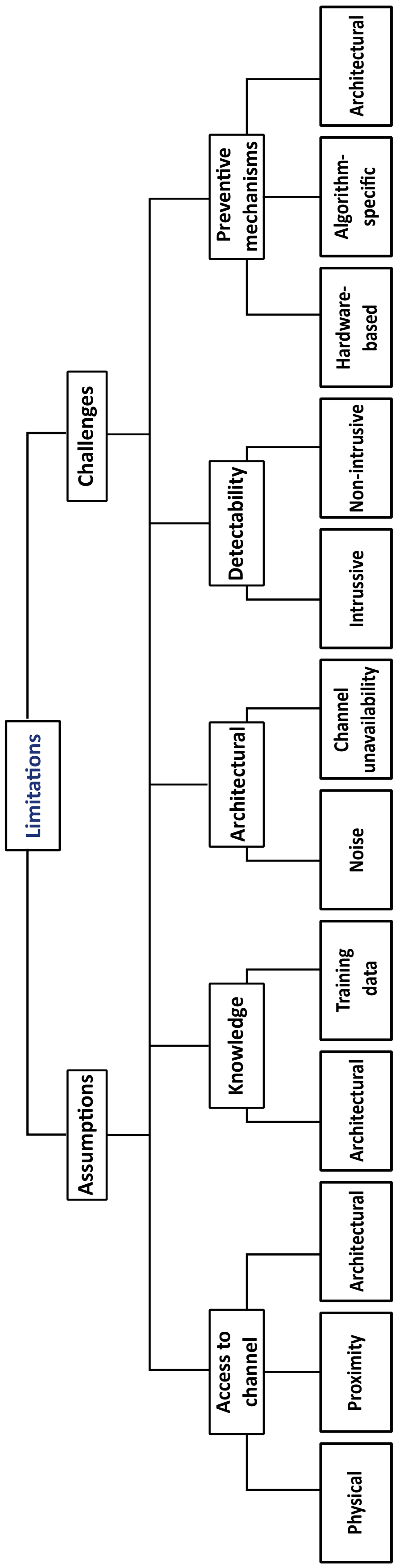


Figure 7: Detailed Overview of "Limitations".

3.2.4.1 *Assumptions*

Although many of the side-channel exploits are quite powerful and pose a serious threat to the confidentiality of the system under attack, they commonly proceed on certain assumptions regarding the underlying system or architecture. The classification of such assumptions can be beneficial for the better analysis of the attack and its applicability in a real-world or realistic scenario.

ACCESS LEVEL TO SHARED RESOURCES. A common precondition for conducting side-channel attacks is the usage or access to a shared resource between the victim and the attacker. This shared resource can be the power supply unit, the cache, etc. This shared medium should be present providing the link between the victim and the attacker. Depending on the access to this resource, we differentiate between the attacks relying on physical access to the shared medium, attacks requiring only proximity to the physical device and attacks requiring access to architectural components.

- Physical access - for conducting certain side-channel attacks having physical access to (parts of) the device hosting the victim is a prerequisite. This access enables measuring different physical aspects of the system which are needed for conducting the attack. A representative example for this category is the attack described in [MDS99] in which the power dissipated by a smart card is monitored. This takes place at the ground pin of the smart card and the attacker has to attach a resistor to the device to conduct the measurements.
- Proximity to the physical device - for conducting other side-channel attacks it suffices for the attacker to be in physical proximity to the device without accessing it directly. Examples for this category include attacks measuring the electromagnetic emanations as the one reported in [Agr+03]. A prerequisite for conducting this attack is to place probes as close as possible to the physical device in order to measure the induced emanations. Another example for this category is described in [GST14]. The attack reported by Shamir et al. in [GST14] requires placing a microphone near the physical device while performing cryptographic operations and recording the acoustic emanations.
- Remote - for conducting certain attacks it might be sufficient that the attacker has a remote access to the device, e.g., by accessing some architectural component. Such a scenario is given when a victim and an attacker use the same CPU for their computations in the Cloud. Examples for this category are presented in [Ris+09; Xu+11; YF14; Zha+12] and exploit the architectural access to shared components.

KNOWLEDGE. Knowledge about the system under attack is often a precondition for conducting a successful side-channel attack.

- Extended - to conduct a successful side-channel attack, the adversary often has to be acquainted with the system under attack and to be aware of its characteristics to take them into consideration when implementing the exploit. Additionally, the attacker might need a series of measurements including the availability of training data (e.g., as described in [Zha+12]). Shamir et al. also reported in [GST14] that previously gathered information is needed to map an acoustic pattern to the bits of the private key to successfully deploy a side-channel attack based on the recorded acoustic emanations of a computer. Although having training data is usually a natural step when conducting a side-channel exploit, acquiring such data might be challenging.

- Basic - attacks requiring basic knowledge, on the other hand, are typically easy to deploy without knowing complex cryptographic structures and details about the system architecture.

3.2.4.2 *Challenges*

Although successful side-channel exploits are reported quite often lately, the researchers commonly need to overcome challenges to deploy the attack. Depending on the involved obstacles, we differentiate between environmental challenges, employed preventive mechanisms or challenges related to the detectability of the exploit. The subsequent paragraphs detail the proposed categories.

ENVIRONMENTAL The environmental challenges are related to the execution environment. Such challenges can be associated with the architecture of the system and affect the channel. They are intrinsic and not intentionally created. We distinguish between challenges that affect the noise in the channel, and challenges that affect the availability of the channel.

- Noise - A variety of factors that affect the noise in the channel exist. The noise levels can vary depending on the scheduling policy, the interference with other processes for the shared resources, core migration in an SMP system when exploiting the core-private cache, etc. Such attacks have been described in [MKS12; Per05; Ris+09; WXW12; Xu+11; YF14; Zha+12]. The problem of having noise in the channel has been recognized in the community. Kocher, for example, defines the noise in the context of the attack proposed in [Koc96] as "timing variations due to unknown exponent bits". Examples for noise sources are given also in [GST14], including acoustic emanations stemming from other machines close to the microphone or, emanations stemming from the device under attack, but representing operations that are not of interest.

- Channel unavailability - another environmental challenge that the attackers exploiting side-channel attacks can face is related to the availability of the side channel. Mowery et al. report in their work [MKS12] that core pinning, which resulted in unavailability of the targeted side channel, is one of the reasons for the lack of success of their attack. In the attack presented in [GST14], the channel can become unavailable if, for example, the recording microphone is removed or gets broken. A core-private cache-based side channel is also not necessarily at disposal of an adversary residing on a distinct CPU core than the targeted the victim.

DETECTABILITY LEVEL. The detectability level characterizes the chances for an adversary to be detected. Some attacks might require preempting the victim frequently to conduct the needed measurements. Such an example is given in [Zha+12]. Keeping preemption rate low might be beneficial for the detectability of the attack and is classified into low and high.

PREVENTIVE MECHANISMS. Different preventive mechanisms can be applied and can affect the success of specific side-channel attacks. Commonly, the preventive measures are tailored for preventing particular attacks. Such measures can harden or even make specific attacks infeasible.

- Hardware-based - special hardware can be employed for securing the system. Examples of such hardware include tamper-resistant crypto modules, acoustic shielding, etc. A hardware-based mitigation mechanism characterizes the context of the exploit described in [MKS12]. Other potential preventive measures are detailed in [RWB04; Tir+05].

- Algorithm-specific - Side-channel attacks against the victim's confidentiality often target at compromising cryptographic keys and rely on the structure of the employed cryptographic algorithm to leak information. Unless vulnerable algorithms leaking information are employed, the attack will be infeasible. To this end, certain side-channel attack can be hardened if algorithm-specific preventive measures are in use. Such measures can affect the performance of the algorithm apart from enhancing system security.

- Architectural - Preventive mechanisms can be applied to protect from side-channel attacks exploiting specific architectural components, e.g., the cache. An example of such a mechanism is given in [KPMR12]. This work describes how to prevent cache-based side-channel exploits by managing a set of locked cache lines per core. These lines never get evicted from the cache. In such a way a virtual machine can hide the involved memory access patterns.

3.3 FEASIBILITY OF SCAS

The subsequent feasibility analysis focuses on the side-channel attacks exploiting the cache in a virtualized environment. We argue that the specific context can affect the success of the exploit by hardening it or by facilitating it. A L1 cache-based attack, for example, is harder to deploy in a multicore environment as the victim or the adversary might be frequently migrated among the CPU cores. On the other hand, if the simultaneous multithreading is enabled, it can lead to an easier deployment of exploiting the cache as a side channel.

The feasibility analysis is conducted with respect to the state-of-the-art works describing side-channel attacks exploiting the cache. The category "Challenges" from the presented classification in Section 3.2 is leveraged as a basis for the feasibility analysis. A set of feasibility factors that have impact on different cache-based side-channel attacks in a virtualized environment are detailed in the subsequent section.

3.3.1 *Feasibility factors*

The classification, proposed in Section 3.2 based on the challenges a side-channel adversary can face underlies the conducted feasibility analysis. We identify diverse factors which can turn a specific attack into a more or less feasible with respect to the given execution environment. Based on the context in which the side-channel are conducted, factors related to the environmental challenges, preventive mechanisms or the detectability of a potential exploit are derived and described below.

3.3.1.1 *Environmental challenges*

With respect to the environmental challenges, we differentiate between challenges with an impact on the noise in the channel and challenges with an impact on the channel availability.

NOISE. In this chapter we consider noise as measurements which the adversary collects through the cache, but these measurements result from data or instructions of no interest to the attacker. Possible noise sources are discussed below.

- Noise stemming from hardware features - examples for such hardware features are hardware prefetching or CPU power saving. Hardware prefetchers are designed and leveraged to increase system performance by speculating about future memory accesses. Modern hardware prefetchers are often complex and might be poorly documented, which makes filtering out the noise cause by prefetching a rather challenging task.

- Noise stemming from core migration - this is especially relevant for core-private cache side-channel attacks. The virtual CPUs of victim or adversary virtual machine might be floated among the available physical CPU cores. In such a scenario, the attacker is not necessarily aware of the core migration and might continue observing the cache by sampling only noisy data.

- Noise stemming from synchronization - typically, the side channel in a noise-free environment is alternately used by the victim and the attacker. Unless the victim and the attacker are properly synchronized, the measured noise in the channel might increase tremendously, as the attacker will acquire measurements that might result from attacker own cache accesses or might include only parts of the data the victim unintentionally encodes in the cache. A proper synchronization might turn out to be challenging and depends on the implementation of the attack and the capabilities and knowledge of the attacker.

- Noise stemming from the CPU scheduling - noise stemming from the scheduling has relation to the noise cause by the synchronization, but it depends more on the hypervisor's configuration and the used scheduling policies rather than on the knowledge and the capabilities of the attacker.

- Noise stemming from the interference with other virtual machines - in a virtual environment it is probable that third parties (e.g., other virtual machines) are accessing the same cache as the victim and the attacker. These third-party accesses cause noise in the channel and the adversary has to sort out which measurements stem from the victim and which measurements are caused by third parties. The levels of noise caused in this case depend highly on the number of involved co-located virtual machines and on the workload running on them. Filtering out such noise might be extremely challenging.

- Noise stemming from victim workload - the attacker might be interested in a part of victim's operations, but there is no guarantee that the acquired measurements are not related to other operations that the victim is conducting which are not of interest to the attacker. In such a case the noise in the channel will be increased.

CHANNEL AVAILABILITY. The cache-based side channel we consider, is available if the attacker can observe and measure victim's interactions with the cache at least for some limited time. Factors affecting channels availability are discussed below.

- Shared memory - specific side-channel attacks exploiting the cache are only possible if shared memory is available between the attacker and the victim. This is often given in systems using memory deduplication to optimize the memory footprint by deduplicating, e.g., shared libraries or the operating system or hypervisor. Examples of such attacks are reported in [Gru+16; YF14].

- Inclusive caches - most of the LLC attacks rely on the inclusiveness property of the caches. These attacks become infeasible if this property is not given. Such attacks are described in [Liu+15]. Intel proposed an architecture based on the Skylake-X processor in which the LLC is not inclusive and thus, not prone to this specific type of side-channel attacks. This, however, does not mean that modifications of these or other attacks are impossible, as shown in [Yan+19].

- CPU core - In the case of core-private side-channel attacks, channel availability is determined by the execution of the victim and attacker virtual machine on the same CPU core. Diverse factors can affect channel availability such as scheduling policies (including core pinning vs. load balancing; work-conserving vs. non work-conserving scheduling, etc.), the number of available CPU cores (e.g., multicore vs. single core architectures), as well as the frequency and type of allowed interrupts (e.g., inter-processor interrupts).

- Simultaneous multithreading - certain side-channel attacks rely explicitly on the simultaneous multithreading (SMT) feature. The processor resources, including the caches, are shared between threads if SMT is enabled. An attack which fits into this category is described in [Ald+19].

3.3.1.2 *Detectability level*

Assessing the detectability level requires a comprehensive analysis of each of the known attacks. It can be differentiated between detectability considering the hypervisor's perspective or the victim perspective. Detecting side-channel exploits is a hard task, but still some attacks leave certain traces. The frequency of victim's preemptions (preemption rate), for instance, can be leveraged as a hint for attacker existence although taking into account such traces is definitely not a trivial task.

3.3.1.3 *Preventive mechanisms*

When assessing the feasibility of a certain side-channel attack, we again distinguish between hardware-based preventive measures, algorithm-specific measures and architectural protection mechanisms, as defined in the classification approach.

HARDWARE-BASED. It should be considered whether the system employs special hardware as a mitigation strategy against the relevant type of attack. For example, tamper-resistant crypto modules can be employed to enhance the system security against a side-channel attack targeting at breaking the AES encryption.

ALGORITHM-SPECIFIC. Another important aspect which has to be taken into consideration is the vulnerability of the deployed cryptographic algorithms. If specific measures are applied to protect, for instance, the encryption algorithm from side-channel exploits - e.g., move AES instructions out of the cache or use algorithms that do not leak timing information, certain attacks might become infeasible.

ARCHITECTURAL. Additionally, it has to be taken into consideration whether certain architectural preventive measures are employed. It might be the case that the cache implementation provides some mechanisms for protection against side-channel attacks, for example frequent cache flushing or cache coloring techniques.

3.4 RELATED WORK

The ways of exploiting side channels to exfiltrate sensitive information and compromise data confidentiality is an actively researched area [Gru+16; KPMR12; LGR13; Ste+13]. Much effort has been devoted to propose a number of sophisticated side-channel attacks which are exploitable in virtualized environments and multi-tenant scenarios such as the Cloud [Hla+11; Ris+09; WXW12; Xu+11; Zha+12]. With this context, this section gives an overview on the existing side-channel attack classification approaches.

3.4.1 *Classification of Side-Channel Exploits*

There exists a number of classification proposals for side-channel attacks. One of the initial works is done by Kemmemer and published in [Kem02]. In [Kem02], Kemmemer investigates communicating through illegitimate channels not intended for communication. He proposes a methodology to increase the security assurance of a system by finding all possible information leakage covert channels through a Shared Resource Matrix. The proposed approach identifies and investigates all shared resources along with their attributes (rows of the matrix), as well as all system primitives e.g., Write-File (columns of the matrix). Criteria are proposed for identifying both storage and timing channels considering the constructed matrix. The bandwidth of the channel is also mentioned as an important characteristic, but Kemmemer's work focuses neither on the ways to calculate it, nor on an information theoretical analysis of the channels.

A variety of research papers focuses on the classification of side-channel attacks targeting cryptographic modules. Commonly, the approaches are divided into simple side-channel attacks (SSCA) and advanced or differential side-channel attacks depending on the analysis involved in the evaluation of the collected data, as described in [Bau+13; Cla+10; ZD05]. Clavier et al. further refines this categorization by distinguishing between horizontal and vertical side-channel analysis in [Cla+10] and considers whether a single power curve is analyzed, or the analysis is conducted in terms of different execution curves.

In [Bau+13], Bauer et al. describe their extensive study on side-channel analysis and propose a side-channel attack taxonomy comprising three classification categories. The first category distinguishes between simple and advanced side-channel attacks. The second category considers the leakage type. Information regarding whether an attack is profiled or not is included as a third category.

These classification approaches cover extensively certain attack aspects such as the methods for processing and analyzing the collected data, but are mainly focused on power and electromagnetic analysis attacks on cryptographic modules. Additionally they do not consider the execution environmental context which limits their applicability for the classification of side-channel attacks in virtualized environment with focus on attack feasibility.

A classification for attacks on cryptographic processors is proposed by Anderson et al. in [And+06]. This work does not explicitly focus on side-channel attacks, but most of the adversary scenarios that the authors give as examples for the classification fall into this category. In addition to the categories proposed by Anderson et al., the traditional classification of side-channel attacks existing in the literature divides them into the classes active and passive [ZD05]. A distinction between trace-driven, access-driven and time-driven side-channel attacks is proposed by Zhang et al. in [Zha+12], whereas the work described in [KPMR12] by Kim et al., considers only trace-driven and time-driven attack categories and further divides them into active and passive.

All the mentioned categorizations and taxonomies contribute to the side-channel attacks research field, however, they do not consider the execution environment characteristics or under which conditions a specific attack can be performed and the limitations therein. On this background, our work extends the state-of-the-art in the field by proposing a classification that is general enough to include the existing approaches and to address their limitations.

3.4.2 *Feasibility Assessment of Cache-Based Side-Channel Attacks.*

Mowery et al. express their doubts regarding the feasibility of AES cache timing attacks on the x86 architectures in their work [MKS12]. The authors conducted an unsuccessful attempt to compromise the confidentiality of a system based on the x86 architecture through the cache as a side channel. Therefore, they argue that due to the existing preventive measures they faced on the x86 architecture it is impossible to conduct their attack. Their experiments are based on a type of attack that targets at compromising the AES encryption and doubt the feasibility of this approach.

In [Xu+11], Xu et al. point out that depending on the achieved bit rate, the exploit of the side channel can be considered harmless, relying on information provided in [Dep85]. They also specify a number of factors that can have an impact on the side-channel bandwidth such as employed workloads in co-located virtual machines, hardware specification, hypervisor configuration, etc. As their conclusions are important, our work aims to continue their investigation by demonstrating that a variety of factors of the system impact the feasibility of the exploits of the cache as a side channel by modeling the execution environment.

Zhou et al. focus on providing a feasibility evaluation in regard to the side-channel exploits in [ZD05]. However, their evaluation includes rather generic aspects which are not concrete enough to take into consideration the characteristics of the execution environment.

3.5 CONCLUSION

Considering the threat posed by the numerous side-channel attacks requires a comprehensive analysis of the underlying system and an extended knowledge regarding the existing side-channel exploits. Although these exploits represent a serious security threat to virtually every system, their feasibility is affected by the execution environment. To reflect on this observation, this chapter proposes a generic side-channel attack classification approach, which includes the assumptions and challenges an attacker has to cope with when deploying a particular attack in a given execution environment context. This approach facilitates systematic reasoning about particular side-channel exploits and the assessment of their feasibility in a given context.

Such attacks pose a threat to the confidentiality within a virtualized environment which increases their relevance. The isolation between processes or virtual machines sharing the same physical resources is endangered due to the existence of side-channel and covert-channel attacks. Nonetheless, their exploitability also depends on the execution environement context. For instance, specific configuration of the hypervisor can enable a side-channel attack which is commonly not possible in other systems. Therefore, we show how to apply the proposed classification approach to enable the analysis of the feasibility of cache-based attacks in the context of a virtualized environment.

4

MODELING OF CACHE-BASED EXPLOITS

A side channel fundamentally deals with information, therefore, developing accurate information leakage models is of practical relevance, as such models can assist security experts when analyzing the robustness of their systems against side-channel exploits. Chapter 3 discussed the feasibility factors that affect the cache-based exploits, including the role of the scheduling. This chapter focuses on the development of an information leakage model that can explicitly consider the scheduling effect. The content of this chapter is, partly verbatim, based on material from [Vat+18] and addresses the second contribution of the thesis (cf., Section 1.3).

The chapter is organized as follows. Section 4.1 gives an overview on the tackled problem. Section 4.2 and Section 4.3 present the developed information leakage model, called InfoLeak along with a possible application scenario and an example on its usage. An overview on the related work is provided in Section 4.4. Section 4.5 concludes the chapter.

4.1 ON THE MODELING OF CACHE-BASED EXPLOITS

The area of information leakage modeling can assist security experts to analyze the chances of a side-channel exploit and to conduct feasibility analysis for a particular system. Much work has been devoted to investigating the area of formal modeling for side-channel attacks and a variety of formal models for covert-channel attacks [Gra93; Hun+15; Mil89] and side-channel attacks [KB07; TV05] exist in literature. Such models are primarily concerned with the way the information is leaked for a given attacker scenario and with the achievable bandwidth. However, the execution environment properties, including, e.g., availability of shared memory, scheduling, etc., often remain neglected by these models, although such properties influence the feasibility of the attacks. This impedes the systematic analysis on the cache-based exploits and the threat related to them with respect to a particular system.

Of special interest is the role of the scheduling, as a fundamental resource allocation schema and its impact on the core-private cache usage. The CPU scheduling approach affects directly the synchronization for the accesses to the core-private cache and its availability to the adversary as a side channel. It has been demonstrated already that proper synchronization is a prerequisite for a successful side-channel exploit, by having an effect on the feasibility and the bandwidth of the transmission channel, as shown in [Hu92a; VRS14]. Therefore, the lack of an information leakage model that explicitly considers the scheduling effect exacerbates performing a thorough system security analysis regarding the side-channel threat.

We develop an information leakage model, called InfoLeak. InfoLeak considers the effect of CPU scheduling on the feasibility of core-private cache exploits. The proposed model facilitates the systematic reasoning regarding potential side-channel threats and builds up the basis for using the scheduling information to conduct post-mortem analysis in regard to the side-channel threat. The CPU scheduling can cause lack of synchronization regarding the accesses to the side channel and make its exploitation impossible. Therefore, studying its influence is a key attribute behind the side-channel security analysis. As the direct integration of the scheduling schema into the analysis is impeded and not straightforward due to the involved non-determinism, InfoLeak enables security analysis by discerning the correlation between the CPU scheduling traces and the success of an exploit of the core-private cache as a side channel.

InfoLeak can assist to reveal the potential for security breaches related to the exploitation of the core-private cache as a side channel by using the CPU scheduling information. The model can be used as an indicator for the possible presence of an attacker exploiting the core-private cache without neglecting the noise induced by other processes in the system. A threshold can be leveraged as a feasibility measure for the amount of possibly transmitted or eavesdropped information, and can be used to raise an alert in case of a higher probability for an attacker. InfoLeak can be a beneficial enabler for conducting a post-mortem analysis of a system potentially under attack.

4.2 MODELING THE SIDE-CHANNEL WITH INFOLEAK

This section details our information leakage model, called InfoLeak. InfoLeak formally describes how the core-private cache is exploited as a side channel, and how its exploitation is influenced by the employed CPU scheduling. We adopt the well-established definition of a side channel, as a communication channel that results from the usage of shared resources and is not intended to be used for communication or for illegitimate data transmission. In the context of the thesis, the side channel can be used without special privileges. The following paragraphs define the side-channel users, their roles and their interaction with the channel.

4.2.1 *Channel Users*

InfoLeak defines channel users as system processes that interact with the core-private cache, and the core-private cache represents the channel. In this context, there exist three user categories, called the *sender*, the *receiver* and *other processes*, abbreviated respectively S, R and O. The core-private cache can be accessed by each of the user categories and therefore, each user can change its state. While R accesses the channel on purpose to monitor and analyze the information available in the channel, S sends information through its accesses to the cache. R plays the role of the receiver in a covert-channel attack or the adversary in a side-channel attack. The role of S might be taken over by the sender in a covert-channel attack or by the victim in a side-channel attack. In the first case the actions of S are intentional whereas in the case of a side-channel attack S sends data unintentionally. O uses the channel, but is not necessarily aware of its usage for illegal data transmission and encodes only noise into it. Similarly, R also encodes noise into the channel through its actions to analyze the sent data.

4.2.2 *Channel States*

Within InfoLeak, the channel state is defined through the cache footprint the user leave or the user cache access pattern. Through their operations, i.e., their cache accesses, the users change deliberately or non-deliberately the channel state. Therefore, we distinguish between the cache states O_i, which refer to observation$_i$, and a state U for undefined. The states O_i stand for to cache access patterns which correspond to meaningful information to R from the perspective of a successful data transmission through the channel. U is basically the channel state that result from noise perturbations. Noise perturbations occur if O, for example, has overwritten the information S had encoded into the channel. before R has been able to decode it. This example denotes a case where S and R are not synchronized properly, possibly as a result of the CPU scheduling policy.

The channel states that are involved in a covert-channel attack, for instance, can be characterized by two predefined cache footprints to represent the bits 0 and 1. These cache footprints that stand for 0 and 1 are then mapped to the states O_1 and O_2 of InfoLeak. The undefined state U is also a part of the model by representing the noisy state in this example.

4.2.3 *Channel Interactions*

To model the interplay of S, R and O with the channel, we employ a non-deterministic finite automaton (NFA) A. The channel states are represented as states in A while the user interactions with the channel trigger transitions in A. These is formally described below.

$A = (Q, \Sigma, \Delta, q_0, F)$ is comprised of:

- a finite set of states $Q = Ob \cup \{U\}$, where $Ob = \cup_{N}^{i=1}\{O_i\}$ for $N \in \mathbb{N}$;
- a finite set of input symbols $\Sigma = \{s, r, o\}$;
- an initial state $q_0 = U$;
- a set of accepted states $F = \{U\}$;
- a transition function $\Delta : (Q \times \Sigma) \to 2^Q$, defined in Table 1.

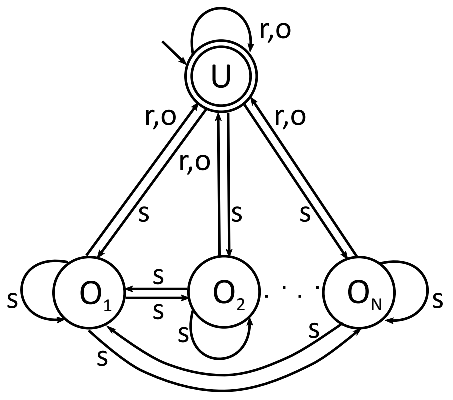
The set of input symbols Σ defines the labels for InfoLeak transitions. Each symbol is chosen in correspondence to the user triggering the transition. More precisely, S, R and O trigger transitions s, r, and o, respectively. The assumption behind InfoLeak is that S always encodes meaningful information into the cache. Therefore, each transition, that is triggered by S always changes the state of the

Table 1: Transition Function of NFA A.

		Input Symbol		
		s	r	o
State	O_i	Ob	$\{U\}$	$\{U\}$
	U	Ob	$\{U\}$	$\{U\}$

channel to O_i, as shown in the NFA in Figure 8. These transitions are denoted with label s and we refer to them as the s-transitions. As an example, in a covert-channel attack S sends one bit of information to R at a time by leaving a distinguishable cache footprint which stands for bit 0 or bit 1.

Figure 8 illustrates also the channel state changes caused by R referred to as r-transitions. These transitions reflect the objective of R to decode the encoded data. In case of a proper synchronization between S and R, R induces a transition between the states O_i and U. This transition denotes the situation when R receives information and produces noise through its decoding operations. The state U is, therefore, a part of the accepted states F, as the operation of receiving information after the r-transition is successful. Since R always produces noise in the channel, the r-transitions always lead to the undefined state U, as depicted in the NFA in Figure 8.



1	$r, o \rightarrow ro$
	$r, r \rightarrow rr$
	$r, Z \rightarrow rZ$
	$o, o \rightarrow oo$
	$o, r \rightarrow or$
	$o, Z \rightarrow oZ$
2	$\mathbf{r, s} \rightarrow \varepsilon$
	$o, s \rightarrow os$
3	$s, o \rightarrow so$
	$s, r \rightarrow sr$
	$s, Z \rightarrow sZ$
4	$s, s \rightarrow so$

Figure 8: NFA A (left) and P's Stack (right).

It is noticeable that the transitions triggered by O (cf, Figure 8), called o-transitions, are the same as the ones triggered by R. Both R and O interactions with the channel result in noise perturbations and thus, their transitions lead to state U. Nonetheless, for the further analysis, we distinguish between O and R due to R's intentional usage of the channel in contrast to O's unintentional channel interaction.

For brevity and to avoid complicating the model unnecessarily, the cache is assumed to be in the undefined state U before the attack. Hence, U is the initial state of A.

4.3 MODELING THE SCHEDULING EFFECT

the synchronization between S and R is a prerequisite for the success of the core-private cache exploits. As this synchronization is mostly determined by the CPU scheduling, we model the scheduling effect by extending the NFA A to a non-deterministic pushdown automaton (PDA) with a stack.

As there are variations regarding the PDA terminology, we specify the notion used in this section. In InfoLeak, ε is used to describe the situation when the stack is popped. In this case, the element at the top of the stack is popped and no new element is pushed onto the stack. In InfoLeak , an input symbol is read with each move. With each transition, the element at the stack top is read and popped and a string that consists of the popped element and a new element (or ε if the stack is popped) is pushed onto the stack. Then, the new element is the new stack top element.

To model the scheduling effect on the core-private cache exploits, we use PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$. Therefore, we define an initial stack symbol, a stack alphabet and a new transition function, as follows.

- a stack alphabet $\Gamma = \{Z\} \cup \{s, r, o\}$;
- a transition function $\delta : (Q \times \Sigma \times \Gamma_\varepsilon) \to 2^{Q \times \Gamma_\varepsilon}$, detailed in Table 2;
- an initial stack symbol Z;

Table 2: Transition Function of PDA P.

Trans.#	Transition
1	$\delta(U, s, Z) = \cup_N^{i=1} \{(O_i, sZ)\}$
2	$\delta(O_i, s, s) = \cup_N^{i=1} \{(O_i, so)\}$
3	$\delta(U, s, r) = \cup_N^{i=1} \{(O_i, sr)\}$
4	$\delta(U, s, o) = \cup_N^{i=1} \{(O_i, so)\}$
5	$\delta(U, r, Z) = \{(U, rZ)\}$
6	$\boldsymbol{\delta(O_i, r, s) = \{(U, \varepsilon)\}}$
7	$\delta(U, r, r) = \{(U, rr)\}$
8	$\delta(U, r, o) = \{(U, ro)\}$
9	$\delta(U, o, Z) = \{(U, oZ)\}$
10	$\delta(O_i, o, s) = \{(U, os)\}$
11	$\delta(U, o, r) = \{(U, or)\}$
12	$\delta(U, o, o) = \{(U, oo)\}$

The finite set of states Q remains the same as defined in A and is detailed in Section 4.2.2.

The transition function of P, defined in Table 2, takes as input: (i) a state; (ii) an input symbol (excluding ε-transitions); and (iii) a stack symbol (which can be ε and denotes the element on the top of the stack). The input tripple is then mapped to a set of states and a set of stack symbols where the first element denotes the stack top element. For example, s in sZ is the top stack element. Figure 8 illustrates the changes in the states along with the respective stack operations.

The following paragraphs describe the transitions presented in the table on the right side in Figure 8 in more detail. The first shown transition, $r, o \rightarrow ro$, indicate that r is the input element and o is the current topmost stack element. With this transition, the new topmost element in the stack becomes r while the state U remains unchanged, as shown in the NFA on the left side in Figure 8. Table 2 also defines this behavior (cf., Transition 8).

The successful transmission of information is given by the transition $r, s \rightarrow \varepsilon$ (cf., first row of Transition 2 in Figure 8 or Transition 6 in Table 2). Similarly to the NFA A, a transition from O_i to U takes place. With this transition, as R interacts with the channel directly after S, the stack is popped. The symbol s is read and popped and no new element is pushed onto the stack.

In the case of two subsequent s, as described in Transition 2 Table 2 or if an o follows an s, as shown in Transition 10 in Table 2, the sent information is overwritten. This happens, as the cache footprint is changed before the receiver has managed to receive and analyze it. To reflect this issue, if an s is the topmost element of the stack, and another s comes, the first s is popped and substituted by an so. This behavior is defined by Transition 2 in Table 2.

The states proposed in InfoLeak reflect on the view on the core-private cache from an observer's perspective. As the objective of InfoLeak is to incorporate the order of accesses to the channel into the model, a change in the channel state is triggered with each user interaction. InfoLeak abstracts from defining specific cache footprints and mapping them to the channel states. This allow for encoding information of different granularity through the definition of the abstract states O_i. The encoded information is primarily in the form of cache footprints, but InfoLeak abstracts from details such as specific footprints. This makes the model applicable to both side-channel and covert-channel scenarios. For instance, it can be generalized to include cases where symbols instead of single bits are transmitted in a covert-channel attack, e.g., by adding a new state O_i in the NFA for each additional symbol. In such cases, the undefined state U remains unchanged.

By using InfoLeak, an estimate of the achieved covert-channel attack bandwidth can be inferred. For this purpose, we need to keep track of the number of successive times the stack has been popped over a predefined time period. A stack of a large size, indicates that either there have been too many other processes interacting with the cache over the monitored period of time or that S and R have not been synchronized properly. An empty stack suggests the successful receipt of the transmitted data and no contention in the channel with other processes. A threshold can be defined for the number of successive times the stack has been popped and can be used as a measure for the feasibility of the covert-channel transmission.

4.3.1 *Discussion and Possible Extensions*

In our model we consider the side channel as a discrete memoryless channel similar to [MM94; WL05], which has an impact on the proposed model. As a result of this consideration, InfoLeak models the channel as a medium where each send, receive or other operation interacting with the core-private cache erases the previously stored data. Hence, in InfoLeak, older states do not influence the capacity of the channel. This assumption is valid, as the core-private caches are relatively small. To confirm that, we conducted experiments considering successive scheduling of sender and receiver in correlation to the success of the exploit of the core-private cache (cf., Part III).

Table 3: PDA P Transition Function – Extended Model.

Trans.#	Transition
6.1	$\delta(O\_i,\ r,\ s_n) = \{(U, \varepsilon)\}$
13	$\delta(O\_i,\ o,\ s) = \{(O\_i, s_n)\}$
14	$\delta(O\_i,\ o,\ s_n) = \{(U, os_n)\}$
15	$\delta(O\_i,\ s,\ s_n) = \{(O\_i, so)\}$

Apart from that, InfoLeak is an extensible model which can also address channels with memory. For this purpose, we have to define the length of the history which should be considered. On an example we consider a history of one operation, and add an additional symbol s_n (n stands for noise) to the stack alphabet (i.e., $\Gamma = \{Z\} \cup \{s,\ s_n,\ r,\ o\}$). This denotes a sending operation perturbed by noise. The additional symbol models the scenario in which after an S, another process O causing noise has interacted with the channel, but the sent bit can still be decoded if R interacts with the core-private cache directly after O. To address such scenarios, the transitions given in Table 2 need to be extended as demonstrated in Table 3.

Additionally, it has to be noted that InfoLeak models the core-private cache as a transmission medium, but do not incorporate the possible adversary actions once the transmitted data has been obtained, e.g., employing error correction or detection mechanisms.

4.3.2 *On the Application of InfoLeak*

We apply InfoLeak to a synthetic example, which resembles a well-know side-channel attack. Additionally, we discuss on how InfoLeak can be leveraged for a post-mortem analysis to investigate covert-channel communication's feasibility when using the core-private cache.

4.3.2.1 *InfoLeak Utility Example*

As an example, we simulate the side-channel attack proposed by Zhang et al. in [Zha+12] and model the leakage with InfoLeak. Zhang et al. [Zha+12] demonstrate a side-channel attack in a virtualized environment, in which the adversary abuses the scheduler to interrupt the victim frequently enough. The adversary exploits the core-private cache and therefore preempts the victim to be able to collect sufficient number of observations on victim's cache accesses.

In the described attack, the adversary monitors victim's cache footprint and creates vectors of 64 values each. These vectors represent the access times to the 64 cache sets. The adversary maps the collected timing values to the Square (Sq), Multiply (M) or ModReduce (MR) operations of the square-and-multiply algorithm used for fast exponentiation. Based on these observations and further analysis, the adversary can extract victim's confidential information (i.e., the secret key). By modeling the described attack using InfoLeak, we consider the Sq, M and MR as the states O_i in the NFA P, as described in Section 4.2. The cache access pattern or cache footprint that represents a squaring operation is mapped to the state Sq. Analogously, the states M and MR represent a multiply and modular reduction operations, respectively. Figure 9 shows the resulting NFA.

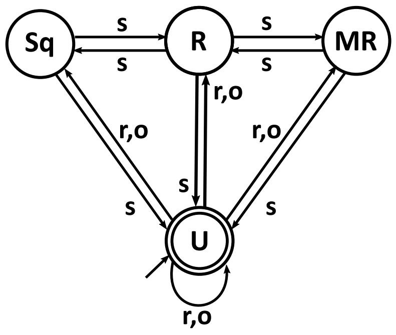


Figure 9: InfoLeak Utility Example.

In the presented example, two successive observations on the channel reveal one secret key bit, in the case of bit 0. On the other hand, four successive observations directly after the victim are needed to extract a bit 1. Therefore, in case of a uniform distribution of the secret key bits, the adversary will need on average three successive observations to extract a single bit. In InfoLeak this results in a sequence s-r-s-r-s-r or the stack is popped three successive times.

The scheduling information can be analyzed for the successive scheduling with respect to any pair of processes. The frequent interrupts caused by the adversary, in the case of the described attack, are visible in the scheduling traces and can be leveraged as an indication that an attack has been feasible or not. It is important to note that even few bits might lead to extracting a private key if the side-channel attack is combined with a consequent attack. Nonetheless, it would be more beneficial for an adversary if the extracted bits are successive. This observation can be used to adjust a feasibility threshold in terms of possibly leaked successive bits of information.

4.3.2.2 *Post-Analysis of Scheduling Information*

By leveraging InfoLeak's idea, we can analyze the scheduling information, e.g., the logs provided by the operating system, and investigate them for suspicious covert communication. If we identify suspicious processes, they can be represented as S and R, whereas the rest of the processes are represented as O's. Then, we can build up the transitions of InfoLeak using the scheduling log and can build a stack of s- r- and o-transitions. By taking into account the number of successively popped s's over a predefined period time, we can give an estimate of the feasibility of information leakage happening. Although the information provided when applying InfoLeak to the system logs is valuable, InfoLeak's applicability is only feasible if suspicious processes have already been identified. Chapter 5 investigates further this problem.

4.4 RELATED WORK

This section reviews the area of information leakage modeling in the context of cache-based exploits.

4.4.1 *Information Leakage Modeling*

Information leakage through covert channels has been considered initially by Lampson in [Lam73]. In his work, Lampson characterizes informally information leakage possibilities for a confined program. His work [Lam73] lists a number of possible leakage sources and classifies the leakage channels as storage-based, legitimate and covert. In [Lam73] Lampson derives also confinement rules to prevent information leakage, e.g., by applying masking and enforcement to block covert channels. Despite the importance of this work from information theoretical perspective, the proposed confinement rules are restrictive to apply in practice nowadays.

The communication through illegitimate channels is also explored by Kemmemer [Kem02]. Kemmemer proposes the Shared Resource Matrix as a system to find all possible information leakage covert channels. The proposed approach models all shared resources along with their attributes and with criteria to identify all the possible storage and timing channels through the resource matrix. Given the large number of the possible leakage sources, the proposed approach, though valuable, is not efficient to be applied in practise.

In his work [Wra91], Wray considers a covert channel as a channel with storage and timing attributes. Based on that, a timing channel is defined as a channel in which information is conveyed by the relative timings of two clocks or event sequences visible by the observer. For this purpose, Wray defines a clock as being characterized by sequences of events which can be distinguished by one another. The work described in [Wra91] proposes also a way to identify covert channels through the construction of a matrix. The matrix' rows list clocks modulated in the channel exploitation and its columns list clocks used as reference clocks in the exploitation. Such an approach, despite being essential from theoretical point of view, requires the exhaustive enumeration of all clocks or shared resources to detect all the possible covert channels, which is not practically feasible.

Another important work considering covert-channel information leakage is presented by Hu in [Hu92b]. Hu proposed a method for decreasing the capacity of the covert-channel tremendously by using fuzzy time system clocks. This approach disables proper synchronization between the communicating parties to affect the capacity of the information leakage channel. Hu demonstrates experimentally that the capacity of the channel is reduced, but does not provide any information-theoretic capacity analysis of the bus contention-channel used under fuzzy time.

In [Gra91], Gray focuses on modeling system and environmental probabilistic behavior independently and expressing channel capacity in information theoretical terms. It considers a system as a finite set of states and a set of communication channels providing only interface to the external environment. Gray does not consider explicitly the scheduling role in [Gra91], but the distinction between system under consideration and the environment in which the system is executing is very important for the covert-channel and side-channel analysis.

A mathematical model for microarchitectural information channels, called the Bucket model is described in [Hun+15]. The Bucket model considers a third party eavesdropping the communication in the covert channel and in such a scenario, the model facilitates the detection of intelligent attackers. However, the represented attacker model is different than the commonly used one. Our work considers the well-established side-channel and covert-channel attacker models.

4.5 CONCLUSION

To reflect on the investigated feasibility aspects, we constructed an information leakage model, called InfoLeak, which accounts for the scheduling effect on the core-private cache exploitability in a side-channel or a covert-channel attack. Through the model, we elaborate on how the exploits of the core-private cache can be impaired by the CPU scheduling of the involved sender and the receiver processes, and integrated the impact of the CPU scheduling into the analysis of the cache-based exploits. Moreover, the proposed information leakage model can assist in investigating a system with regard to a possible information leakage through the core-private cache post-mortem. The proposed information leakage model beg the question of assessing the exploitability of the caches systematically by considering the context of their execution environment.

Part III

EMPIRICAL ANALYSIS OF THE EFFECT OF THE EXECUTION ENVIRONMENT ON THE SCAS

5

COVERT-CHANNEL ATTACKS AND THEIR TRACES

Years of profound research activities and intense work mark the area of side-channel attacks. Part II discussed the role of the CPU scheduling when conducting covert-channel attacks exploiting the core-private cache. In this chapter, we further investigate the scheduling effect by conducting an empirical study in a native system and derive potential indicators for the existence of a side-channel adversary. The contributions presented in this chapter are, partly verbatim, based on the material presented in [Vat+18] and [VG+19].

This chapter is structured as follows. Section 5.1 gives an overview on the problem. Section 5.2 focuses on a set of possible attack indicators in the context of core-private cache exploits. Section 5.3 presents the experiments and the obtained results, and discusses their implications. Section 5.4 gives an overview on the related work, and Section 5.5 concludes the chapter.

5.1 ON THE INDICATORS OF CACHE-BASED EXPLOITS

It has been demonstrated that properties of the execution environment can affect the side-channel exploits and can prevent certain attacks [Zha+12]. With our focus in this part on core-private caches, among these execution environment properties is the CPU scheduling. The CPU scheduling can allow for fine-grained observations on victim's cache usage [Hu92a; VRS14; Zha+12] or can make observing the cache at the right time impossible for the attacker. Certain side-channel attacks even rely on abusing the scheduler as a prerequisite for a successful side-channel exploit. Inter-processor interrupts (IPIs) can be used to achieve frequent victim preemptions. For instance, Zhang et al. leverage IPIs in their attack [Zha+12] to be able to collect fine-grained data on victim's cache accesses.

On this background, we explore the correlation between the success of the core-private cache exploits and the scheduling of the attacker directly after the victim, referred to as successive scheduling within this chapter. As analyzing the effect of the scheduling algorithms on the side-channel attacks directly turns out to be less feasible due to the involved non-determinism, we study the available scheduling traces. Thereby, we try to obtain information regarding traces of a potential confidentiality compromise. While contemporary research often considers side-channel attacks relying on shared memory [Ira+14; YF14], their exploitability is partly limited, as it depends on the usage of features such as memory deduplication which can be disabled by the providers, e.g., [VMwa; VMwb]. Therefore, in this chapter, we focus on Prime+Probe attacks exploiting the core-private cache.

Our proposal is to monitor three scheduling-related metrics, called attack indicators, to try to infer information on the feasibility of side-channel or a covert-channel exploit in the given execution environment. Among the proposed attack indicators are the frequency of inter-processor interrupts, the busy waitings issued by a process and the number of successive scheduling cases of two processes on the same CPU core. We suggest the usage of scheduling traces for post-mortem analysis or feasibility assessment of side-channel or covert-channel attacks exploiting the core-private cache. Our experimental analysis is based on a case study in which we consider the success and feasibility of a L1 covert-channel attack.

To this end, the contributions of this chapter encompass: (i) the definition of potential indicators for cache-based attacks, based on IPIs, busy waiting and successive scheduling and (ii) the assessment of the correlation between the proposed indicators and the success of a core-private cache-based covert-channel attack in a case study.

Our empirical results demonstrate that the proposed indicators represent a step towards establishing the link between the execution environment and the success of a side-channel or a covert-channel exploit. This amplifies the expectation that the characteristics of the environment play a crucial role in the feasibility of these attacks using the core-private cache and can be leveraged to derive indicators of compromise for the cache-based exploits.

5.2 CPU SCHEDULING AND ATTACK TRACES

This section discusses the possible traces an attacker exploiting the core-private cache might leave on the system and the role the CPU scheduling plays.

5.2.1 *Scheduling-Related Considerations*

The investigated attack indicators or attack traces are related to the scheduling of the attacker process(es). Hence, the next paragraphs discuss the scheduling issues relevant for the attack deployment and detail what can be taken into consideration when investigating attack indicators.

5.2.1.1 *Abusing the Scheduling Approach*

In [Zha+12], Zhang et al. describe a side-channel attack in a Cloud scenario that exploits the core-private cache. For the success of the described attack, the authors leverage an additional third-party virtual machine that issues IPIs, to facilitate the frequent observations of the attacker on the core-private cache. The goal is that the attacker can be scheduled frequently enough directly after the victim, and can collect fine-grained observations over victim's cache usage. Such an approach enables the usage of the core-private cache as a side channel, but is, however, only possible if the third-party process can preempt the process of the victim frequently enough. To this end, non-preemptive scheduling approaches appear to pose a hurdle on conducting core-private cache attacks across virtual machines, as they cannot be abused through frequent interrupts. However, due to their possible performance implications, these approaches are almost obsolete in user space of the modern operating systems today. Furthermore, the attacker process can try to influence the choice of CPU core it is scheduled to run on by adjusting its priority (e.g., by trying to set the scheduling affinity) or the scheduling scheme [Ker13]. Such features are beneficial from a performance and usability point of view, but they can increase the risk of abusing the scheduler.

5.2.1.2 *Operations Atomicity and Synchronization.*

Under synchronization we refer to the ability of the attacker to observe victim's usage of the core-private cache directly after it has taken place for the required period of time. As already discussed in Chapter 4, a proper synchronization between the attacker and the victim is crucial for the core-private cache exploits. In the context of a covert-channel attack exploiting the core-private cache, the receiver has to be scheduled directly after the sender. In addition, in such an attack the atomicity of operations has to be retained. This means that the receiver has to be scheduled after the sender has completed its sending operation. If the sender is interrupted in the middle of the sending process, the receiver will not be able to decode the original data. The same applies if the sender leverages the core-private cache for a long time while performing more than one sending operations. In such a case, the receiver is not able to gather all the transmitted data, as they have been overwritten by newer data. For optimal attack results, the scheduling of the attacker should consider the execution times of the involved sending and receiving operations.

5.2.2 *Attack Indicators Selection*

Considering he scheduling impact on the success and feasibility of core-private cache-based side-channel exploits, we propose to investigate the traces an attacker leaves on the system related to scheduling. For this purpose, we identify three possible attack indicators which are briefly discussed below.

5.2.2.1 *Successive scheduling*

The successive scheduling (SS) refers to the number of times the receiver process has been scheduled to run on the CPU directly after the sender process and determines the synchronization over the usage of the core-private cache. SS of an adversary process and a victim process or of sender and receiver processes can facilitate the exploits of the core-private cache as a side channel. Therefore, we apply it as a possible indicator for such an exploit. To monitor the SS, a process or a pair of processes with suspicious behavior should be identified first to observe and analyze the SS cases with respect to the suspect processes.

5.2.2.2 *Busy waiting*

In attacks exploiting the cache as a side channel, the already mentioned synchronization is often done through the adversary by explicitly yielding the CPU to let the process of interest run, e.g., victim or sender process. Commonly, after each measurement, i.e., after performing a complete operation, the adversary process yields the CPU and ideally, while the attacker is waiting, the victim is scheduled to run. We suggest to explore the frequency of a process voluntary "releasing" the CPU, referred to as busy waiting (BS) for the rest of the chapter. BW can be used not only as a general attack indicator, but also to identify the processes with suspicious behavior for which the SS indicator should be investigated by considering processes characterized by an increased number of BWs.

5.2.2.3 *Inter-processor interrupts*

As already mentioned, through IPIs the adversary can abuse the scheduling approach to grant frequent access to the core-private cache across virtual machines. Hence, the number of IPIs over a predefined time span can be considered and monitored as a potential attack indicator.

5.3 EXPERIMENTAL SETUP AND RESULTS

To empirically validate the utility of the proposed attack indicators we study the correlation between the successive scheduling of two adversary processes in the context of a covert-channel attack through the core-private cache and the attack success. Furthermore, we consider the busy waiting triggered by the attacker processes in both adversarial scenario and in a non-attacked system. We basically perform post-mortem analysis of the logs provided by the operating system that contain scheduling related events and system calls. This section details the setup for the experiments, the implementation details, and concludes with a presentation and discussion of the results.

5.3.1 *Setup*

We conducted two sets of experiments considering SS and BW separately on a Debian Stretch in both noiseless and noisy environments. The experiments include executing a covert-channel attack 100 times to transmit messages from the sender to the receiver. Simultaneously with the execution of the attack, we logged the CPU scheduling information. For this purpose, we make use of the tracing options in Debian through */sys/kernel/debug/tracing* by enabling the tracing of the *context switch* event. The collected logs are parsed, and are correlated to the attack's success. For the experiments, we leveraged a system, as the one depicted in Figure 10.

5.3.2 *Implementation Details*

For both the SS and BW the experiments, we employed covert-channel attacks that adhere to the Prime+Probe attack strategy, described in [Per05]. Each covert-channel attack consists of a sender and a receiver adversary processes which communicate through indirectly accessing the core-private L1 cache. The two adversary processes have agreed on the meaning of cache footprints to represent

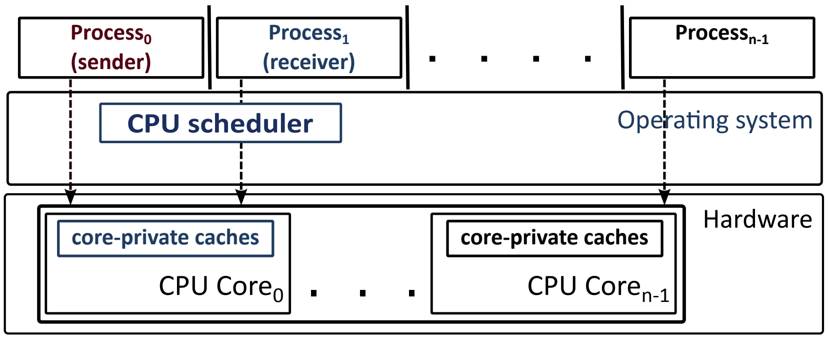


Figure 10: System Model Underlying the Assessment of the Traces of Core-Private Cache Exploits.

bits 0 and 1 and each of them allocates memory of the size of the L1 cache (32KB) to encode and decode the predefined cache footprints.

In our implementations, the sender encodes a single bit of information at a time (per sending operation) and, to do that the sender process accesses deliberately chosen parts of the allocated memory. The data accessed by the sender is copied into the cache sets representing either bit 0 or bit 1. Thereby, the sender evicts receiver data that had been stored in these parts of the cache, and fetches other data from the main memory or from the higher cache levels to the core-private cache. Analogously to the sender, the receiver accesses the cache through its allocated memory and to receive the message, the receiver process measures the time for accessing specific parts of the cache. Depending on the access times, the receiver can determine which cache parts have been evicted by the sender, and derives the cache footprint the sender has left. Then, the cache footprint is mapped to either bit 0 or bit 1 or no bit can be due to noise interference.

As a promising time reference, we sample the timestamp counter hardware register TSC to obtain the number of the elapsed clock cycles during an operation.

However, obtaining precise time measurements comes along with certain challenges. For instance, we have to ensure that by the moment the receiver samples TSC, the instructions that come before the sampling instruction (RDTSC), have already been executed. Additionally, the time has to be sampled before the execution of the instructions that come after the TSC sampling instruction in the source code of the receiver process. To ensure that the compiler and the processor respect the original order of instructions with respect to the TSC instruction, we make use of volatile as a memory barrier and the CPUID instruction for serialization.

Additionally, conducting a covert-channel attack is complicated through the distinction between physical and virtual memory addresses, as described in [OST06]. Our adversary processes do not have privileged access rights and have a view on the virtual addresses of their allocated memory blocks only. This complicates accessing predefined cache parts, but as we use the L1 cache, it is still possible to access specific cache sets by conducting operations on deliberately chosen parts of the allocated memory blocks.

In our setup the measurements are done in a C application, leveraging inline assembly instructions, and the decoding is conducted offline by scripts once the access times have been collected, so that operations of the receiver do not interfere with or invalidate the measurements, as the L1 cache is very sensible to noise.

5.3.3 *Successive Scheduling Experiments and Results*

For these experiments, the sender sends messages of length 1200 bits in different setups with varied levels of noise to assess the correlation of the successfully transmitted bits with the successive scheduling between sender and receiver. The length of the messages and the number of experiments are chosen to ensure statistically significant coverage and therefore represent 120000 sent bits. Within one experiment, the receiver process is executed 1200 times to receive the sent data and the sender performs 1200 sending operations. Simultaneously with the conduct of the attack, we log the context switch events provided by the tracing options in Debian through */sys/kernel/debug/tracing*, and study the log for successive scheduling cases with respect to the sender and the receiver processes.

To assess the correlation between the number of successfully transmitted bits and the successive scheduling between the sender and the receiver statistically, we employ Pearson's correlation coefficients. For this purpose, we use the successive scheduling, i.e., the number of times the receiver is scheduled directly after the sender and the number of successfully transmitted bits (ST) over the channel per experiment, respectively. We deploy the experiments in three scenarios with varied noise levels.

5.3.3.1 *Noiseless environment.*

In the noiseless setup, we conducted 100 experiments whereas 11 bits on average per experiment (out of 1200 bits sent) have not been properly decoded. The mean value for ST in the noiseless setup is 1189 with a standard deviation of 4.151. The mean value for SS within the same experiments is 1199 with a standard deviation of 0.89. Table 4 summarizes the results, and Figure 11 visualizes the relationship between the number of successive scheduling cases and the number of successfully transmitted bits. The x-axis depicts the experiment number and the y-axis depicts both the number of successfully transmitted bits per experiment (cf., blue plot) and the number of successive scheduling cases per experiment (cf., purple plot), whereas each point in the depicted linechart corresponds to one experiment.

Due to the low noise level in the transmission channel and the lack of contention with other processes for the channel, in the conducted experiments the receiver has been scheduled directly after the sender in almost all of the possible cases. The number of successfully transmitted bits is also high mainly due to the lack of contention for the channel. There is a small discrepancy in the values shown in the two linecharts in Figure 11 in which the sent bits were not successfully transmitted despite beneficial scheduling. This can be due to a violation in the atomicity of the sending and receiving operations. In fact, the sender and the receiver were scheduled 1201 and 1202 times on average, respectively, instead of 1200 times.

Table 4: Successive Scheduling Experiments – Noiseless Setup.

Conducted experiments	100
Receiver operations (avg)	1200
Sender operations (avg)	1200
Receiver actually scheduled (avg)	1202
Sender actually scheduled (avg)	1201
Successive scheduling cases SS (avg)	1199
Standard deviation for SS	0.89
Successful transmissions ST (in bits on avg)	1189
Standard deviation for ST	4.151

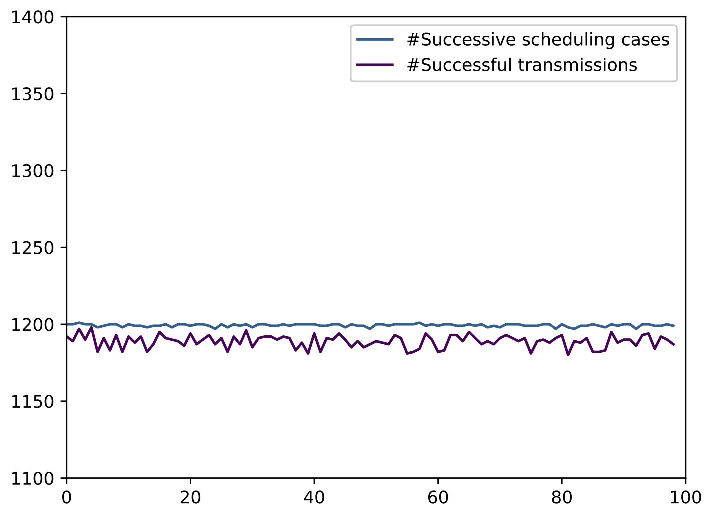


Figure 11: Noiseless Setup. Relationship Between SS and ST.

5.3.3.2 *Noisy environment.*

To investigate the traces an attacker leaves in a more realistic scenario, we conducted 200 experiments in a noisy environment. To simulate load on the system, we leverage the stressing tool stress-ng [Kin19]. We performed 100 experiments with background load level of 40% and 100 experiments with background level of 80%. The load is considered per CPU core.

Considering the load of 40%, for the 100 experiments 934 bits on average are successfully transmitted. The cases describing the successive scheduling between the sender and the receiver are 1138 on average. The standard deviation for the successful bit transmissions is 225, whereas for the successive scheduling it is 134.

In the other set of experiments characterized by a more noisy environment with load of 80%, 535 bits are successfully transmitted on average with standard deviation of 200, whereas the mean value of the successive scheduling cases is 905. The standard deviation in the latter case is 266. Table 5 gives a summary of the results along with the respective minimal and maximal values for successful transmission and successive scheduling cases.

The relationship between the number of successful transmissions measured in bits and the number of successive scheduling cases for the two sets of experiments

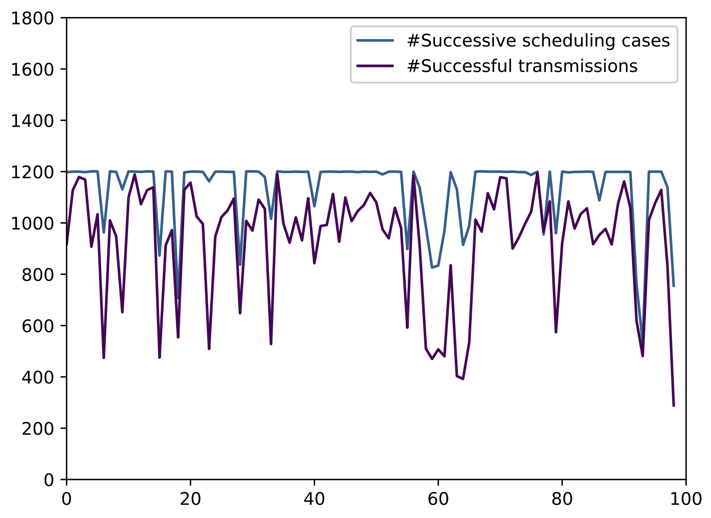


Figure 12: Load 40%. Correlation Between SS and ST.

Table 5: Successive Scheduling Experiments – Noisy Setup.

	40%	**80%**
Conducted experiments	100	100
Receiver operations (avg)	1200	1200
Sender operations (avg)	1200	1200
Successive scheduling cases SS (avg)	1138	905
Standard deviation for SS	134	266
Min SS (over all experiments)	534	255
Max SS (over all experiments)	1201	1201
Successful transmissions ST (in bits on avg)	934	534
Standard deviation for ST	225	200
Min ST (over all experiments)	289	194
Max ST (over all experiments)	1194	1030
Pearson correlation coefficient	0.8	0.44

in a noisy setup is shown in Figure 12 and Figure 13. Each point in the linecharts corresponds to a single experiment where a single experiment comprises 1200 receiver operations. Analogously to the noiseless case, the x-axis depicts the experiment number and the y-axis depicts the successful transmissions and the successive scheduling cases.

The blue linecharts in the figures show the number of successive scheduling cases for each of the 100 experiments per linechart. The purple linecharts show the number of successful transmissions per experiment. As can be seen from the figures, there is a similarity in the trends of the blue and purple linecharts. This is also confirmed by the positive correlation between the scheduling and the success of the attack. By applying Pearson's correlation coefficient, we obtain a correlation of 0.8, and a correlation of 0.44 for the 40% load and the 80% load, respectively.

As shown by the results given in Table 5, the correlation between the successful transmissions and the successive scheduling cases decreases in a more noisy environment. This relies on the violated atomicity of sender and receivers operations. In the setup with 80% load, the sender has been scheduled only 982 times

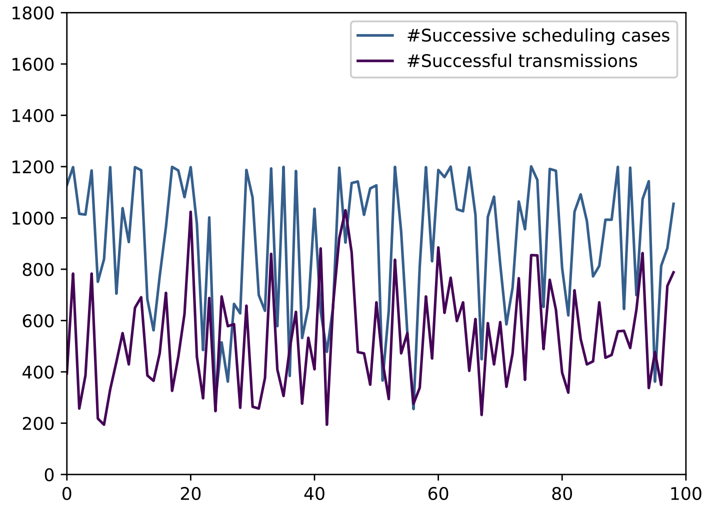


Figure 13: Load 80%. Correlation Between SS and ST.

on average, and the receiver has been scheduled 1178 times on average. Ideally, both the sender and the receiver should perform 1200 atomic operations (sending a bit and receiving a bit, respectively). This means that in the ideal case their processes should be scheduled exactly 1200 times each which increases the chances for atomic operations. As this is not the case in the noisy setup, the atomicity of operations is not always given, and this affects both the successful transmission, and the correlation with the successive scheduling.

5.3.4 *Busy Waiting Experiments and Results*

For the busy waiting experiments, the employed Prime+Probe attack consisting of a sender and a receiver processes. The employed attack resembles the attack described in [Mau+15], but the sender and receiver processes communicate with each other through the cache footprints left on the L1 cache instead of the LLC, as in [Mau+15]. The transmitted data is wrapped up by a header and a footer. In that way the receiver is able to identify the start and the end of the actual message. Per experiment we transmitted messages of length 5000 sent over the L1 cache as packets consisting of 1000 bits payload and 500 bits header and footer each.

Altogether, we have five settings for five different scenarios considering cases in the presence of an attacker and scenarios without an attack running. To simulate varied noise levels in the five setups, we leveraged the stressing tool stress-ng [Kin19], and employed background load levels of 0%, 40% and 80% per CPU core in the cases with a running attack. For the non-attacked system cases, we employed background noise levels of 40% and 80% per CPU. In each of the experiments the number of busy waiting events (BW) is logged simultaneously with the conduct of the attack. Such events are the issued sleeps, nanosleeps and waits provided by the operating system.

The results summarized in Table 6 demonstrate a significantly lower number of busy waiting events in a non-attacked system compared to the number of busy waiting events in an attacked system. The huge difference in the values is also visible from Figure 14 demonstrating the tremendous BW increase in an attacked system compared to a non-attacked system. This huge difference confirms that BW can be considered as a possible indicator for the presence of a suspicious process exploiting the core-private cache. Therefore, BW is a particularly suitable indicator for initiating further investigation due to the minimal overhead related to its measurement. A high BW value suggests also that investigating the successive scheduling with respect to the respective process can be beneficial.

As expected and similar to the results presented in Section 5.3.3, with the load increase, the number of successful transmissions decreases due to more contention for the cache. The number of BWs remains relatively stable in regard to all the scenarios in the presence of an attacker. This is expected, as the large number of BW is a characteristics of the attack implementation and not of the noise levels.

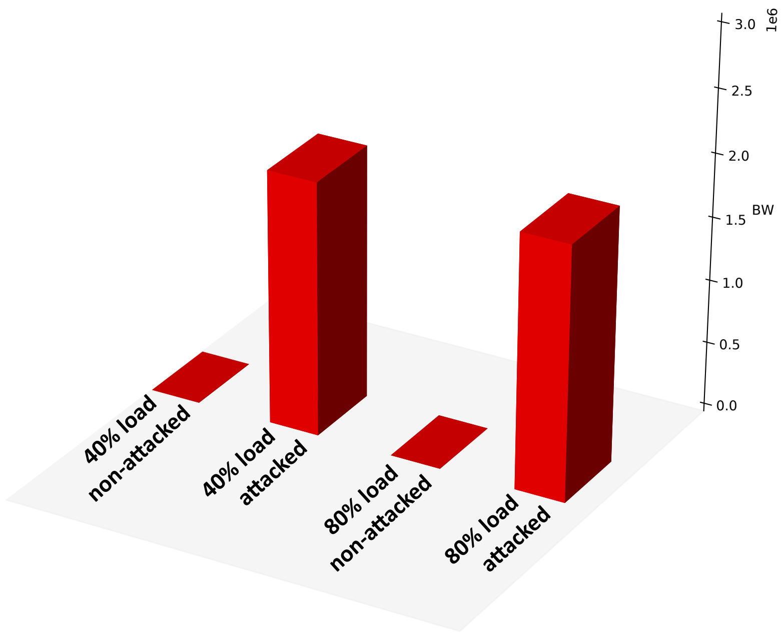


Figure 14: BWs in a Non-Attacked and in an Attacked System.

Table 6: Results in an Attacked and Non-Attacked System with Varied Background Noise Levels.

	Attack employed			No attack	
	0%	40%	80%	40%	80%
# Experiments	100	100	100	100	100
BW (overall)	2024152	2005113	2016318	4	16
BW (average per exp.)	20241.52	20051.13	20163.18	0.04	0.16

5.4 RELATED WORK

The security community has recognized the relevance of exploiting the cache as a side channel to compromise data confidentiality and the applicability of these exploits to the virtualized environment [Ris+09; WXW12; Xu+11; Zha+12]. Profound research in the area is still ongoing.

This section details the related work in regard to the role of the synchronization between the involved processes for the success of side-channel and covert-channel exploits.

5.4.1 *Synchronization and the Exploits of the Cache as a Side Channel*

The synchronization effect in the context of side-channel exploits has been studied already. In [Hu92b], Hu proposes a way to reduce the channel's capacity by leveraging fuzzy time system clocks as a measure to disable the proper synchronization for the adversary. Hu demonstrates experimentally that the channel capacity can be reduced in that way, and the proposed approach is promising in terms of enhancing the system security, however, it is impractical due to the involved restrictions on the execution environment for other benign processes.

In [Gra93], Gray considers the synchronization possibilities between the sending and receiving end in a covert-channel communication and estimates an upper bound about the capacity of the covert channel under fuzzy time in information theoretical terms, as introduced by Hu in [Hu92b]. The varied factors that impact the feasibility of cache-based exploits including the synchronization of the covert communication are also discussed in [Mau+17] by Maurice et al. Considering the possible synchronization issues, the authors defined three types of errors in the covert channel distinguishing between deletion, insertion and substitution errors.

Despite the threat posed by the side-channel exploits, the context of the execution environment often affects the performance of the different attack strategies. In [MKS12], Mowery et al. express doubts regarding the feasibility of side-channel attacks on certain architectures due to the applied preventive measures after an attempt to exploit the cache as a side channel, which turned out to be unsuccessful. Among the reasons for the lack of success, Mowery et al. point out core pinning which affects the synchronization between the attacker and the victim.

The vital role of proper synchronization over the accesses to the side channel for the successful data exfiltration has been mentioned in a variety of works that discuss different cache-based exploits. Among them are the ones described in [VRS14; Zha+12].

In this work, by considering the existing research in regard to the synchronization's role for the successful exploit of the cache, we focus on the traces or indicators a side-channel adversary leaves related to the required synchronization over the channel usage.

5.5 ATTACK INDICATORS – SUMMARY AND DISCUSSION

The conducted experiments focus on the traces or possible indicators for the presence of an adversary that exploits the core-private cache for covert communication. As synchronization indicators we considered the number of issued "sleeps", "nanosleeps" and "waits" in the available system logs provided by the operating system. The results show that in the absence of an attacker the number of these events is almost negligible, whereas the employed attack has led to a tremendous increase in the values indicating synchronization attempts. The obtained results are independent of the tested noise levels but dependent on the presence of an attacker. This result demonstrates that the synchronization indicators can be used post-mortem to identify suspicious processes which can be involved into a side-channel attack or a covert-channel attack considering specific attack implementations.

In regard to the experiments investigating the successive scheduling cases of the receiver and the sender in the context of a covert-channel attack, we studied the correlation between the beneficial scheduling of the sender and receiver processes and the success of the conducted covert-channel attack. Our results demonstrate a correlation of 0.44 between the scheduling and the success of the attack in the most noisy employed setup characterized by a background noise level of 80%. In the setup characterized by 40% load, the Pearson's correlation coefficient contends a correlation of 0.8. These results are expected, as the employed covert-channel attack exploits the core-private cache which is relatively small in size, and any process inbetween the execution of the sender and the receiver can affect the communication by erasing the cache footprint left by the sender process. In this case, the more noisier environment weakens the correlation between the successive scheduling cases and the number of successful transmissions. This relies on the violation of the atomicity of the sending and receiving operations.

The analysis of the SS by using the system logs shows a correlation between the number of SS and the success of the attack. However, this analysis is infeasible unless a suspicious process has been identified in advance due to the overhead related to the analysis of all the processes running on the system. Therefore, it can be only applied post-mortem if a suspicious process has been identified, e.g., through analyzing the BW of the involved processes. Although SS and BW cannot be applied directly for detecting of side-channel attacks, the experiments demonstrate that there are characteristics specific to the side-channel exploits which can be leveraged to enable their detection.

6 THE EFFECT OF THE HYPERVISOR SCHEDULING CONFIGURATION

The role of the CPU scheduling for the success of the cache-based exploits has been discussed in Part II. In this chapter, we further investigate the relationship between the execution environment and the exploitability of the core-private cache in the context of a covert-channel attack. For this purpose, we study the effect of the hypervisor scheduling parameters on attack feasibility in a virtualized environment. The contributions presented in this chapter are, partly verbatim, based on the material presented in [VSM15].

This chapter is structured as follows. Section 6.1 gives an overview on the addressed problem and the respective contributions. Section 6.2 presents the underlying system model. Section 6.3 details the scheduling approach employed at the hypervisor level along with the available scheduling parameters, whereas Section 6.4 presents the setup for the conducted experimental study and the implementation details. Section 6.5 reports on the experimental results regarding the impact of the hypervisor's scheduling configuration on the covert-channel attack. Section 6.6 investigates the state-of-the-art in the area, and Section 6.7 concludes the chapter.

6.1 OVERVIEW

There exist numerous execution environment properties which can impact the success of a side-channel or covert-channel attack. Among them is the employed CPU scheduling approach and configuration. Certain cache-based exploits are reliant on tricking the scheduler into specific behavior which enables frequent observations of the cache [Zha+12]. This is especially important when exploiting the core-private cache, which is in the focus of this chapter. The CPU scheduling, enabling proper synchronization upon the usage of the core-private cache, is a cornerstone for the success of such attacks mainly due to the limited size of the core-private caches.

The virtualized environment is even more endangered by the side-channel threat, as it relies primarily on the usage of shared resources across security boundaries. In such an environment, the hypervisor is responsible for decoupling the operating system from the hardware state of the underlying machine and for providing secure isolation between the co-located virtual machines. Hypervisor's scheduler, on the other hand, is responsible for scheduling the virtual CPUs (VCPUs) on the physical CPU cores and with that, it has a direct influence on the exploitability of the core-private cache as a side channel. The results presented in [VRS14] by Varadarajan et al. reinforce the expectation that the scheduling configuration has an influence on the exploitability of the cache as a side channel. The work presented in [VRS14] studies the impact of a single hypervisor scheduling parameter and proposes a defense mechanism against a particular side-channel attack. On this background, it would be beneficial to comprehensively study the effect of the rest of the hypervisor scheduling parameters on the feasibility of cache-based exploits.

To further explore the relationship between the execution environment and the exploitability of the core-private caches, we study the impact of the hypervisor scheduling configuration including diverse parameters on the feasibility of a covert-channel exploit. Hence, we consider a set of scheduling parameters employed in the the scheduling decisions. We investigate how the changes in the parameters' values can affect the success of the covert communication between a sender process and a receiver process through the core-private cache in a virtualized environment. We ascertain the effect of the scheduling parameters on the exploit's feasibility empirically by deploying a cache-based exploit in a controlled lab setup. We conduct experiments with different hypervisor scheduler configurations to assess the effect of the configuration on the feasibility of the performed attack. Our results suggest that the scheduling configuration plays a role in the success of core-private cache-based exploits and show that for the employed attack implementation and our controlled virtualized environment, certain configurations tend to be more secure in regard to the covert-channel attack feasibility. Nonetheless, the exact feasibility of the side-channel attacks highly depends on the particular attack implementation, and the acquired results apply to the specific implementation and setup.

6.2 SYSTEM MODEL

This chapter focuses on the virtualized environment. Therefore, the system model, shown in Figure 15, comprises the hypervisor, along with the hypervisor scheduler. In our experiments, the abused hardware resource is the core-private cache. Therefore, the core-private caches and the respective CPU cores are also a part of the system model. The core-private cache or L1 cache is the smallest and also the fastest cache. Out system model has also a level 2 cache and a LLC, which is larger and slower than the L1 and L2 caches, but is faster than the main memory. Whenever some data is requested, it is searched in the core-private cache first. After that the higher levels of cache are considered and if the data cannot be found there, it is fetched from the main memory into the cache. Logically, if the data has not been found in the cache (referred to as a cache miss), it takes longer to access it, as it has to be fetched from the main memory first. If the data has been found in the cache (referred to as a cache hit), accessing it is faster.

In the focus of this contribution are covert-channel attacks exploiting the core-private cache, which are characterized by cooperating attackers (i.e., a sender and a receiver), which communicate covertly through the core-private cache. As a part of the system and attacker models, we consider two distinct co-located virtual machines, referred to as Virtual Machine$_0$ and Virtual Machine$_1$. In the first one,

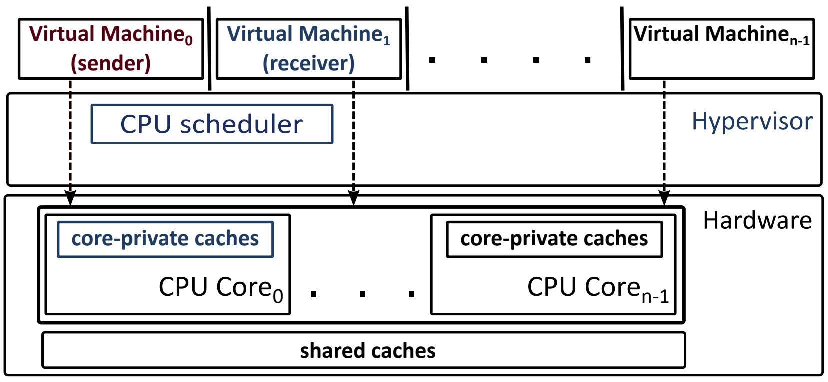


Figure 15: System Model – Virtualized Environment.

the processes of the sender are running, whereas in the second one, the processes of the receiver are running.

These two virtual machines can be co-located with further virtual machines, whereas we assume that the attackers have taken over at least the virtual machines Virtual Machine$_0$ and Virtual Machine$_1$. The attacker does not have control over the hypervisor. Through their actions, the attacker processes violate the presumed secure isolation between the separate virtual machines and the confidentiality in the system and can indirectly access data beyond the logical boundaries of their virtual machines.

For the empirical study presented in this chapter we rely on two distinct attacker models. In the first case, we assume a stronger attacker model characterized by the ability of the attackers to sustain better synchronization over their operations. The second attacker model is weaker and the synchronization over the involved sending and receiving operations between the sender and the receiver is not guaranteed.

6.3 hypervisor scheduler

The hypervisor scheduler is responsible for assigning the virtual CPUs, called VCPUs, to the physical CPUs of the underlying machine. A number of scheduling approaches are applied in hypervisors, e.g., Simple Earliest Deadline First (SEDF) [Sch], Completely Fair Scheduler (CFS) [KA14; M. 09], Borrowed Virtual Time (BVT) [DC99], etc.

Our goal is to study the impact of the scheduling configuration on the feasibility of exploiting the core-private cache in a covert-channel attack in a virtualized environment. For this purpose, we consider fair-share schedulers which are used in various hypervisors such as Xen [Xen] and KVM [KA14]. We focus on the Credit scheduler used in Xen [Sch; Xen], as it leverages most of the relevant scheduling parameters used also in other virtualized environments. It is worth noticing that scheduling parameters having the same conceptual function might be known under different names when used in other schedulers. We aim at investigating how the configuration of the hypervisor scheduler affects the feasibility of the employed cache-based exploit and study how changes in the scheduling parameters values affect the transmission of bits through the core-private cache as a communication channel between a sender and a receiver processes.

6.3.1 *Credit Scheduler*

The Credit scheduler implemented in Xen is a proportional fair-share CPU scheduler. The approach of the Credit scheduler aims at achieving fairness in respect to the resource allocation and assigns each VCPU a certain number of credits [CGV07; Zho+11]. The assigned credits are consumed and reduced with the consumption of CPU time. Each VCPU is associated with a priority which denotes whether the VCPU has used up all of its credits (i.e., the priority is "over"), or the VCPU has not exceeded its credits (i.e., priority is "under"). The scheduler keeps track of whether the VCPU has run out of credits. Thereby, the absolute value of the available credits is less important than the fact whether the credits have been exceeded. VCPUs of priority "under" can be scheduled until their credits are depleted. The VCPUs are assigned new credits periodically, and the number of their available credits is periodically updated, as well.

Table 7: Hypervisor Scheduling Parameters.

Parameter	Function	Default value
Load balancing	Determines whether automatic load balancing is used	automatic
Weight	Determines the relative CPU allocation among the VCPU	256
Cap	Imposes an absolute limit on max amount of CPU a VCPU can consume	0
Timeslice	Determines max time period a VCPU is allowed to run	30 ms
Rate limiting	Guarantees min amount of time a virtual machine is allowed run without being preempted	1000μs

6.3.2 *Hypervisor Scheduling Parameters*

The hypervisor scheduling parameters in the focus of this work are load balancing, cap, weight, time slice and rate limiting. These parameters are a part of the Xen Credit scheduler, but are relevant for other scheduling approaches, too, though they might be known under different names. An overview of the hypervisor scheduling parameters functions and their default values is given in Table 7.

LOAD BALANCING. The system load can be automatically balanced among the available CPUs by the Credit scheduler. For this purpose, the scheduler sustains a scheduling runqueue considering the active VCPUs priorities (i.e., "under" or "over"). Each of the running VCPUs is charged credits per clock tick. Commonly, the scheduler assigns the next local VCPU of priority "under" to run. If there is no such local VCPU, then a remote VCPU of priority "under" is scheduled to run. Otherwise, a local VCPU of priority "over" is scheduled and only then a remote VCPU of priority "under" can be scheduled.

The system administrator has the possibility to manually balance the load by assigning certain VCPUs to particular CPUs. This is known as CPU pinning or fixing the CPU affinity, and is used as a restriction to limit the CPU cores a VCPU can run on. The load balancing or CPU pinning is specified per VCPU.

weight. The weight, referred to as shares in other systems, is assigned per domain, and represents the significance of a VCPU in regard to the rest of the VCPUs. It indicates the relative CPU allocation time of a domain with respect to the other domains. A domain with a weight twice as high as the weight of another domain receives twice as much CPU time as the other. The scheduler involves the weight value for the calculation of the credits that a VCPU is assigned to. The acceptable weight values in the Credit scheduler are in the range from 1 to 65535 whereas 256 is used as a default value.

cap. Through the cap parameter, also referred to as limit, the Credit scheduler supports work-conserving and non-work-conserving modes. This parameter imposes a limit on the maximum amount of CPU a domain is allowed to consume. If the cap is set to a non-zero value, the scheduler is in a non-work-conserving mode. In this case, the cap limits the consumption of CPU cycles for the VCPUs even if more processing resources are available. This can lead to an increase in the time for a task completion. A domain with a cap of 0 is not capped which corresponds to scheduler work-conserving mode. In this case, the CPU is idle only if no VCPU is waiting to be scheduled. The cap value is expressed in percentage per domain, where a cap of 100 means one physical CPU, a cap of 50 means half a CPU, etc. Similarly to the weight, the cap is also involved in the computation of the credits.

timeslice. The timeslice, also referred to as quantum in other contexts, determines the maximum period of time a VCPU is allowed to run. Once this period has elapsed, the VCPU is de-scheduled by the scheduler, and another VCPU is scheduled to execute on the CPU. Such a preemption takes place if the execution time of the VCPU exceeds the timeslice. It has to be noted that the timeslice defines the maximum time a VCPU is allowed to run, but it does not guarantee that a VCPU will run without being preempted for the whole timeslice period. The timeslice value is the same for all the domains, and it can have performance implications. For example, for latency-sensitive workloads having a long timeslice can be devastating, while a long timeslice can have a positive effect on computationally intensive workloads.

context-switch rate limiting. The rate limiting parameter determines the minimum amount of time a virtual machine is allowed to run without being preempted if it is set to a value different than zero. The rate limiting value is given in microseconds. The default value of this parameter in the Credit scheduler is set to 1000 microseconds, where the rate limiting can be disabled by setting its value to 0. Similar to the timeslice, the rate limiting value can have performance implications depending on the workload requirements.

6.4 HYPERVISOR SCHEDULING EVALUATION

This section presents the experimental setup and the implementation details involved in our study.

6.4.1 *Experimental Setup*

We employ two distinct experimental setups. In both cases, our experiments are conducted on commodity hardware. The used systems are equipped with Intel Quad Core i7-4770 CPU@3.4GHz and Intel Skylake processors. On the first system, we run experiments that involve a stronger attacker model, as mentioned in Section 6.2. In the scenario, the attackers (i.e., a sender and a receiver processes) are presumed to have control over the synchronization of their sending and receiving operations. This is not a realistic scenario, but we leverage it to be able to study the effect of the scheduling with respect to the other co-located processes on the attack success independently of the issues that stem from the violation of operations synchronization. For this setup, we use Xen 4.3 hypervisor [Xen] as a virtualization solution equipped with an administrative guest domain with special privileges, called Dom-0. Dom-0 is leveraged to create or destroy other virtual machines, referred to as guests or to adjust various parameters, including the scheduling configuration.

The other experimental setup runs on Intel Skylake processor on which we use Xen 4.9 as a virtualization solution. For the conduct of these experiments, we do not rely on a guaranteed synchronization over the sending and receiving operations.

In both cases, we perform a covert-channel attack which is based on the Prime+Probe strategy and leverages the core-private cache for covert data transmission. For this purpose, we use two physically co-located virtual machines that reside on the hypervisor. The receiver processes are running in one of the virtual machines and the sender processes are running in the other virtual machine. The goal of the sender and receiver processes is to communicate via the core-private cache by the means of predefined cache access patterns.

To analyze the effect of the scheduling configuration on the exploitability of the core-private cache in this context, we vary the values of the scheduling parameters introduced in Section 6.3 while performing a covert-channel attack. We measure the error rate with respect to the received data to assess the the success of the attack. The error rate is determined by the ratio of the number of incorrectly decoded bits (including also the bits which cannot be decoded) to the number of all the sent bits. For practical reasons, we consider deliberately chosen values for the scheduling parameters that are expected to result in different levels of noise on the system.

6.4.2 *Implementation Details*

In the context of the employed covert-channel attack, the sender and the receiver processes try to communicate with each other secretly through the core-private (i.e., L1 data) cache. As they cannot access the cache directly, both processes allocate buffers as bit as the L1 cache. The allocated buffers are aligned by the cache line size. The processes adhere to the Prime+Protocol (cf., Part I) where the sender accesses parts of the cache to encode one bit of information by evicting receiver data from the cache. The receiver times its accesses to the respective cache parts to obtain the data. In the first experimental scenario the adversaries synchronize explicitly by means of POSIX sockets. In this scenario, the receiver does not start any receiving operation unless the receiver has received a confirmation from the sender regarding the completion of the sending operation. Analogously, the sender sends a new bit only if it has obtained a confirmation about the receiver that the receiving operation has been completed. In the second setup, the attacker model is weaker. The attackers try to synchronize with each other, but do not use sockets. Instead, they yield the CPU upon each completion of their operations.

Similarly to the attack described in Section 5.3, also here, we have to cope with challenges such as prefetching and addressing uncertainty. Nonetheless, as the employed cache is the core-private cache, accessing specific cache parts based on data from the virtual addresses, is possible. In addition, we do not take into account the access times to a single cache line, as it is nearly infeasible to differentiate between cache hits and cache misses in a virtualized environment in the context of a covert-channel exploit. Therefore, we consider accumulated values to increase the measurable difference when sending bits 0 and 1. With one sending operation, the sender transmits one bit of information and with one receiving operation, the receiver tries to decode one bit of information. Ideally, the receiver tries to decode the bit of information directly after the sender has encoded it into the cache. Otherwise, another process can overwrite the cache access pattern meanwhile.

To accurately measure the access times to the cache, we make use of the RDTSC assembly instruction. We address the issues stemming from potential out-of-order execution in regard to the sending and receiving operations, by making use of the CPUID instruction to serialize instruction execution and *volatile* as a memory barrier.

Both experimental setups are not exposed to additional load. The system is not stressed, and no additional workloads except for the basic load are running on the system.

6.5 THE ROLE OF SCHEDULING PARAMETERS – RESULTS

This section presents the results of our experimental study on the impact of the scheduling configuration on the feasibility of a covert-channel attack that abuses the core-private cache. The results are presented per scheduling parameter. Per experiment, we vary a single parameter, and the rest of the parameters are set to the default values.

6.5.0.1 *Load balancing*

To assess the effect of the load balancing on the attack success, we consider four different configurations involving the two virtual machines of the sender and the receiver (referred to "sender" and "receiver" for brevity from now on) and the administrative guest Dom-0. Dom-0 also competes for resources along with the sender and the receiver. The tested hypervisor scheduling configurations are listed below.

- **LB1:** The sender and receiver are pinned to an identical CPU core whereas Dom-0 is pinned to another CPU core
- **LB2:** The sender and receiver are automatically load balanced and Dom-0 is pinned to a particular CPU core
- **LB3:** The sender, receiver and Dom-0 are pinned to an identical CPU core
- **LB4:** The sender and receiver are pinned to distinct cores

Through the load balancing, the hypervisor practically controls the availability of the core-private cache as a side channel or as a communication channel between the sender and the receiver in our experiments. Therefore, our speculation is that the configuration LB1 is the most beneficial one from attacker perspective, as in this case, the core-private cache is at disposal to the adversaries as a communication channel. Moreover, in this case Dom-0 does not compete for the same resources as the sender and receiver. This logical observation is also substantiated by our experimental results. We obtain the lowest error rate compared to the rest of the configurations which is of 38% for the stronger attacker model. The error rate increases and reaches 67% for the experimental setup with a weaker attacker model. This deterioration in the results is expected, as the atomicity of the send and receive operations is not guaranteed in such a scenario.

As expected, the worst case controlled through the load balancing from attacker perspective is LB4. In this case, the channel is not available for covert communication, as the sender and receiver processes are scheduled to run on distinct CPU cores and do not share the the core-private cache. In this case, the stronger attacker model exhibits an error rate of almost 99%. This is the worst configuration case also in regard to the weaker attacker model. Similar results are obtained for LB2, when the sender and the receiver are automatically load balanced. This results in an error rate almost as high as the error rate in LB4. The stronger attacker model exhibits an error rate of 97% in this case. The obtained error rate in LB3 is worse than the error rate measured for LB1, but better than the values measured in LB2 and LB4. For the attacker model with guarantees regarding the synchronization over the sending and receiving operations, the error rate reaches 72%.

To ensure the availability of the core-private cache as a communication channel for the rest of the experiments, we use LB1 configuration with regard to the load balancing of the virtual machines.

6.5.0.2 *Weight*

To assess the impact of the weight values on the feasibility of the deployed covert-channel attack with respect to both attacker models, we distinguish three scheduling configurations involving the weight parameter.

- **W1:** Both weights are set to their default value 256
- **W2:** Sender weight is set to 512 and receiver weight remains 256
- **W3:** Receiver weight is set to 512 and sender weight remains 256

Our results demonstrate that the three chosen weight configurations do not affect the error rate significantly considering both attacker models. The configuration W1 actually overlaps LB1, as the receiver and sender weight values are set to the default values of 256. In this case, the error rate is 38% for the stronger attacker model. The obtained results show an error rate of 42% and 44% for W2 and W3, respectively. The weight parameters controls the CPU time a virtual machine is assigned to with respect to the other co-located virtual machines. In the implementation with regard to the stronger attacker model, after the sender has finished a sending operation, the receiver is informed upon its completion. Only then, the receiver starts the receive operation. Analogously, the sender waits for the receiver to complete the receive operation. Therefore, even if the receiver of the sender are assigned more CPU time, this does not have a significant influence on the success of the attack.

6.5.0.3 *Cap*

To investigate the effect of the cap parameter on the success of the attack, we consider three values.

- **C1:** The sender and receiver are not capped (cap is set to 0)
- **C2:** The sender and the receiver caps are set to 40
- **C3:** The sender cap is 1 and the receiver cap is 0

In the tested configuration, we consider also the cap default values in C1 which overlaps with configuration LB1. The influence of the cap parameter highly depends on the employed implementation. It will have an effect on the attack, if the sender or receiver run long and have to be interrupted in the middle of their operations. This can lead to a violation of the atomicity of the sending or receiving operations. However, as our stronger attacker model ensures the synchronization over the operations of the receiver and the sender, this is not expected to have an effect on the error rate of the attack. If the sender and the receiver are capped at 40%, we exhibit almost the same error rate for C2 (41%) as in the best case C1 (error rate of 38%).

In the case of a weaker attacker, if the sender is preempted in the middle of a send operation, the error rate increases. This is demonstrated also by the obtained results. For C3, we obtain an error rate of 78% compared to an error rate of 67% for C1. In this case the employed time slice is 30ms and the sender uses 1% CPU. This can lead to frequent preemptions of the send operation and affect the success of the attack. C2 does not affect the error rate significantly, as a cap of 40% is sufficient for both the sender and receiver processes to complete the send and receive operations before the time has elapsed.

6.5.0.4 *Time slice*

The time slice limits the maximum time a virtual machine is allowed to run. Once the time slice has elapsed, the virtual machine is de-scheduled. To be able to study the effect of the time slice parameter, we employ the attacker model which relies on a weaker (and more realistic) assumption regarding the synchronization between the sender and the receiver and does not ensure atomicity of the send and receive operations. In this context, we have tested three values of the time slice parameter.

- **T1:** The time slice is set to 1ms
- **T2:** The time slice is set to the default value of 30ms
- **T3:** The time slice is set to 1000ms

Configuration T2 corresponds to configuration LB1 (30 is the default value of the time slice) and exhibits logically the same error rate of 67%. Our results do not demonstrate a significant impact of the time slice on the error rate which characterizes the covert-channel communication. A reason for that is that the time slice limits the CPU time a virtual machine is allowed to run, but does not impose that a virtual machine has to run that long. As also in the weaker attacker model, the adversaries try to synchronize with each other, they voluntary yield the CPU after the completion of a send or receive operation. Therefore, a long time slice does not result in an error rate deterioration. The shortest time slice period, on the other hand, is 1ms which is sufficient for the sender and the receiver to complete a send or a receive operation before being interrupted. Therefore, the effect of the time slice parameter on the error rate is minimal. Nonetheless, with the decrease of the time slice, the error rate increases slightly. The error rate values are 68% , 67% and 66% for T1, T2 and T3, respectively.

6.5.0.5 ***Rate limit***

Also in this case, we employ the weaker attacker model to analyze the impact of the rate limit on the success of the covert-channel attack. We consider the following three configurations.

- **RL1:** The lowest possible value - 100
- **RL2:** The default value - 1000
- **RL3:** The highest possible value - 500000

To study the effect of configuration RL3, we set the time slice parameter to 500ms, as the rate limit may not exceed the time slice period. For the default configuration (i.e., RL2), the error rate is 67%. An increase in the rate limit parameter and in the time slice parameter result in a slightly increased error rate (69% for RL3), but in this case two of the hypervisor scheduling parameters are changed which makes the comparison of the effect of the rate limit with the other cases harder. A decrease in the rate limit parameter increases the error rate to 73% (cf., RL1). In this case, the minimum runtime guarantee for the sender and the receiver is set to 0.1ms. This does not mean that the sender and the receiver will necessarily be interrupted after 0.1ms , but increases the chance for frequent preemptions which can lead to violation of the atomicity of the operations. This case is analogous to C3, in which the sender is preempted frequently during the send operation, as it is capped at 1%.

The presented results indicate that certain hypervisor scheduling parameters affect the feasibility of the deployed covert-channel exploit. They show that the parameter which affects the channel at most is the load balancing. The results in our both experimental setup employing a weaker and a stronger attacker models confirm that the channel can become unavailable due to the pinning of the sender and receiver to distinct cores. On the contrary, if the adversaries are pinned to the same core, the associated error rate is lower than when they are automatically load balanced among the CPU cores. Moreover, the automatic load balancing of the sender and receiver pose a challenge to the adversaries, as the availability of the channel is affected. Very low values of time slice and cap can also affect the covert-channel communication. Nonetheless, this depends on the implementation of the attack (the execution times of the receiver and the sender), the employed attacker model and the underlying hardware.

6.6 RELATED WORK

Side-channel attacks have been a known threat for a long time. However, due to the prevalence of complex systems such as the Cloud which offer shared resources to their customers, these attacks became even more popular during the last decade [Ris+09; Zha+12]. Such complex systems are often reliant on virtualization solutions to multiplex the available hardware resources among their customers. This chapter overviews the research of the security community on the scheduling effect within the virtualization environment on the exploitability of the cache as a side channel.

6.6.1 *The Scheduling Effect*

There exist a number of factors characterizing the execution environment including the scheduling policies which impact the feasibility of side-channel attacks. Such factors have been employed to enhance the systems security against side-channel attacks in diverse approaches [KPMR12; LGR13; MDS12; Shi+11; Ste+13], but many of them are commonly targeting at a specific attack type. In some attack strategies, the scheduling policy controls the granularity of observations an adversary can obtain or affect the synchronization between the cooperating attackers in a covert-channel attack.

In [Hu92a], Hu emphasizes on the major role the scheduling policy plays in the hardware timing covert-channels exploits stemming from the shared CPU usage. To address this threat, Hu proposes a scheduling scheme [Hu92a] for the VAX security kernel. Hu's idea is to eliminate the covert channel by changing the processes execution order. In the proposed approach, the execution order of the processes is not based on their readiness to be executed. Instead, the initial execution order and the CPU allocation are determined through a time slot list. This limits the alternating execution of the malicious processes in a covert-channel attack which is often a prerequisite for the success of the covert communication. Although the focus of Hu's work is on the VAX security kernel, it demonstrated the relationship between the scheduling scheme and the hardware timing covert-channel exploits.

In [VRS14], Varadarajan et al. addresses the threat related to the side-channel exploits by modifying the value of one of hypervisor parameters in a virtualized environment. The authors assess the impact of the change both on the system performance and the exploitability of the cache as a side channel. To affect the granularity of the observations on the cache an attacker can achieve, Varadarajan et al. adjust the rate limit parameter to provide the VMs with a minimum runtime guarantee. This approach restricts the ability of an adversary to frequently preempt victim's VM which is a core element of the attack, described in [Zha+12]. On this background, the work presented in [VRS14] proposes a change in the hypervisor scheduling configuration as a measure against a specific side-channel. Our goal is to assess the impact of the spectrum of hypervisor scheduling parameters considering an exploit of the core-private cache.

To eliminate the threat stemming from the usage of cache-based side channels, Stefan et al. [Ste+13] propose an instruction-based scheduling approach. Their work does not address virtualized environments, but the derived conclusion that the scheduling approach can be used as a preventive mechanism against cache-based exploits raises awareness regarding the role the scheduler plays in security. Stefan et al. highlight that deterministic information flow control systems are vulnerable to cache-based timing channels especially if time-based scheduling is used.

Zhou et al. [Zho+11] report on a hypervisor scheduling vulnerability which can be exploited by an adversary to obtain more CPU time for its VM. This work, despite being only indirectly related to the feasibility of cache exploits, demonstrates that the choice of the scheduling approach and its parameters is crucial not only from performance but also from security perspective.

6.7 CONCLUSION AND SUMMARY

This chapter focuses on the impact of the hypervisor scheduling parameters on the success of covert-channel attacks exploiting the core-private cache. It experimentally analyzes the role of the scheduling configuration and how the scheduling parameters affect the covert communication in our controlled setup considering two distinct attacker models. Our results show that the configuration of the hypervisor scheduling parameters has an impact on the feasibility of covert-channel attacks for the considered experimental setup.

The conducted empirical study demonstrates that the load balancing is the parameter that affects the communication over the side channel at most. This is logical, as this parameter basically controls the availability of the communication channel. The most beneficial configuration from attacker's perspective is when both sender and receiver processes are pinned to the same CPU core and any additional workload is deployed to the other cores. The results in our setup demonstrate that the automatic load balancing also affects the communication conducted in the employed covert-channel exploit. Very low values for the cap and the time slice parameters can also have a negative effect on the covert communication through the core-private cache in our empirical study by increasing the associated error rate. Based on our results, for a more secure virtualized environment, we recommend using the hypervisor's automatic load balancing feature. The choice of an adequately low value for the cap and the time slice parameters is also recommended, but potential performance degradation and its implications should be taken into consideration.

Part IV

ON THE DETECTION OF SIDE-CHANNEL ATTACKS

7

SPYALARM – A RELIABLE CACHE-BASED EXPLOITS DETECTOR

In Chapter 5, we identified potential indicators of compromise which is a step towards the post-mortem analysis of cache-based exploits. Nonetheless, the detection of side-channel and covert-channel attacks leaking information through the cache still constitutes a challenge for the security community due to the lack of definite traces suggesting an ongoing cache exploit. In this chapter, we come to the last thesis contribution focused on the runtime detection of cache-based exploits. The contributions presented in this chapter are, partly verbatim, based on material from [VGCS].

The remainder of the chapter is structured as follows. First, we give an overview of the problem in Section 7.1. Then, we present the system and attacker models underlying the chapter in Section 7.2. We describe the proposed approach and architecture in Section 7.3 and Section 7.4, respectively. Then, we detail our setup in Section 7.5 and the experimental results in Section 7.6 and discuss the challenges related to detecting exploits of the cache in Section 7.7. Section 7.8 gives an overview on the related work. Finally, we present the concluding remarks in Section 7.9.

7.1 THE NEED TO SPY THE SPY

In recent years Cloud computing has become nearly ubiquitous due to its advantages of scalability, cost efficiency and high availability. However, the flexible sharing of resources that is one of the causes for these properties has security implications. Therefore, despite its benefits, Cloud computing can also be considered as a business risk [Gar18], in part due to the existence of cache-based side-channel and covert-channel attacks. Hence, this chapter focuses on attacks that use the cache as a side channel to leak information.

Exploiting the cache as a side channel can be a quite powerful mechanism to leak information, as such leakage is challenging to detect mainly due to the lack of definite traces a side-channel adversary leaves. The traces left when a side-channel attack is performed are hardly distinguishable from the normal operation of the system or its benign processes. Moreover, the cache is a component which can be accessed without privileged access rights and is usually not a subject to special security policies. Thus, intrusion detection systems that rely on access rights are commonly limited to the detection of other types of attacks and turn out to be unsuitable to detect cache-based side-channel exploits.

To enhance the security of systems that offer shared resources across security boundaries, such as the Cloud, a reliable detection mechanism that can spot the usage of the cache as a side channel is required. Theoretically, such cache-based exploits are almost always possible. However, as already discussed in the previous parts of the thesis, various properties of the execution environment affect the cache exploitability. Moreover, certain attack implementations leave traces on the system which can be investigated post-mortem to be considered as indicators for an attack. On this background, the traces that the side-channel adversaries leave should be systematically studied and their applicability as an enabler to uncover side-channel exploits should be investigated. A robust side-channel and covert-channel attack detection mechanism can significantly improve the security of any system, including the Cloud, given the numerous side-channel attack approaches proposed over the last years.

The objective of this thesis contribution is to address the lack of a reliable detection that uncovers covert-channel and side-channel attacks. Therefore, we propose SpyAlarm, a robust detection mechanism, which reliably identifies cache-based exploits by leveraging a combination of hardware performance counters and software events. SpyAlarm is tailored to detect side-channel and covert-channel attacks exploiting the cache at runtime. The contributions in this chapter include a detection approach which comprises two phases: definition and detection. In the definition phase, we derive a model from covert-channel attack implementations relying on the Flush+Flush (cf., [Gru+16]), Flush+Reload (cf., [YF14]) and Prime+Probe attack strategies, whereas in the detection phase, the actual runtime detection takes place. Based on the approach, we define an architecture which relies on continuous monitoring of the system for the attack detection and can be integrated and applied in both non-virtualized and virtualized environments.

To evaluate the reliability and applicability of SpyAlarm, we implement it as a stand-alone application to monitor the side channel. We employ different sampling configurations and training parameters to achieve a high detection accuracy and low false positive and false negative rates. Our empirical results demonstrate that SpyAlarm can reliably detect two state-of-the-art side-channel attacks unseen by the detector while exhibiting a low false positive rate.

7.2 SYSTEM AND ATTACKER MODELS

In this section, we detail the system and attacker models underlying SpyAlarm. Additionally, we discuss the application restrictions of the presented detection approach.

SpyAlarm is tailored to detect side-channel and covert-channel attacks exploiting the cache. The proposed approach is applicable to both virtualized and non-virtualized environments, but as the validation of the approach is conducted in a non-virtualized environment, we narrow down the system model presented in Figure 3 (cf., Section 2.4) to the non-virtualized environment, as presented in Figure 16. As the proposed approach leverages system monitoring components which are not accessible from user-land, SpyAlarm has to be granted privileged access to the necessary data, including logs of performance counters and software events.

The focus of this work is the cache as a side channel. Therefore, the hierarchy of caches along with the CPU are part of the system model representing the means of possible covert communication or side-channel exploit. Thus, our system model includes the core-private data and instruction caches which are typically separated at level 1, and LLC which is shared among the CPU cores and unified containing both data and instructions. The core-private cache is fast and small in contrast to the big and slower LLC.

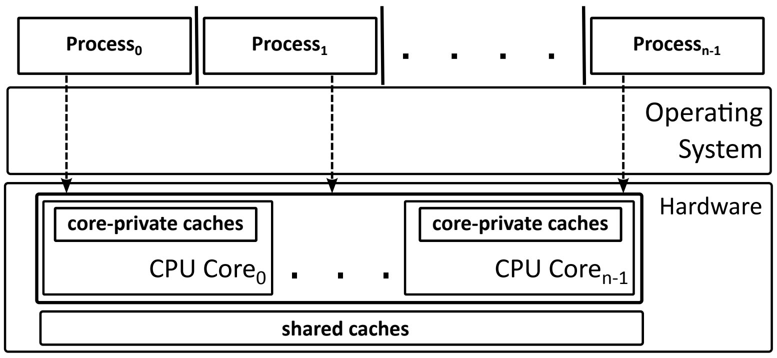


Figure 16: SpyAlarm System Model.

The employed adversaries should be able to time their accesses to the cache, but as sampling the TSC does not require special privileges, the attacker can be any arbitrary process. However, to deploy a LLC cache attack, huge pages are needed and are therefore enabled on the system. We consider both side-channel attacks, characterized by a victim and an attacker that spies on the victim cache usage, and covert-channel attacks, characterized by two cooperating attackers that communicate via the cache. Both attack scenarios are detailed in Part I.

SpyAlarm itself requires root access. Moreover, our detection approach assumes that the attacker controls one or more non-privileged processed running on the system, but excludes adversaries with privileged access to the system, as the one described in [Bra+17].

7.3 APPROACH

This chapter describes our detection approach, called SpyAlarm. SpyAlarm comprises two main phases, referred to as a Definition phase and a Detection phase.

7.3.1 *Definition Phase*

The Definition phase establishes the distinction between the normal and. abnormal system behavior. It consists of the steps: (i) indicators selection, (ii) training data acquisition, and (iii) signatures derivation. The steps of the approach are represented in Figure 17 and detailed in the following paragraphs.

INDICATORS SELECTION. In the indicators selection step, the existing known attacks are investigated and their specific characteristics are defined. This analytical step establishes the basis to define adversarial behavior in the system and is essential for the presented approach, as the quality of the selected indicators reflect on the detection of the attack. The chosen indicators have to be representative for the exploits and have to be able to characterize attacker behavior. An improper selection can result in a high false positives rate or undiscovered attacks. The union of attack indicators which have to be studied and continuously monitored is the outcome of this analytical phase.

The attack indicators are divided into three groups: (i) usage, (ii) synchronization, and (iii) measurements. Each of the categories comprises different measurable events of the system. More details on the individual categories are given below.

- usage - this category refers to events that describe the indirect effect of an attacker process on the system. Such usage effect could be, for instance, an increased cache miss ratio which is a characteristic of the Prime+Probe attack strategy.
- synchronization - this category refers to the effects of the synchronization over the usage of the cache as a side channel. To obtain fain-grained observations, many side-channel and covert-channel attack implementations leverage explicit synchronization. This often results in an increased number of sleeps, for example. Other attack indicators typical for this category are an increased number of context switches or an increased number of inter-processor interrupts. These attack indicators have been also discussed in Part III.
- measurements - this category refers to the effect of the conducted measurements required for the execution of the attack. A side-channel attacker exploiting the cache, for instance, often times its various operations with the cache. Thus, frequent samples of the RDTSC can be considered as a possible attack indicator.

TRAINING DATA ACQUISITION. SpyAlarm relies on the acquisition of training data in regard to the defined indicators of compromise. The defined attack indicators are monitored and logged for diverse scenarios including both cases in the presence and absence of an attacker and various noise levels. The larger the diversity of scenarios and workloads are, the better the chances for good predictions results are, as this would result in a larger training set for the applied training algorithm. Therefore, the availability of numerous implementations of existing cache-based exploits is vital for this step and represents a foundation for good prediction results.

SIGNATURES DERIVATION. The training data collected in the previous step is used for the training of the detector. The values of the indicators of the compromise gathered for both malicious (i.e., attacker) and benign (i.e., non-malicious) processes are leveraged as structured data which is used for the training. The attack indicators represent the feature set applied in the training algorithm and

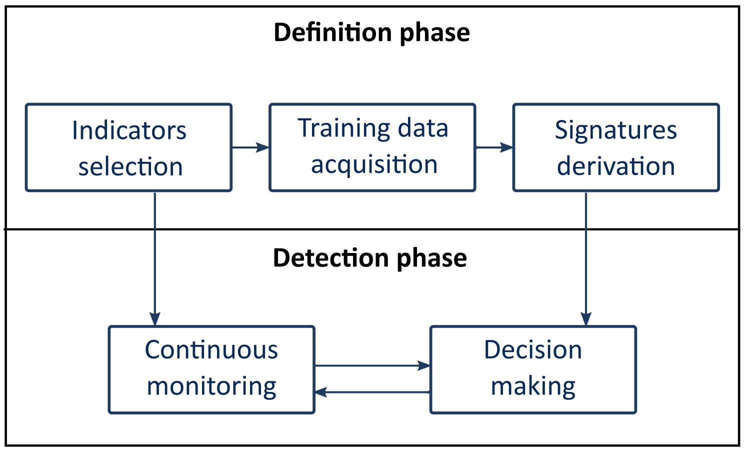


Figure 17: Detection Approach.

the y set corresponds to the categories "attack" or "no attack". The outcome of this step is a model which is trained to distinguish signatures of cache-based exploits discerning benign and malicious processes given a vector of indicators of compromise as input.

7.3.2 *Detection Phase*

Given the constraints on the normal and abnormal behavior of the system defined in the previous phase, in this phase the system is monitored and the attack indicators are collected, analyzed and evaluated. Based on the evaluation, a decision is taken in regard to the presence or absence of an attacker exploiting the cache. The Detection phase comprises a continuous monitoring step and a decision making step, which are further detailed in the next paragraphs.

CONTINUOUS MONITORING. As the name suggests, in this step the system is being continuously monitored. The indicators of compromise, including the ones defined in Chapter 5, are collected at runtime for each of the processes which run on the system.

DECISION MAKING. In this step, the continuously monitored indicators of compromise are periodically assessed and a decision is taken in respect to each process regarding being malicious or not. As soon as a process is considered being compromised, the process is descheduled and an indication is created, consisting of the identification of the suspected process and a list of the processes, running along with the suspected one.

7.4 ARCHITECTURAL OVERVIEW AND IMPLEMENTATION

This section details the proposed SpyAlarm architecture and our implementation of the detector.

7.4.1 *Architecture*

The architecture of SpyAlarm consists of six major components: Indicators of Compromise, Data Collector, Preprocessor, Trainer, Monitoring Agent and Attack Detector. The components are detailed in the following paragraphs and are shown in Figure 18. The components depicted in blue color in the figure require privileged access to the system.

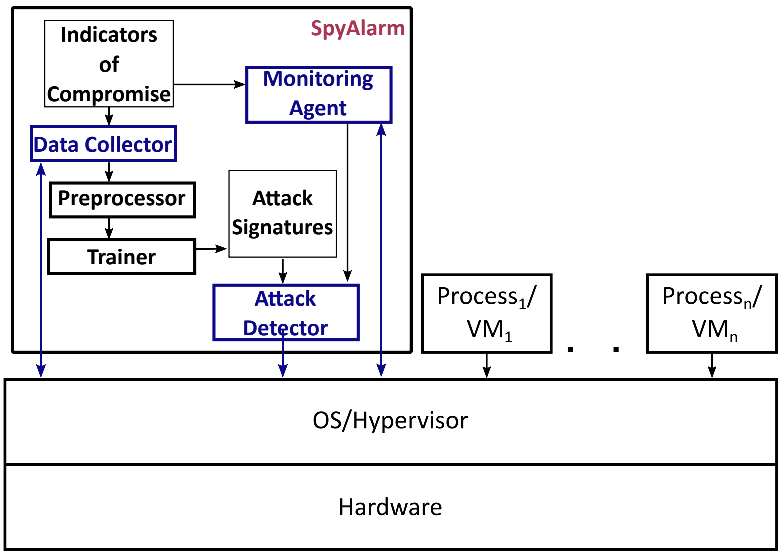


Figure 18: Overview of the Architecture.

7.4.1.1 *Indicators of Compromise Descriptor*

The Indicators of Compromise Descriptor is a file containing the analytically derived indicators of compromise. These indicators are used to derive attack signatures as well as for the actual attack detection. For each attack a set of indicators is specified and included along with the measurable events describing the exploit and the corresponding indicators. Certain indicators can apply to various attacks, therefore a single occurrence of each measurable event is considered. The indicators collection can be extended anytime a new attack is defined, and measurable events can be derived. The currently included attack indicators fit into the categories discussed in Section 7.3.

7.4.1.2 *Data Collector*

This component is leveraged to collect a sufficient number of training samples of the indicators of compromise in both attacked and non-attacked scenarios. For the collection of training data, the Data Collector needs privileged access to the performance counters and the software events logs, as visible from Figure 18.

7.4.1.3 *Preprocessor*

As its name suggests, this component is employed to preprocess the data gathered by the Data Collector, so that it can be easily processed in the next steps of the approach.

7.4.1.4 *Trainer*

The trainer leverages a supervised learning algorithm to derive constraints or signatures indicating an attack by creating a model. For that, the algorithm uses the preprocessed training data and extracts features vectors along with their mapping to one of the classes "benign" or "malicious".

7.4.1.5 *Monitoring Agent*

The continuous data acquisition in the system takes place in a component, called monitoring agent. The monitoring agent monitors the attack indicators during runtime and forwards their values to the Attack Detector.

7.4.1.6 *Attack Detector*

The actual attack detection happens at a component, called Attack Detector. It takes as input the data collected from the Monitoring agent along with the data provided by the Trainer. The latter provides a model for evaluation of the obtained values to check for adversaries. Based on the input, the Attack Detector decides on whether there is an ongoing attack or not. The logic of the Attack Detector does not have to be implemented at the hypervisor or OS level, but can be encapsulated in a dedicated virtual machine or as a process with privileged access.

7.4.2 *SpyAlarm Implementation Details and Parameters Consideration*

SpyAlarm is implemented in Python as a stand-alone tool.

7.4.2.1 *Indicators of Compromise Descriptor, Data Collector and Preprocessor*

The analytically derived Indicators of Compromise are software events and performance counters that have to be collected or monitored both during the Definition and Detection phases. For this purpose, we sample various events using the kernel perf subsystem including context switches, sleeping and waiting events Table 8 gives an overview on the full list of currently employed software events and performance counters in SpyAlarm. They constitute the structured training data set collected during the Definition phase by the Data Collector and are used by the supervised learning algorithm to train a prediction model. During the Detection phase, these events and counters are continuously monitored and based on their values and the created prediction model, the detection of ongoing attacks takes place.

The Data Collector is a shell script which uses the perf tool for data gathering (leveraging the perf record option) and to prepare the data for further analysis

Table 8: Monitored Software Events and Performance Counters.

Indicators of Compromise Type	Perf Measurable Event
Usage Effect	offcore_response.other.any_responses
Usage Effect	cache-misses
Usage Effect	cache-references
Usage Effect	LLC-stores
Usage Effect	instructions
Usage Effect	branch-instructions
Measurements	msr:write_msr
Synchronization Effect	syscalls:sys_enter_sched_yield
Usage Effect	task-clock
Measurements	msr:read_msr
Measurements	msr:rdpmc
Synchronization Effect	cs
Usage Effect	tlb:tlb_flush
Usage Effect	cpu-cycles

(using the perf report option). The collected data is stored in data files and given as input to the Preprocessor. The Preprocessor is a script which takes as input the collected software events and performance counters and preprocesses them to the right format which is then fed into the Trainer. For this, the Preprocessor parses the data obtained from the Data Collector, extracts the feature vectors, maps them to the classes *malicious* or *benign* and removes the unnecessary information. The output of the Preprocessor consists essentially of the feature vectors and the corresponding value representing the attacker class.

7.4.2.2 *Trainer*

We implemented the Trainer in Python, and leveraged the TensorFlow framework [Aba+15] for the supervised learning on the training data set. To select an accurate prediction model, we employed two distinct supervised learning algorithms on our training data set consisting of 14 features. Each of the employed algorithms classifies the data into one of the classes *malicious* or *benign*. To assess the prediction accuracy of the applied algorithms before employing them in the runtime detector SpyAlarm, we consider a training data set and a validation set, called dev-set. The validation set is a dedicated data set collected during the training data acquisition, but it has not been used for training the model so that the data in this set remains unseen by the model. We measure the detection accuracy of the respective algorithm against these two data sets. The two supervised learning algorithms are briefly discussed in the next paragraph.

As a binary classifier, which predicts whether a cache exploit is ongoing or not, we leverage simple Logistic Regression (LR) with sigmoid activation. Additionally, we employ a distinct deep learning algorithm by implement a neural network to achieve better validation accuracy compared to the LR-approach and refer to the more complex model as 1HL. The neural network in 1HL consists of an input layer of 14 neurons, a hidden layer with 10 neurons and an output layer of a single neuron. It uses the Gradient Descent algorithm.

7.4.2.3 *Monitoring Agent and Attack Detector*

The Monitoring Agent, as its name suggests, is a component which performs the continuous monitoring of the system. It comprises a parser and a monitoring module. First, the parser extracts the Indicators of Compromise to obtain the software events and performance counters that have to be monitored, and feed them to the monitoring module. Then, the monitoring module accesses the operating system or the hypervisor to obtain the required values and to create the current features vector required by the Attack Detector. For that the Monitoring Agent, similar to the Data Collector, requires privileged access to the system and uses the perf tool for sampling the required events and counters. The gathered data is periodically sent to the Attack Detector.

The Attack Detector leverages the model created in the Definition phase and applies it to the current data provided by the Monitoring Agent. If an attack is detected, the Attack Detector can return the process identifier (pid) of the suspect process along with the pids of the processes that have been running simultaneously with the suspect one.

7.4.2.4 *Attacks implementation*

We employ distinct implementations of side-channel and covert-channel attacks in both the Definition and the Detection phases. For the Definition Phase, we use implementations of covert-channel attacks that employ Flush+Flush, Flush+Reload and Prime+Probe as attack strategies. These implementations are used for the training of the supervised learning algorithm.

For the Detection Phase and to evaluate the reliability of the proposed detection approach, we leveraged two publicly available side-channel attack implementations[1],[2] based on [Gru+16; YF14]. These attacks are unseen by SpyAlarm. We log the pids of all the involved processes which enables the de-scheduling of the suspected process.

7.5 SPYALARM EVALUATION

In this chapter, we first cover the setup used in our evaluation, and focus on the conducted experiments. Then, the obtained results are presented.

7.5.1 *Experimental Setup*

To evaluate our detection approach, we conducted experiments on a system equipped with an Intel Skylake processor which runs Debian buster/sid operating system. As SpyAlarm is based on sampling hardware and software events, we vary the employed sampling frequency and interval to enhance the detection accuracy of SpyAlarm through deliberately choosing the sampling configuration.

To compare the detection accuracy of SpyAlarm, we consider six distinct sampling configurations. The employed sampling frequencies, controlled by F in perf

1 https://github.com/IAIK/flush_flush
2 https://github.com/defuse/flush-reload-attacks/tree/master/flush-reload

are: (i) 1000, (ii) default (4000 for most of the sampled events), and (iii) 8000. The sampling frequency denotes the average sampling rate per second. Naturally, the higher the employed frequency, the larger the number of the collected samples. At the same time, a higher sampling rate is also related to a higher performance overhead.

We also vary the sampling interval which is controlled by perf sleep, and employ sampling interval values of 0.5, 1, 2 and 3. The six sampling configurations are: (i) default frequency, sleep 0.5 (ii) default frequency, sleep 1 (iii) default frequency, sleep 2 (iv) default frequency, sleep 3 (v) frequency 8000, sleep 2, and (vi) frequency 1000, sleep 2.

The experiments follow the presented approach which consists of a Definition and a Detection phase. To cover the Definition phase, we collect training data and apply a supervised learning algorithm on this data. The supervised learning algorithm creates a model to predict the presence of an attacker. The employed algorithm leverages different training parameters which are tuned during the experiments to find the parameters configuration which leads to accurate predictions.

The training data is collected in both attacked and non-attacked environments. Moreover, we have employed two distinct supervised learning algorithms to choose the best prediction model for SpyAlarm. The reliability of SpyAlarm detection is assessed through experiments which measure SpyAlarm detection accuracy and time-to-detect. Our experiment system is a subject to diverse and representative workloads simulated using the PARSEC benchmark suite [Bie11] and the stress-ng tool [Kin19] to create a realistic scenario for the experiments.

7.5.1.1 *Towards the Selection of Training Model and Parameters*

To choose the model which allows for better predictions on unseen data, and use it for SpyAlarm, we assess how well the two implemented algorithms, LR and 1HL, described in Section 7.4.2.2, generalize from the training data set to the dev-set. As already mentioned, the dev-set is a dedicated data set, acquired during the training data acquisition but not used for the definition of the prediction model. We measure the achieved prediction accuracy of the algorithm both with respect to the training data set (i.e., training accuracy), and the accuracy of the training algorithm with respect to the dev-set (i.e., validation accuracy). For the selection of an adequate parameters configuration, we consider also the courses of the training and validation loss functions.

To ensure that we prevent the model from overfitting, we conduct experiments with dropout as a regularization technique. Overfitting occurs when the model learns the training data too well to the extent that the model ability to generalize is reduced which affects its performance on new data. In this case, the training accuracy is high, but the testing accuracy is poor. To avoid such issues, we employ dropout as a hyperparameter to train the model and conduct experiments to tune its value. The dropout is an effective regularization method which randomly ignores certain layer outputs during training to simulate a diversity of employed neural network architectures. Moreover, we leverage minibatches to prevent the algorithms from learning the links between the training data set, and experiment with minibatches of different sizes. We conduct experiments varying the training epochs length and the learning rate employed in the models.

As discussed in Section 7.5.1, we employ six distinct sampling configurations and collect a new training data set per each sampling configuration. Thereby, for each configuration we measure the training loss (i.e., the cost function with respect to the training data set) and the validation loss (i.e., the cost function with respect to the dev-set). Furthermore, we log the elapsed time and the training and validation accuracies.

With the LR-algorithm, we conducted experiments to test altogether 25 distinct parameters configurations. For none of the employed configurations the achieved training and validation accuracies turned out to be satisfactory. Therefore, the further experiments and SpyAlarm, we leverage the 1HL-algorithm.

Table 9 reveals the selected training parameters configurations per sampling configuration. The table shows also the overall number of the tested parameter configurations per sampling setting. The results presented in the table consist of the time spent on training the model in each of the cases, the achieved validation and training accuracies and the training and validation loss function values. The chosen parameter configurations encompass the number of employed training epochs, the minibatch size and the learning rate, which are discussed in the following paragraphs. In all the tested cases, it turned out that no regularization is required, as varying the dropout rates did not result in an increase of the accuracies or affected the loss functions. Therefore, the keep probability is set to 1.0 for all the chosen configurations.

Table 9: Selected Training Parameters Configurations.

Sampling config.		#Param. config.	Chosen config.			Train. acc.	Valid. acc.	Train. loss	Valid. loss	Time (sec.)
freq.	interval		epochs	MS	LR					
default	0.5	69	650	24	0.0001	0.981	0.994	0.615	0.59	343.76
default	1	41	50	24	0.0001	0.994	0.923	0.680	0.58	439.78
default	2	53	450	32	0.0001	0.955	0.955	0.609	0.62	512.9
default	3	63	650	24	0.0001	0.969	0.991	0.562	0.55	783.76
8000	2	63	450	24	0.001	0.978	0.981	0.517	0.51	617.68
1000	2	32	250	24	0.1	0.994	0.997	0.506	0.50	304.63

Although we log the time spent on training, we do not consider it when choosing the appropriate parameters configuration, as the training is conducted only once, and obtaining a high detection accuracy has a higher priority over the elapsed time. Commonly, we have chosen the parameters configuration for which the difference in the values of the training and the validation loss functions are relatively small, and the achieved validation and training accuracies are high.

The minibatch sizes we employed during the training phase are 24, 32, 64 and 128. For almost all of the sampling settings, we have chosen minibatch size of 24 for the further experiments. Only in the cases where the experiments are conducted by sampling at default rate with a sampling interval of 2, the employed minibatch size is 32, as it resulted in higher validation and training accuracies than when employing a minibatch size of 24. Increasing the minibatch size to 128 resulted in a decrease in the accuracies for all of the tested configurations.

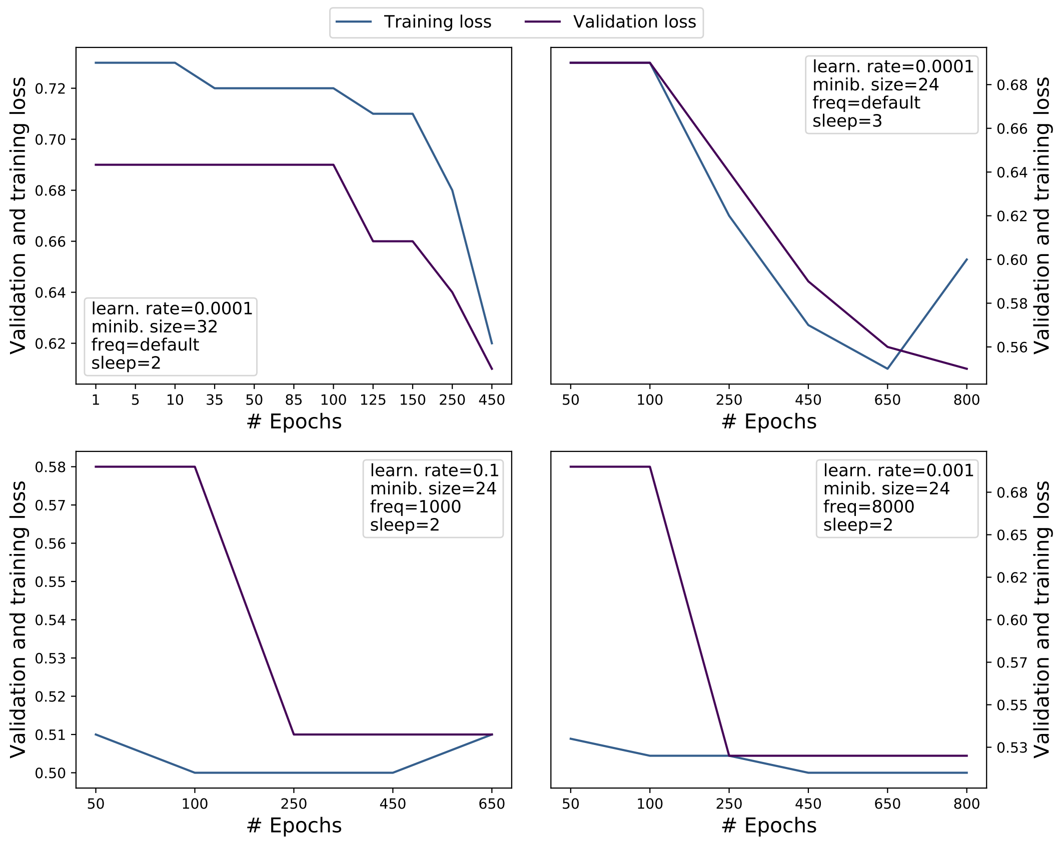


Figure 19: Epochs Tuning.

To determine the number of training epochs SpyAlarm needs, we conducted experiments by considering training epoch values between 10 and 1000. Figure 19 depicts the validation loss and the training loss functions in terms of the number of epochs for the cases when the sampling frequency remains unchanged (i.e., default), while the employed sampling interval is set to either 2 or 3 and the cases when the sampling frequency is set to 1000 and 8000, while the sampling interval is set to 2.

As visible from the plot at top left in the figure, for the chosen parameters configuration employing 450 epochs results in a small difference in the values of the validation and the loss functions of only 0.2. Furthermore, in this case, both the validation and the training accuracies are high (ca. 95%). Employing a higher number of epochs (650 and 800) with minibatch size of 24 and learning rate of 0.0001 does not result in training and testing accuracies higher than 95%. Therefore, for the sampling configuration shown at top left in Figure 19, the model used for the further experiments and the runtime detection is trained using 450 epochs.

The tuning of the number of training epochs when sampling at default rate for 3 seconds per iteration, demonstrates that by employing 650 training iterations, the values of the training and validation loss are almost equal, and the training and validation accuracies are both very high, being above 96% (cf., Table 9). The trends of the validation and training loss functions in this case are shown at top right in Figure 19.

When sampling at frequency of 1000 for 2 seconds per iteration, the chosen number of epochs is 250, whereas when the employed sampling rate is 8000, the number of epochs for the further experiments and the runtime detection is set to 450. The courses of the validation and training loss functions for these cases are shown at bottom left and bottom right in Figure 19, respectively.

To determine the learning rate during the training of the model, we conducted experiments by employing learning rates from the set of values 0.9, 0.7, 0.4, 0.1, 0.01, 0.001 and 0.0001. Figure 20 depicts the validation and the loss functions courses along with the achieved validation accuracy when sampling at frequency of 1000 (top), and at frequency of 8000 (bottom). In these cases, the learning rates are set to 0.1 and to 0.001, respectively. As shown in Figure 20, the chosen learning rate configurations are characterized by high validation accuracies and minima for the validation loss functions. Moreover, the difference in the values between the training and validation loss functions is very low when sampling at rate of 1000 and the two functions are even equal when the learning rate is set to 0.001 for the other sampling configuration (cf., bottom in Figure 20). For the rest of the tested sampling configurations, the obtained results demonstrated that setting the learning rate to 0.0001 is a reasonable choice (cf., Table 9).

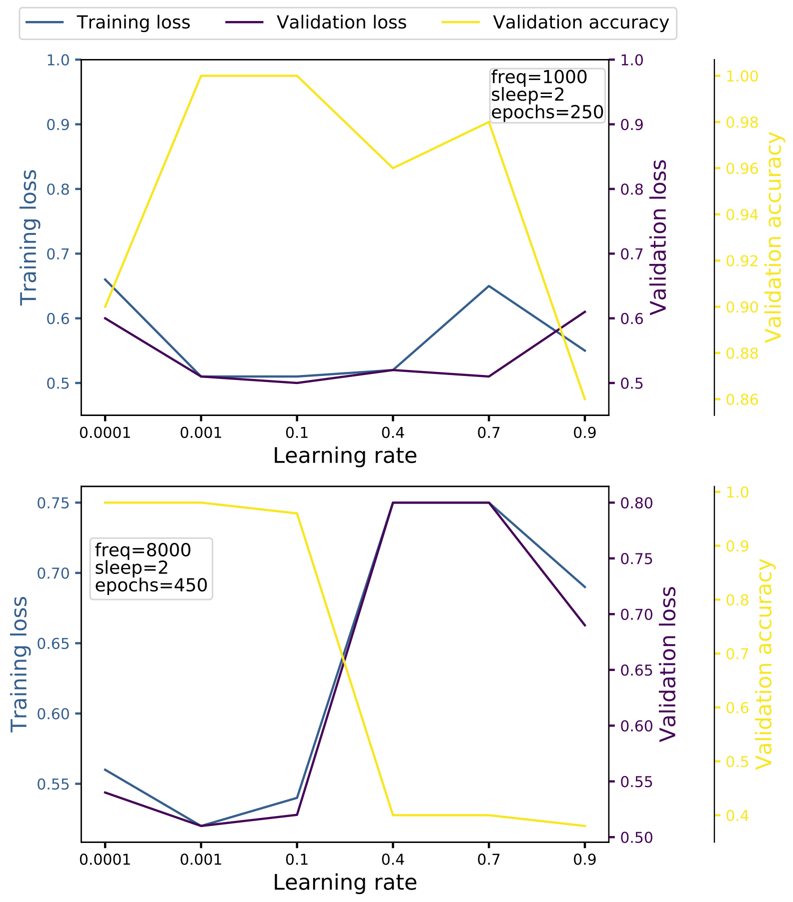


Figure 20: Learning Rate Tuning.

Altogether, we conducted experiments testing 321 parameter configurations by varying different training parameters. There were both cases of underfitting when the achieved training accuracy was too low and overfitting, when the achieved training accuracy was much higher than the validation accuracy. In some cases, the training accuracy was even less 50%. By considering the training and validation loss functions and the achieved training and validation accuracies, we were able to select adequate parameters configurations for each of the six sampling configurations.

7.5.1.2 *Accuracy Evaluation Setup*

To evaluate the accuracy provided by SpyAlarm, we computed the false positives, false negatives, true positives and true negatives the detector exhibits. A true positive is the outcome if the employed model correctly identifies an ongoing attack as such. A false positive occurs when an attack is detected in a non-attacked system. In this case, the detector has incorrectly suspected a benign process as an attacker. A false negative denotes the case when an attack has not been detected, whereas a true negative corresponds to the case when a process is correctly identified as non-malicious. In addition to the accuracy of the attack detection, the time-to-detect is measured and analyzed. We run experiments considering different sampling intervals by setting the sleep value of perf to 0.5, 1, 2 and 3, and leveraging a default frequency and frequencies of 1000 and 8000, as mentioned in the previous sections.

Our experiments are conducted on a dedicated testing set consisting of diverse non-attacker processes and two implementations of side-channel attacks that are publicly available on the Internet[3],[4] and unseen by SpyAlarm. For the rest of the chapter, the attack which uses Flush+Flush as attack strategy is referred to as FF attack and the attack which uses Flush+Reload as attack strategy is referred to as FR attack. For simulating workload, we used PARSEC dedup and blackscholes [Bie11], and stress-ng [Kin19] with 60% load per core. Overall, we consider six distinct workloads per sampling configuration, as shown in Table 10. For each of the six experimental setups, the main workload (dedup, blackscholes, stress-ng) or the involved attack is started at least 50 times. Additionally to the main workload, basic load including instant messaging application, browsers runs on the system. The gathering of the testing data and the detection take place at runtime.

Table 10: Employed Workload.

attack	workload
FF	basic load
FR	basic load
none	blackscholes + basic load
none	dedup + basic load
FF	stress-ng + basic load
FR	stress-ng + basic load

3 https://github.com/IAIK/flush_flush
4 https://github.com/defuse/flush-reload-attacks/tree/master/flush-reload

7.6 DETECTION RESULTS

In this section, we present SpyAlarm detection results to analyze empirically the utility of the proposed side-channel attack detection approach.

7.6.1 *SpyAlarm Detection Accuracy*

Table 11 summarizes the evaluation results for the considered sampling configuration setups presenting the false positives, false negatives, true positives and true negatives for each of the employed workloads defined in Table 10. As visible from the table, the optimal results are obtained when the sampling frequency remains unchanged (i.e., default) for all of the sampled events, and the sleep value is set to either 1 or 2 seconds.

Table 11: Evaluation Results – SpyAlarm Detection Accuracy.

Sampling config.		Attack setup		false pos.	false neg.	true pos.	true neg.
interval	freq.	attack	workload				
0.5 s	default	FF	basic	1	1	101	1441
		FR	basic	0	104	0	1602
		none	blackscholes + basic	6	0	0	3471
		none	dedup + basic	1	0	0	3054
		FF	stress-ng + basic	2	3	111	1968
		FR	stress-ng + basic	3	101	0	1931
1 s	default	FF	basic	194	0	119	2483
		FR	basic	161	2	105	3346
		none	blackscholes + basic	296	0	0	2723
		none	dedup + basic	276	0	0	5904
		FF	stress-ng + basic	194	0	113	2707
		FR	stress-ng + basic	161	1	104	3807
2 s	default	FF	basic	294	2	150	7516
		FR	basic	264	6	138	9091
		none	blackscholes + basic	423	0	0	11712
		none	dedup + basic	799	0	0	14525
		FF	stress-ng + basic	324	5	166	8089
		FR	stress-ng + basic	428	8	198	9604
3 s	default	FF	basic	79	3	172	9445
		FR	basic	151	119	195	12370
		none	blackscholes + basic	203	0	0	18689
		none	dedup + basic	322	0	0	17041
		FF	stress-ng + basic	137	28	216	11301
		FR	stress-ng + basic	132	111	163	12605
2 s	8000	FF	basic	395	180	128	8577
		FR	basic	315	4	128	8380
		none	blackscholes + basic	797	0	0	12634
		none	dedup + basic	898	0	0	13260
		FF	stress-ng + basic	470	216	6	9183
		FR	stress-ng + basic	423	5	172	8801
2 s	1000	FF	basic	87	3	114	5339
		FR	basic	99	110	0	5069
		none	blackscholes + basic	294	0	0	9923
		none	dedup + basic	316	0	0	9706
		FF	stress-ng + basic	99	8	109	5719
		FR	stress-ng + basic	214	228	0	12126

The results shown in Table 11 demonstrate that the leveraged FF and FR attacks are reliably detected when sampling at default frequency for 1 second per detection iteration. Both the false positive and false negative rates remain low in this case for all the workloads. Table 12 represents the confusion matrix for this sampling configuration which takes into account the results of all the workload settings. As can be noticed from the confusion matrix, the false positive rate is 5.8%, and the false negatives rate is almost 0%. This case demonstrates a good prediction accuracy and high detection reliability of SpyAlarm while exhibiting a relatively low false positive rate.

Table 12: Confusion Matrix. Sampling Interval 1 and Default Frequency.

	predicted (positive)	predicted (negative)
actual (positive)	TPR=0.9999	FNR=0.0001
actual (negative)	FPR=0.058	TNR=0.942

By increasing the sampling interval from 1 to 2, the false positive rate slightly decreases, but also the false negative rate slightly increases, as visible from the confusion matrix shown in Table 13. In this case, the false positive rate drops to 4%, and the false negative rate is 3%. The detection of the attacks remains mostly reliable, but has slightly deteriorated compared to the case when the value of the sampling interval is set to 1. As shown in Table 11, overall 7 FF attacks out of 316 have not been identified as such and 14 FR attacks out of 336 have not been detected.

Table 13: Confusion Matrix. Sampling Interval 2 and Default Frequency.

	predicted (positive)	predicted (negative)
actual (positive)	TPR=0.97	FNR=0.03
actual (negative)	FPR=0.04	TNR=0.96

By changing the frequency from default to 8000, the FF attack is not reliably detected anymore exhibiting a false negative rate of almost 98%. Thus, such sampling configuration is not suitable and is therefore not employed for the detection approach. The same applies to the case when the frequency is set to 1000. In this case, the other attack which leverages Flush+Reload attack strategy, is not detected at all exhibiting a false negative rate of 100%. In both cases the false positive rates are also higher compared to the false positive rates of the experimental setups with default sampling rate. It has to be noted, that the default sampling rate is 4000 samples per second for most of the events, but nonetheless for some of the events the sampling period is set to 1 instead, whereas when changing the default frequency, the new frequency is set for all the sampled events.

An increase or decrease of the sampling interval from 2 to 3 or from 1 to 0.5 also has a negative impact on the detection reliability and accuracy. As can be seen from Table 11, in the case of sampling interval of 3 none of the attacks is reliably detected, whereas in the case of sampling interval of 0.5 the FR attack remains uncovered for approximately half of its occurrences. Therefore, these sampling configurations are not suitable for SpyAlarm.

in our experiments, in addition to the measurements of the false positives and negatives of the detection approach, we measure also the time to the first detection of the FF attacks involved in the experimental setups with both basic load and with load caused by stress-ng (cf., Table 10). For this, we consider the configurations with sampling intervals of 1 and 2 and sample at default sampling frequency.

In our experimental setup, the attacks are not immediately killed after they have been detected, although SpyAlarm enables that through the process id of the detected attack. Therefore, the experimental results encompass attacks that have been detected more than once, as they have continued to run after their first detection. The time-to-detect measurements, however, consider only the first occurrence of the attack. Figure 21 depicts the time to the first detection of the FF attacks for the scenarios with basic load and load created through stress-ng and the scenarios with only basic load. The considered sampling interval values in the figure are 1 and 2.

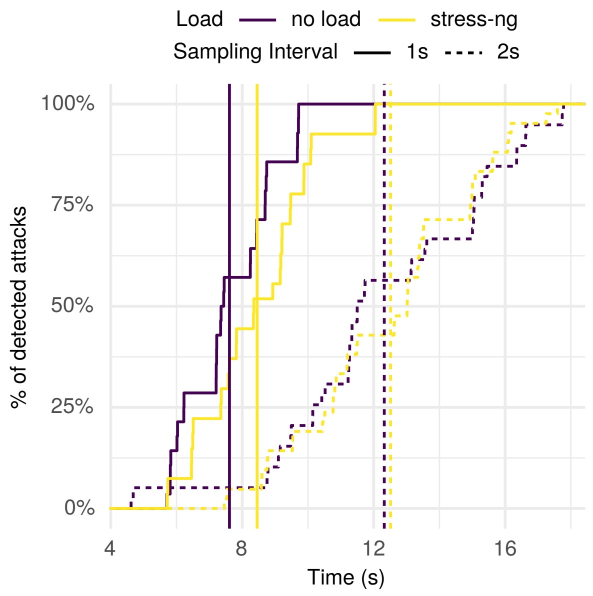


Figure 21: FF Time-to-Detect with Different Sampling Intervals and Different Load Levels.

Table 14 summarizes the obtained results. If only basic load is running on the system and the sampling interval is set to 1, the time-to-detect of the attack is 7.61 on average. The average FF detection delay is 8.45 for the same sampling configuration, but in a stressed system.

Table 14: SpyAlarm Mean-Time-To-Detect Results.

Sampling interval (s)	Load	Mean-Time-to-Detect (s)
1	basic load	7.61
1	basic load + stress-ng	8.45
2	basic load	12.3
2	basic load+ stress-ng	12.5

Logically, Figure 21 and Table 14 reveal that the detection takes longer if the sampling interval is increased to 2. This is due to the fact that each detection iteration takes place only after the runtime data has been collected by sampling the involved events and hardware performance counters. The longer the sampling interval is, the longer the respective detection iteration is delayed. If the system is under load, the FF attack is detected 12.5 seconds after it has been started on average under this sampling configuration. When the system is subject to basic load only, the mean-time-to-detect is 12.3 seconds with sampling interval of 2.

It is interesting to note that the longer the sampling interval is, the smaller the influence of the load on the detection delay is. The difference in the detection delay between the stressed system and the system with only basic load is almost 1 second when the sampling interval is set to 1, whereas it is ca. 0.2 seconds when the sampling interval is set to 2.

As seen from the presented results, the detection accuracy of the approach is high, given that the sampling rate and interval are properly chosen. The results confirm that the detection is reliable with a low false positive rate. Nonetheless, due to the sampling of hardware events, attacks might remain unrevealed if they are scheduled during a context switch of the sampling process. In our evaluation, we detect such cases, as we log the timestamps of the start of the attacker processes. We have not detected any discrepancy between the number of started attacker processes and the detection results. The few false negative cases are due to the failure to detect attacks that have been started, have been detected at least once, but have not been detected during some of the subsequent detection iterations.

7.7 CHALLENGES AND DISCUSSION

This section overviews the challenges and limitations related to SpyAlarm usage.

7.7.1 *Limited Resources and Non-determinism*

A constraint in the approach is the limited number of registers that are available for monitoring the hardware performance counters. Our experiments are conducted on an Intel Skylake-based architecture, which provides 8 performance monitoring counters and 3 fixed-function performance counters. Due to the limited number of resources for the performance counters, we consider a trade-off between the number of utilized performance counters and the accuracy of the detection, considering the guidelines described in [IK]. Additionally, our indicators of compromise combine software events with hardware events to provide a reliable detection mechanism despite the limited resources. As demonstrated by the results, the selected hardware and software events enable the reliable detection of cache-based exploits, and overcome the limited resources problem.

As discussed in [Das+19], the usage of hardware performance counters is often related to non-determinism and might result in inaccuracy of the results of up to 30%. We investigate this issue, as we maintain a log of all the started attacks during the experiments along with their respective timestamps. The results obtained in our experimental setup do not comply with this statement, as for the two cases in which the cache-based exploits are reliably detected (i.e., when sampling at default frequency for 1 or 2 seconds), all the started attacks have been detected. The few false negatives that occur result from cases in which attacks have been detected once, but have not been detected during subsequent detection iterations. This issue has not been investigated further for the rest of the sampling configurations, as we cannot distinguish whether the obtained false negatives are due to the sampling configuration and the non-determinism or due to the prediction model itself.

Nonetheless, to ensure that we minimize the possible impact of this issue, we sample at different frequencies to find the optimal sampling configuration. Apart from that, with the introduction of hardware support, precise sampling is enabled on some architectures, and can assist in reducing the non-determinism stemming from the out-of-order execution (cf., [Wea16]) if needed. Another source of non-determinism is referred to as skid which denotes the offset between the instruction that has been captured as causing an interrupt and the instruction that actually had caused the interrupt. By using PEBS, the number of interrupts is reduced which results in a constant skid of one instruction which also reduces the non-determinism involved in the detection approach. In our experimental setup, we have not noticed issues due to skid and therefore, have not used this technique.

7.7.2 *Choice of Events*

The selection of Indicators of Compromise is crucial for the success of the detection mechanism. Initially, we conducted experiments with another set of 14 events which did not result in a reliable detection mechanism. Generally, due to the limited resources, the number of monitored hardware performance counters should not exceed the number of available monitoring registers. However, the number of software events can be increased, as there exist no limitation on their usage, as long as there are available file descriptors which perf can leverage per event.

In our experiments, we have noticed that the selection of software events is very important for the detection of the cache-based exploits. As discussed in Chapter 5, the attackers of such exploits often employ explicit synchronization mechanisms using nanosleeps, or yielding the CPU after their execution. Except for that, the exploits are reliant on precise timing information. Therefore, employing a combination of software and hardware events is a good strategy for the detection of cache-based side-channel and covert-channel attacks and can enable the detection of attacks such as FF which are reported to be robust against detection approaches which rely on analysis of the cache usage.

Figure 22 confirms this observation. It visualizes the median value for each of the employed events (hardware and software) for both the malicious and benign processes. The statistics are derived from the testing data obtained during the runtime detection of the FF attack under basic load without additional stressing of the system. As visible from Figure 22 (bottom right), there is a discrepancy between the number of write_msr, read_msr and rdpmc events for malicious and benign processes in the cases for which the attack has been detected and the cases when the attack remains unrevealed (i.e., the sampling frequency is set to 8000).

For these experiments, we have set the frequency to 8000 for all the events including write_msr, read_msr and rdpmc, whereas for the cases when the frequency remains unchanged, it has not been set to the default 4000 for these three events.

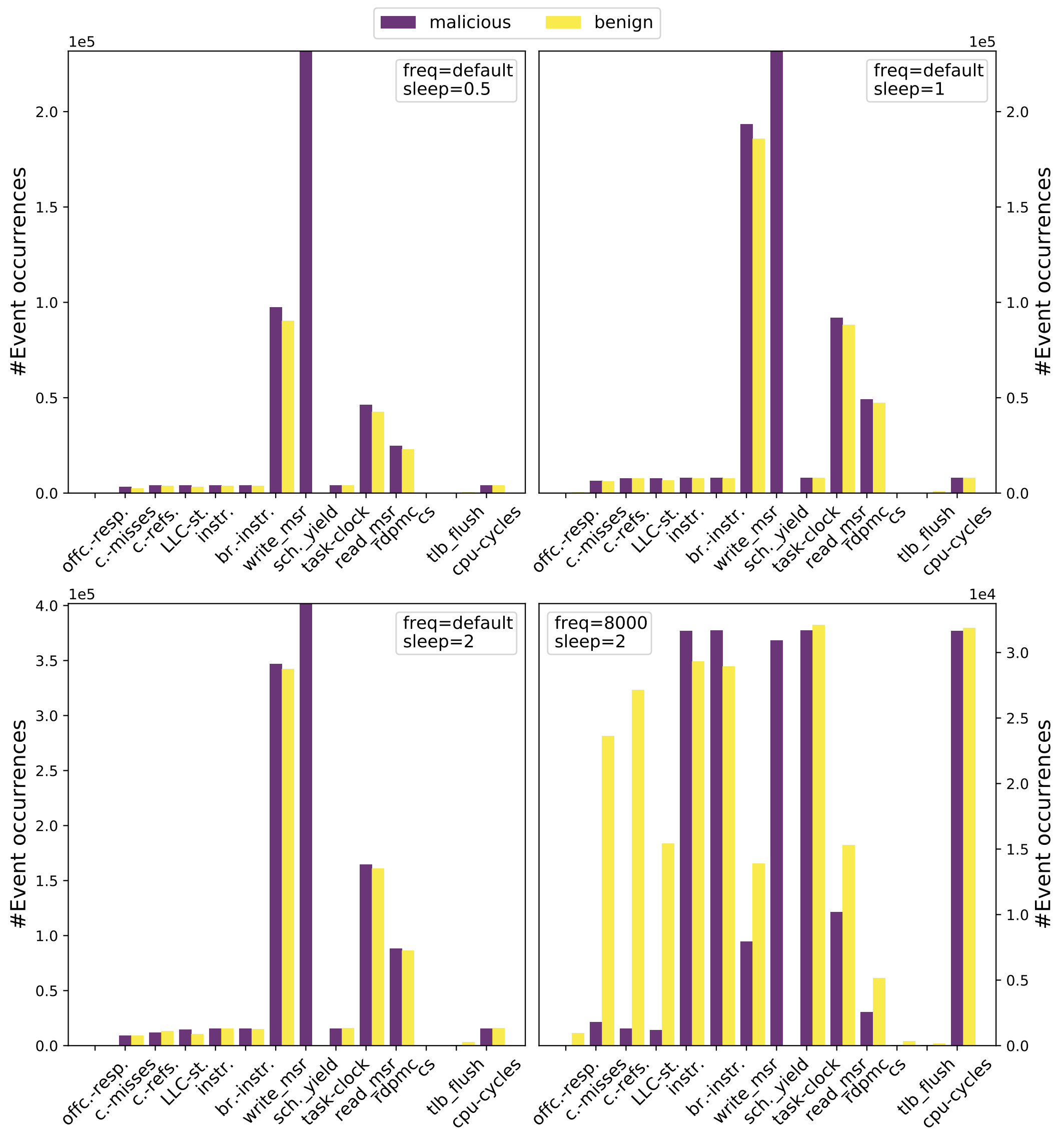


Figure 22: Events Occurrences for FF Attacker Processes and Benign Processes – Median.

Instead, by default perf sets the sample_period to 1 for the events write_msr, read_msr, syscalls:sys_enter_sched_yield, tlb:tlb_flush and rdpmc, which are all the monitored tracepoint events. The sample period denotes the number of the occurrences of an event.

Figure 23 and Figure 24 depict the median values and the mean values of the monitored events employed in the detection of the FR attack under basic load, respectively. In the case when the sampling interval is relatively small, set to 0.5 seconds (cf., Figure 23 top left and Figure 24 top left), the attack remains unrevealed possibly due to the insufficient collected samples. However, a slight difference between the statistics of the detected and not detected attacks is not obviously visible in the figures.

Table 16 gives an overview on the mean, max, median and standard deviation values for the separate events for the testing data set containing the FR attack. The Event IDs used in Table 16 are defined in Table 15. The table presents the values

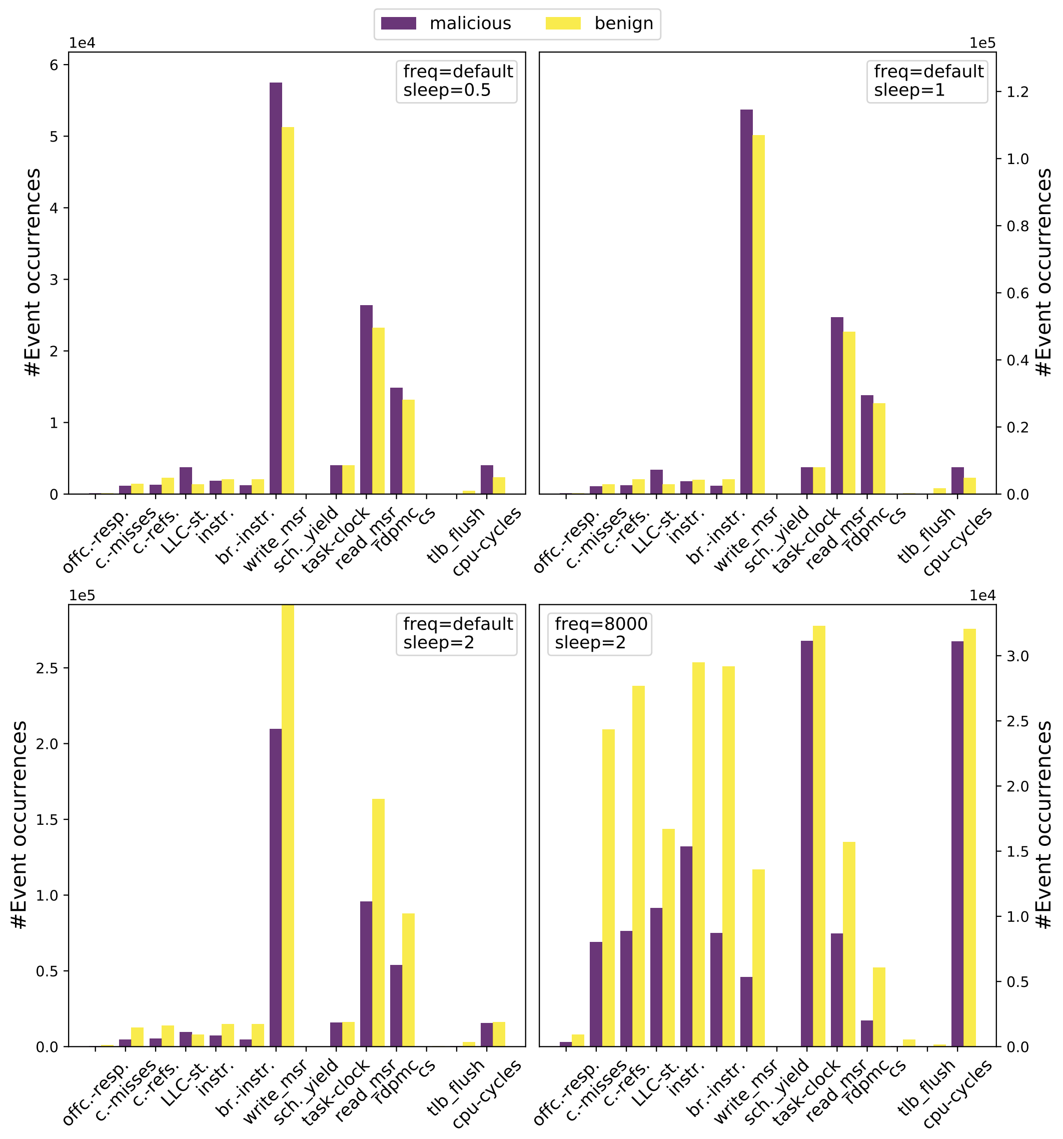


Figure 23: Events Occurrences for FR Attacker Processes and Benign Processes – Median.

for the case when the attack has not been reliably detected (sampling interval 0.5) and the values for the case when the attack has been detected (sampling interval 2). The only obvious differences in the average and median values are for the events write_msr, read_msr, rdpmc and cpu-cycles. When sampling for a shorter period of time per detection interval, the malicious processes tend to exhibit higher mean and median values for the respective events compared to the longer sampling period.

Nonetheless, it is hard or even close to infeasible to derive rules in regard to the Indicators of Compromise for an attack by considering accumulated or average values, as the involved non-malicious processes are diverse and some of them are characterized by large cache footprint whereas others are not. A one-to-one comparison between each benign process and the attack might enable deriving rules, but it is a challenging and cumbersome task due to the large set of benign processes which have to be considered.

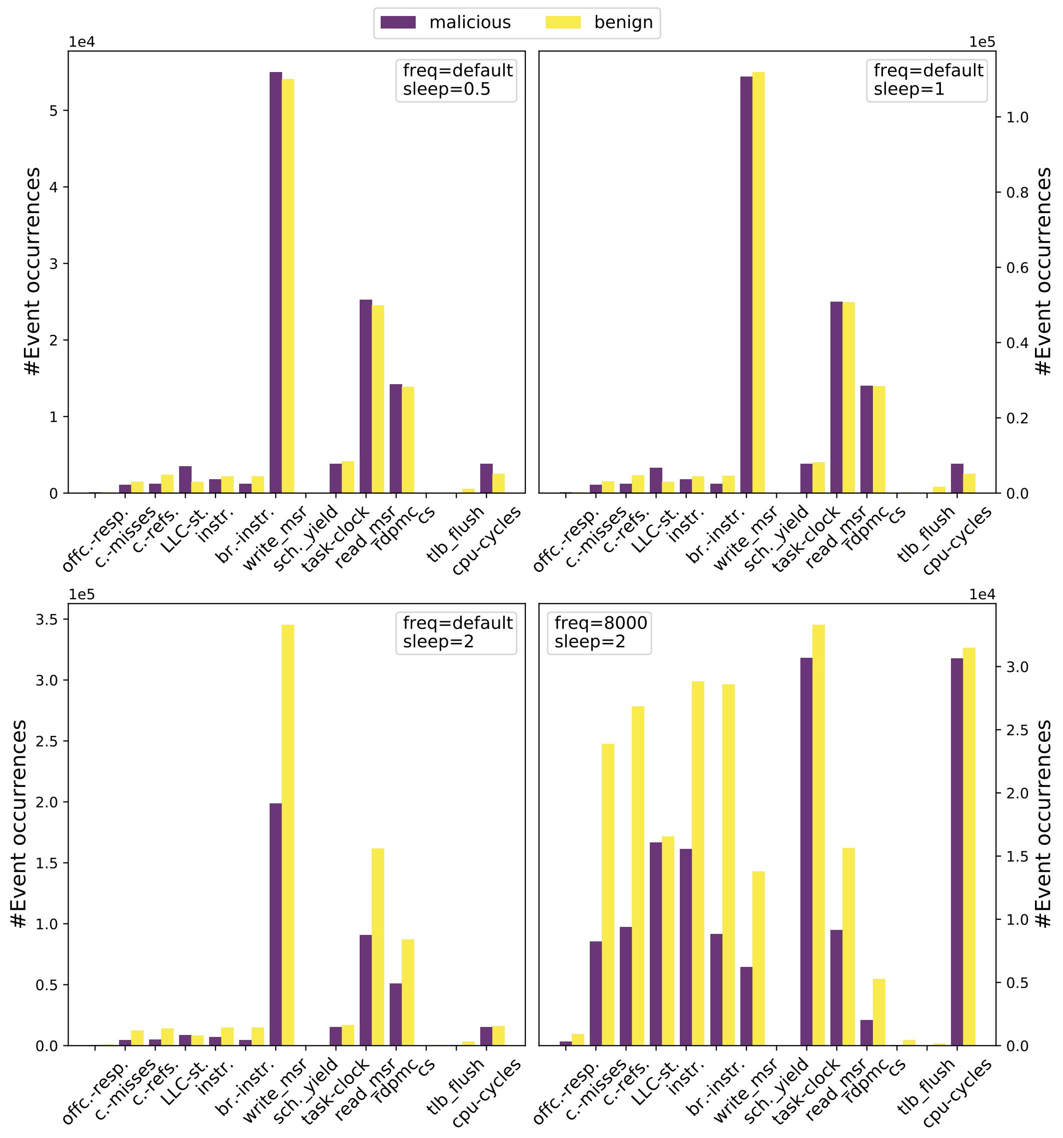


Figure 24: Events Occurrences for FR Attacker Processes and Benign Processes – Average.

Table 15: Mapping of Event ID to Event Name.

Event ID	Event name
1	offcore_response.other.any_response
2	cache-misses
3	cache-references
4	LLC-stores
5	instructions
6	branch-instructions
7	msr:write_msr
8	syscalls:sys_enter_sched_yield
9	task-clock
10	msr:read_msr
11	msr:rdpmc
12	cs
13	tlb:tlb_flush
14	cpu-cycles

Table 16: Events Usage of FR Malicious Processes and Benign Processes.

Event ID	**Process**	**sleep 0.5, freq default**				**sleep 2, freq default**			
		Median	Mean	Max	Std. Dev.	Median	Mean	Max	Std. Dev.
1	malicious	77	74	84	13.99	347	329.04	355	50.4738
1	benign	108	110	218	22.27	933	898.24	1059	165.244
2	malicious	1150	1103	1265	208.11	4769	4559.69	4947	680.199
2	benign	1399	1462	2625	229.17	12679	12390.3	15805	2244.87
3	malicious	1269	2115	1465	231.86	5244	5038.75	5681	800.672
3	benign	2249	2392	4371	424.14	13987	14020.5	17079	1583.38
4	malicious	3682	3482	3724	704.82	9590	8697.98	14650	3326.84
4	benign	1353	1461	3767	426.50	7968	8173.45	10706	1021.32
5	malicious	1880	1812	2107	343.70	7474	7146.02	7910	1131.58
5	benign	2070	2193	4061	383.13	15004	14734.4	17887	1928.48
6	malicious	1246	1211	1402	231.15	4787	4514.43	5065	1131.58
6	benign	2085	2204	4154	394.62	14940	14640	17686	1850.42
7	malicious	57502	54993	59823	10536	209610	198583	228845	34506.8
7	benign	51233	54050	89745	8073.9	348202	345444	420341	41314.1
8	malicious	0	0	0	0	0	0	0	0
8	benign	0	0	0	0	0	0	0	0
9	malicious	4007	3835	4008	722.56	15774	15097.5	16009	2285.73
9	benign	4007	4171	7912	712.94	16223	16914.8	32025	2287.39
10	malicious	26422	25265	27550	4847.4	95702	90622.6	105294	16023.6
10	benign	23252	24504	40085	3475.6	163335	161600	197908	20693.7
11	malicious	14839	14187	15448	2717.55	53888	51043.1	58938	8901.61
11	benign	13140	13909	25414	2429.27	87751	87232.3	106096	10222.5
12	malicious	2	1	6	1.65	33	43.76	143	25.859
12	benign	34	53	121	17.75	266	376.02	591	85.0547
13	malicious	2	5	47	12	418	405.26	746	164.718
13	benign	479	584	1666	267.19	3166	3301.18	4386	356.564
14	malicious	4017	3835	4022	722	15722	15069.8	15944	2275.83
14	benign	2364	2506	4965	465.80	16099	16013.3	19091	1870.65

An accumulated number of event occurrences for all the test cases with basic load for both attacks along with the number of involved processes are given in Table 17. As visible from the table, the metrics syscalls:sys_enter_sched_yield and msr:read_msr occur more often in malicious processes than in benign processes even though the number of sampled FF malicious processes is only 9273 compared to 93597 sampled non-malicious processes. This confirms that not only hardware, but also software events can be applied as an indication for a side-channel exploit.

7.7.3 *Supervised Learning Algorithm and Training Data*

As our training data is structured and consists of a relatively small features vector of 14 performance counters and events, the question that arises is whether applying a relatively complex neural network is required for the success of SpyAlarm or the low false positive rate and very low false negative rate can be achieved without leveraging a neural network with hidden layers. To address this question, we conducted experiments using simply logistic regression without employing any hidden layers. We tested altogether 25 distinct training parameters configurations,

Table 17: Event Occurrences in Malicious and Benign Processes – Accumulated Values from the Testing Data Set. Sampling Interval 2 and Default Frequency.

Event name	FF testing data set		FR testing data set	
	#Occurr. attacks	#Occurr. benign proc.	#Occurr. attacks	#Occurr. benign proc.
cpu-cycles	773449	812725	768562	816677
instructions	772141	775150	364447	751450
branch-instructions	772636	772034	230236	746639
cache-misses	521830	512552	232544	631905
cache-references	666591	708477	256976	715046
LLC-stores	715976	547199	443597	416846
msr:write_msr	17770309	17713049	10127746	17617656
syscalls:sys_enter_sched_yield	257817286	2794214	0	0
task-clock	773100	841322	769974	862655
msr:read_msr	8424648	8310927	4621753	8241607
msr:rdpmc	4514193	4475542	2603198	4448845
cs	1283	13632	2232	19177
tlb:tlb_flush	46421	149016	20668	168360
offcore_response.other.any_resp.	396	31010	16781	45810
#total processes	152	7810	144	9355

but the highest training accuracy that we achieved is less than 80% and the highest validation accuracy is 71% which is considerably lower than the training and validation accuracies of over 95% achieved when employing a neural network with one hidden layer.

It is also important to note that the quality of the training data set is significant for the detection success. A highly imbalanced training data set can deteriorate the process of proper training parameters selection by providing misleading high training accuracy if the algorithm falsely classifies everything to the larger class. Moreover, the employed training samples should be representative for all the involved classes. Only then, the foundation for an accurate and reliable detection mechanism is given.

7.8 RELATED WORK

This chapter briefly reviews the approaches for conducting and mitigating exploits using the cache as a side channel, and then discusses the state-of-the-art research related to their detection.

7.8.1 *Covert-Channel and Side-Channel Attacks and their Mitigation*

Numerous attack approaches have been proposed that exploit the cache as a side channel, e.g., in [Liu+15; Xu+11]. These exploits have been demonstrated also in Cloud scenarios [Ris+09; Zha+12] which makes them extremely relevant nowadays. As a result, performance optimization features such as memory deduplication are often considered as a threat to systems security and major providers change their services to address that threat. For example, some providers change the default settings by disabling memory deduplication [VMwa; VMwb], to eradicate specific attack scenarios such as Flush+Flush attacks.

Because of its high relevance, there are plenty of proposals on how to mitigate these attacks. Among the proposed mitigation strategies are works that suggest to eliminate the cache access patterns observation possibility of the applications so that an attacker cannot monitor them, e.g., through new secure cache designs (cf., [Dom+12; LL14; WL08]). Other mitigation approaches propose partitioning of the cache among the users using dynamic page coloring techniques which would reduce the degree of resource sharing, as described, e.g., in [Shi+11]. Among the proposals are also new prefetching schemes that add randomness to the cache access patterns [FL15] or endeavours that propose to disable the usage of fine-grained timers for certain applications such as [Hu92b; LGR17]. Static analysis techniques are also proposed to cope with the side-channel exploits, but they do not solve the security issues related to the side channels, as they are not a universal remedy and have to be applied separately to each third-party software.

Despite their merits, these various proposals either require additional or completely new hardware or add complexity and performance overhead to the existing software. Therefore, the community focuses also on proposals tailored for detecting exploits of the cache as a side channel to address the side-channel threat.

7.8.2 *Detection and Attestation Approaches*

Chen et al. proposed a framework for covert-channel attack detection termed CC-Hunter which is detailed in [CV14]. The idea behind the proposed approach is to dynamically track conflict patterns on shared processor hardware by employing an additional hardware unit. Through the proposed hardware extension, the authors of the approach study the conflicts related to the usage of a shared resource. Similarly to the approach proposed by Chen et al., we make use of the information provided by the performance counters and analyze the occurrence of specific events as an indicator for an ongoing attack, but we do not leverage any additional hardware extensions support.

Chiappetta et al. also focused on detecting side-channel attacks and proposed three distinct detection approaches described in [CSY16]. The work detailed in [CSY16] does not require changes in the operating system and detect still ongoing attacks. This approach is promising, but its application relevance is limited, as it detects explicitly attacks that leverage the Flush+Reload attack strategy which are only possible if features such as memory deduplication are in place, which is often not the case [VMwb].

Zhang et al. developed a combined signature-based and anomaly-based approach to detect side-channel attacks in [ZZL16], called CloudRadar. It leverages performance counter monitoring, similarly to the CCHunter framework proposed in [CV14], but unlike CCHunter CloudRadar correlates the cryptographic applications execution in one virtual machine with the monitoring of another virtual machine which might be performing adversarial actions to spy on the ongoing cryptographic operations in the co-located virtual machine. To detect the ongoing cryptographic operations during runtime, Zhang et al. employ a signature-based approach. The attack detection takes place once an ongoing cryptographic application has been detected. Then, an anomaly-based approach is triggered which analyzes the cache miss and hits ratios. As an indicator for an ongoing Flush+Reload-based attack the authors consider the increased cache hit rate, and the increased cache miss rate is considered as an indication for an ongoing Prime+Probe attack. The idea behind the proposed side-channel attacks detection approach CloudRadar is that it can be implemented as a Security-on-Demand solution to enhance the virtual machine security of the customers who would pay more for security guarantees.

The goal of our work is to detect not only side-channel attacks, but to uncover also covert-channel attacks. Therefore, our approach cannot rely on establishing the link between the execution of cryptographic operations and possible ongoing adversarial operations. In addition, the employed signature-based approach in [ZZL16] requires keeping up-to-date signatures database and updating it whenever a new cryptographic application which requires the interaction with the user of the application to provide the necessary data. Still, we consider this work as a base line for our research.

An architecture for security monitoring and attestation of virtual machines in a Cloud scenario is also developed by Zhang et al. and described in [ZL15]. The proposed approach enables Cloud customers to request the monitoring and attestation in regard to certain security properties of their virtual machines. The monitoring and attestation can be done periodically or upon request and the proposed architecture is flexible to include different security properties. Unlike the proposed approach, our goal is not to provide security attestation, but to uncover exploits of the cache as a side channel and to enable respective reactions at the operation system or the hypervisor level.

Zhang et al. proposed an auditing approach, called HomeAlone in [Zha+11]. HomeAlone's goal is to ensure the absence of co-resident virtual machines in the Cloud scenario. The approach targets at customers of dedicated instances in the Cloud which are commonly offered at a higher price. The proposed approach enables the customers to verify the physical isolation of their virtual machines on the dedicated instances. For this purpose, Zhang et al. leverage a side channel in an unconventional way to detect the activity of a co-resident virtual machine. Our approach does not provide security assurance but aims to enhance the systems security with regard to the side-channel exploits by providing a reliable detection mechanism.

The Bucket model was proposed by Hunger et al. in [Hun+15]. It provides a mathematical abstraction of the microarchitectural covert channels and leverages information theory to estimate the capacity of contention-based channels. The authors of the proposed model ascertain that the covert channels can achieve much higher capacity than previously suggested given an optimized clocks alignment between the receiver and the sender processes. The ultimate goal of the work described in [Hun+15] is to detect intelligent adversaries, and suggest a different attacker model than the commonly established covert-channel exploits. Our work sticks to the well-established and well-working attacker models exploiting the cache as a side channel.

7.9 CONCLUSION

The threat related to exploiting the cache as a side channel poses a security risk to current systems, which cannot be eradicated easily. To address this threat by providing means to mitigate such attacks is an absolute necessity in the security community. However, this is a challenging, non-trivial task, as the possible leakage stems from side effects of the currently leveraged hardware. Therefore, the computer architecture community strives for proposing solutions that cannot leak sensitive data through side channels. Nonetheless, such measures will not prevent the existing hardware from being insecure and leaky. Apart from that, despite their merits, novel secure hardware solutions will not necessarily be adopted by the hardware vendors in the near future, as such solutions have to be performant except for secure. Defenses at the software level are also possible, but they are typically limited to the specifity of a single manifestation of the cache-based exploits and usually can be tweaked after becoming public.

Applying SpyAlarm or similar approaches will not eradicate the threat posed by the side-channel exploits, but can enhance system security in regard to the side-channel attacks. SpyAlarm is a detection mechanism tailored to detect cache-based side-channel and covert-channel attacks. SpyAlarm can be applied as an addition to the typical intrusion detection systems to uncover explicitly side-channel exploits. Our empirical results demonstrate SpyAlarm as a reliable runtime side-channel attacks detector. As SpyAlarm is based on a combination of hardware and software events, we explore different events sampling configurations to increase the achieved detection accuracy. The empirical evaluation presented in this chapter shows that SpyAlarm achieves a high detection accuracy, and detects reliably two side-channel attacks which have not been seen by the detector while exhibiting a low false positives rate of about 5%.

The success of SpyAlarm in the detection of side-channel attacks is reliant on the fact that side-channel attacks often share common characteristics, and leave footprints in terms of their usage of software and hardware events. SpyAlarm can be potentially overwhelmed as a detection measure if the side-channel attacks adapt, but the detection approaches can adapt as well to address the new threat. For this purpose, the security researcher proposing novel attack approaches should provide the details on the newly explored attacks so that approaches such as SpyAlarm can be kept up-to-date.

The area of side-channel attacks is actively researched lately. Despite that the mitigation of such attacks turns out to be an unsolved challenging task. Ideally, this security threat has to be addressed and eradicated. Complementary to such endeavours, applying detection approaches until that time comes, will not hurt and can only enhance the security of the system. The usage of a combination of hardware and software events for the analysis of cache-based exploits is promising and has potential to enhance system security if further investigated and approaches such as SpyAlarm are kept up-to-date.

Part V

CONCLUSION

8 SUMMARY AND CONCLUSION

Side-channel and covert-channel attacks with their numerous manifestations pose a serious threat to today's computer systems, and particularly to the confidentiality of their users' data. They make use of diverse side effects of the hardware or software to derive sensitive information and compromise data confidentiality. Since the term "covert channel has been mentioned by Lampson in [Lam73] in 1973, these attacks have been in the focus of research. They have overcome periods of less popularity to reach the top of the heap today. Throughout all these years, the threat associated with side-channel exploits has been present, but during the last decade the interest in these exploits has increased tremendously mainly due to the emergence of business models offering shared resources to their customers, e.g., the Cloud. Such models presume secure isolation between the mutually untrusted users that rely on the same resources, making side-channel and covert-channel attacks more relevant. In the co-location context, side-channel exploits can break the presumed isolation and lead to a violation of customers' confidentiality. A number of attacks leaking information through different system components by involving analysis of hardware side effects have been published and have become quite popular, also in the Cloud context.

In the focus of this thesis are the side-channel and covert-channel attacks, which leak secret data by exploiting the side effects of the cache memory. The cache has been present as a convenient and powerful leakage channel throughout all the years since the first mention of the term "covert channel". The main reason for this development is that the cache organization, which is the root cause for emanating side effects, has not changed rapidly over the past 20 years and the side-channel problem, though being known, has not been tackled. The existence of cache-based side-channel exploits in the context of systems offering shared resources to their users calls for rigorous methods to cope with this confidentiality threat. Only then, the trust of the customers in the secure isolation of their data in the context of shared resource usage can be justified. Whether eradicating the cache-based exploits by the computer architecture community in the course of the last two decades has been impossible due to cost or performance overhead, or this community has proposed solutions which turned out to be impractical for the vendors, and therefore have not been applied in practice, remains an open question.

Nowadays, both the security and computer architecture communities strive to find practical solutions against the side-channel and covert-channel attacks, but this is a challenging task. Mitigation approaches at the software level are proposed to cope with the problem, but they commonly tackle a specific implementation, and are not resistant against new versions of attacks, or introduce high performance penalties. Moreover, until recently, the lack of different publicly available attack implementations has impeded the development of more abstract or general preventive measures. Fortunately, the last five years have seen a positive development in this regard, and publicly available versions of the cache-based attacks can be accessed and used. This is a necessity when trying to tackle the problem, and is an important improvement in the security community.

This thesis aims to provide a basis for enhancing system's security, by exploring the relationship between the execution environment and the exploitability of the cache. The target is to investigate the issues related to the cache-based exploits from three points of view: First, the thesis studies the basis for establishing the relationship between the execution environment and the cache-based exploits from a theoretical perspective. Second, certain characteristics of the execution environment and their relationship to the success of the side-channel and covert-channel cache-based attacks are analyzed empirically. Then, the thesis focuses on applying the acquired information to develop a detection strategy tailored to uncover adversaries exploiting the cache as a leakage channel. In summary, this thesis investigated the following research questions along with the resulting contributions.

Research Question 1 (RQ1): Can we classify and model side-channel exploits considering the impact of the execution environment on their feasibility?

The diversity of side channels along with their numerous manifestations pose a challenge to their analysis, and make it infeasible to cope with the security threat stemming from their exploitation. In fact, there is no generic solution which can preclude an adversary from exploiting certain side effects of the underlying system. This is aggravated by the variety of side-channel exploits, which are not necessarily well-documented, making it hard to consider all the details involved in an attack and derive solutions to combat the problem. This fact reinforces the necessity of a classification that points out the differences and the similarities between the diverse side-channel exploits, and encompasses the relationship between the exploits and the execution environment in which they are deployed. Such a classification can enable the feasibility analysis by providing the means to consider the effect of the execution environment on the exploitation of a specific side channel.

Considering the effect of the properties of the execution environment on the individual attacks can also enhance awareness regarding the feasibility of the attacks in a given context. In fact, not all the reported side-channel attacks are feasible under any conditions. Properties of the execution environment such as CPU scheduling can affect the success of the attack, and, therefore, these properties should be investigated. The lack of models that consider not only the way the adversary uses the measurable side effects, but also the interplay between the execution environment and the adversary should be considered.

Contribution 1 (C1): A classification approach of side-channel exploits and a side-channel attack model including execution environment factors

To enable systematic reasoning in regard to the large variety of side-channel exploits, this thesis proposed an extensive classification approach which incorporates the interplay between the execution environment and the individual strategies exploiting measurable side effects. The proposed classification approach focuses on the assumptions and limitations of the approaches which can affect the feasibility and deployment of the exploits under different conditions.

In addition, the thesis discusses the feasibility factors that are specific to the virtualized environment which is among the underlying and enabling technologies for the Cloud computing model and is especially threatened by adversaries exploiting the side effects of the cache.

Among the factors affecting the feasibility of these exploits is the CPU scheduling. Its influence is especially relevant for core-based side-channel and covert-channel exploits. Therefore, the thesis proposes an information leakage model, called InfoLeak, which incorporates the effect of the CPU scheduling on the exploits of the core-private cache. The first contribution of the thesis facilitates systematic reasoning about cache-based exploits, and enables the assessment of their feasibility given the execution environment properties.

Research Question 2 (RQ2): What is the relationship between execution environment properties and the success and feasibility of side-channel attacks?

Among the approaches that have been proposed in the security community to combat side-channel attacks are approaches that consider properties of the execution environment. The scheduling of the adversary processes, in particular, also has an impact on the attack's success.

The impact of the scheduling approach itself is hard to measure because of the involved non-determinism. However, the adversaries abusing the core-private caches often rely on a specific scheduling configuration, or certain scheduling decisions, to be able to collect sufficient observations on the cache usage of the victim in the context of a side-channel attack or on the cooperating attacker in the context of a covert-channel attack, and be able to derive sensitive information out of the collected data. Therefore, the relationship between the employed scheduling configuration and its effect on the success of the core-private cache exploits should be studied. Moreover, the scheduling-related traces an adversary can leave when striving for collecting fine-grained observations over the cache usage have to be investigated.

Contribution 2 (C2): Empirical evaluation of the effect of the scheduling on the side-channel attacks and their possible traces

As the first contribution of the thesis points out, the context of the execution environment plays a significant role in the success of the side-channel attacks. Of special interest is the CPU scheduling which affects in particular the core-private cache exploits mainly due to the small size of the core-private cache.

To assess the exploitability of the execution environment in terms of the scheduling configuration, this thesis presented an empirical study in a virtualized environment that investigates the relationship between the scheduling configuration and the success of a core-private cache exploit. For this purpose, a variety of hypervisor scheduling parameters has been considered. Although, this relationship depends on the characteristics of the employed attack, and the means the adversary leverages, the results demonstrate that certain scheduling parameters are vital to core-private cache exploits.

In addition, the thesis focused on possible indicators of side-channel and covert-channel adversaries, and proposed metrics which can be used post-mortem to investigate the available logs of the operating system for side-channel attack indications. The conducted experiments demonstrated that there is a correlation between the successful transmission of information in the context of a covert-channel attack exploiting the core-private cache and the proposed indicators. However, due to the involved overhead, applying these indicators at runtime is not feasible, and they cannot provide definite statements regarding ongoing attacks. Nevertheless, the conducted experiments demonstrate that there are characteristics of the cache-based exploits which can be explored to enable their detection.

Research Question 3 (RQ3): Can side-channel attacks be reliably detected?

As side-channel and covert-channel attacks do not leave any definite traces on the system, and fall out of the scope of the traditional intrusion detection systems relying on security policies and access rights, these attacks are hard to uncover. However, the results presented as the second contribution of the thesis suggest that the information contained in the system logs can be investigated post-mortem for indicators for side-channel exploits. Although the presented results are not sufficient on their own for building a detection mechanism, they suggest the possibility to search for traces of cache-based exploits in the system logs.

Currently, the eradication of cache-based side-channel attacks seems to be an infeasible task. However, given the relevance of these exploits as a real-world threat to the data confidentiality, the need for a reliable detection mechanism of cache-based exploits only increases. Ideally, the existing approaches have to be complemented with novel proposals, developed to encounter particularly the side-channel issues.

Contribution 3 (C3): A reliable side-channel attack detection approach using performance counters and software events

As a third contribution, the thesis proposed SpyAlarm, a reliable detection mechanism which is tailored to detect cache-based covert-channel and side-channel attacks at runtime. The evaluation results demonstrate that the proposed approach which is based on the usage of a combination of performance counter and software events can reliably detect cache-based attacks while exhibiting a low false positives rate.

SpyAlarm leverages both hardware and software events to overcome the limited monitoring resources and to enhance the detection accuracy of the approach. As part of this contribution, the architecture of SpyAlarm is defined. The approach is evaluated by leveraging a prototype implementation and testing it against two publicly available attacks, unseen by the detector. As SpyAlarm leverages hardware and software events, the sampling frequency and interval are tuned to obtain accurate detection results. In addition to that, two supervised learning algorithms are considered and a number of training parameters are tuned to achieve prediction results characterized by low false positives and false negatives rates. Approaches such as SpyAlarm will not eradicate the side-channel threat, but can enhance the security of systems against this threat, and can enable triggering further actions to contain the damage caused by an attacker.

This thesis confirms that side-channel attacks abusing the cache side effects are hard to combat. They have been a known threat for a long time, and there is no clear indication that these exploits will be eradicated soon, although they pose a serious threat to many widely deployed and used systems. Moreover, side-channel attacks exhibit specific characteristics not comparable with the characteristics of other attack types, as they abuse components such as the cache which do not obey to particular access rights or policies. In addition, cache-based exploits are partly caused by important optimization solutions which have resulted in a more performant hardware. Therefore, coping with these exploits remains a challenging task despite the endeavours of the security community. Moreover, it is likely that further such vulnerabilities will be discovered, as well as new and novel side channels to leak information will be proposed.

However, despite the anticipation that these attacks will remain present for the near future, there are positive developments in the community which, although very slowly, move forward towards solving the problem. Among them is the serious and sustained research that focus on mitigation strategies. Also, more and more researchers publish diverse implementations of side-channel attacks. Possibly, an up-to-date database with attack's implementations can be beneficial for approaches such as SpyAlarm which study the details of these attacks to be able to detect them. Keeping a detection approach up-to-date with the new attacks will definitely improve system's security until the underlying architectural issues are fixed to be both secure and performant. The contributions presented in this thesis constitute mechanisms to alleviate the side-channel problem until it is solved, and aid the development of new solutions to enhance the security of current systems.

LIST OF TABLES

LIST OF FIGURES

BIBLIOGRAPHY

[AEW09] Andrea Arcangeli, Izik Eidus, and Chris Wright. "Increasing memory density by using KSM". In: *Proc. of the Linux Symposium*. 2009, pp. 19–28.

[Aba+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/. Software available from tensorflow.org. 2015.

[Agr+03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. "The EM Side—Channel(s)". In: *Proc. of the 4th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2002*. Ed. by Burton S. Kaliski, Çetin K. Koç, and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 29–45.

[Ald+19] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. "Port Contention for Fun and Profit". In: *Proc. of the IEEE Symposium on Security & Privacy, S&P 2019*. 2019, pp. 870–887.

[Ama14] Amazon Web Services. *Amazon Virtual Private Cloud User Guide- Dedicated Instances*. http://awsdocs.s3.amazonaws.com/VPC/latest/vpc-ug.pdf. 2014.

[And+06] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov. "Cryptographic Processors-A Survey". In: *Proc. of the IEEE* 94.2 (2006), pp. 357–369. DOI: 10.1109/JPROC.2005.862423.

[Ari19] Pamela Arimoto. *Preparing for 3DES Retirement - O365 Skype for Business*. Microsoft. https://techcommunity.microsoft.com/t5/Skype-for-Business-Blog/Preparing-for-3DES-Retirement-O365-Skype-for-Business/ba-p/440968. Last accessed on 13.08.2019. 2019.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. "On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN". In: *Proc. of the Conference on Computer and Communications Security*. CCS '16. New York, NY, USA: ACM, 2016, pp. 456–467. DOI: 10.1145/2976749.2978423.

[BR19] Elaine Barker and Allen Roginsky. *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Tech. rep. 800-131A. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf, Last accessed on 11.09.2019. National Institute of Standards and Technology (NIST), 2019.

[Bau+13] A. Bauer, E. Jaulmes, E. Prouff, and J. Wild. "Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations". In: *Proceedings of Topics in Cryptology – CT-RSA 2013*. Ed. by Ed Dawson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–17.

[Ber05] Daniel J. Bernstein. *Cache-timing attacks on AES*. Tech. rep. https://cr.yp.to/antiforgery/cachetiming-20050414.pdf, Last accessed on 11.09.2019. 2005.

[Bie11] Christian Bienia. "Benchmarking Modern Multiprocessors". PhD thesis. Princeton University, 2011.

[Bir+18] Henk Birkholz, Christoph Krauß, Maria Zhdanova, Don Kuzhiyelil, Tolga Arul, Markus Heinrich, Stefan Katzenbeisser, Neeraj Suri, Tsvetoslava Vateva-Gurova, and Christian Schlehuber. "A Reference Architecture for Integrating Safety and Security Applications on Railway Command and Control Systems". In: *International Workshop on MILS: Architecture and Assurance for Secure Systems, Luxembourg, 25.06.2018*. Zenodo, 2018. DOI: 10.5281/zenodo.1314095.

[Bos+16] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *Proceedings of Symposium on Security and Privacy, SP 2016*. 2016, pp. 987–1004. DOI: 10.1109/SP.2016.63.

[Bra+17] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. "Software Grand Exposure: SGX Cache Attacks Are Practical". In: *Proc. of the USENIX Workshop on Offensive Technologies (WOOT 17)*. Vancouver, BC: USENIX Association, 2017.

[CGV07] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. "Comparison of the Three CPU Schedulers in Xen". In: *SIGMETRICS Perform. Eval. Rev.* 35.2 (Sept. 2007), pp. 42–51. DOI: 10.1145/1330555.1330556.

[CSY16] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. "Real Time Detection of Cache-based Side-channel Attacks Using Hardware Performance Counters". In: *Appl. Soft Comput.* 49.C (Dec. 2016), pp. 1162–1174. DOI: 10.1016/j.asoc.2016.09.014.

[CV14] J. Chen and G. Venkataramani. "CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware". In: *Proc. of the Annual International Symposium on Microarchitecture (MICRO 14)*. 2014, pp. 216–228. DOI: 10.1109/MICRO.2014.42.

[CWL11] Chao-Rui Chang, Jan-Jan Wu, and Pangfeng Liu. "An Empirical Study on Memory Sharing of Virtual Machines for Server Consolidation". In: *Proc. of the International Symposium on Parallel and Distributed Processing with Applications*. ISPA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 244–249. DOI: 10.1109/ISPA.2011.31.

[Car+04] Vincent Carlier, Herve Chabanne, Emmanuelle Dottax, and Herve Pelletier. *Electromagnetic Side Channels of an FPGA Implementation of AES*. IACR Cryptology ePrint Archive http://eprint.iacr.org/2004/145. 2004.

[Ccp] *Assembly Bill No. 375*. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375. Legislative Counsel Bureau State of California, June 29, 2018.

[Ccs] *ACM CCS 2018: Program*. https://www.sigsac.org/ccs/CCS2018/program/. Last accessed on 11.09.2019. Toronto, Canada, 2018.

[Cla+10] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. "Horizontal Correlation Analysis on Exponentiation". In: *International Conference on Information, Communications and Signal Processing*. LNCS. Springer-Verlag, 2010.

[DC99] Kenneth J. Duda and David R. Cheriton. "Borrowed-virtual-time (BVT) Scheduling: Supporting Latency-sensitive Threads in a General-purpose Scheduler". In: *Proc. of the 17th Symposium on Operating Systems Principles*. SOSP '99. New York, NY, USA: ACM, 1999, pp. 261–276. DOI: 10.1145/319151.319169.

[Das+19] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose. "SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security". In: *Proc. of the IEEE Symposium on Security & Privacy)*. IEEE Computer Society, 2019, pp. 362–380. DOI: 10.1109/SP.2019.00021.

[Dem+18] Kubilay Demir, Hatem Ismail, Tsvetoslava Vateva-Gurova, and Neeraj Suri. "Securing the Cloud-Assisted Smart Grid". In: *International Journal of Critical Infrastructure Protection (IJCIP)* 23 (2018), pp. 100–111. DOI: 10.1016/j.ijcip.2018.08.004.

[Dep85] Department of Defense. *Trusted Computer System Evaluation Criteria*. Tech. rep. DoD 5200.28-STD. Also known as the "Orange Book". National Computer Security Center, Ft. Meade, MD 20755, 1985.

[Dom+12] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. "Non-monopolizable Caches: Low-complexity Mitigation of Cache Side Channel Attacks". In: *ACM Trans. Archit. Code Optim.* 8.4 (Jan. 2012), 35:1–35:21.

[FDF05] R. Figueiredo, P. A. Dinda, and J. Fortes. "Guest Editors' Introduction: Resource Virtualization Renaissance". In: *Computer* 38.5 (2005), pp. 28–31.

[FL15] Adi Fuchs and Ruby B. Lee. "Disruptive Prefetching: Impact on Side-channel Attacks and Cache Designs". In: *Proc. of the 8th ACM International Systems and Storage Conference*. SYSTOR '15. New York, NY, USA: ACM, 2015, 14:1–14:12. DOI: 10.1145/2757667.2757672.

[Fea+02] Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. "Infranet: Circumventing Web Censorship and Surveillance". In: *Proceedings of the USENIX Security Symposium (USENIX Security '02)*. San Francisco, CA, 2002.

[Fre+17] Sylvain Frey, Awais Rashid, Pauline Anthonysamy, Maria Albuquerque, and Syed Asad Ali Naqvi. "The Good, the Bad and the Ugly: A Study of Security Decisions in a Cyber-Physical Systems Game". In: *IEEE Transactions on Software Engineering* PP (Dec. 2017), pp. 1–1. DOI: 10.1109/TSE.2017.2782813.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. "RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis". In: *Proc. of the 34th Annual Cryptology Conference on Advances in Cryptology – CRYPTO 2014, Santa Barbara, CA, USA, August 17-21, 2014*. Ed. by Juan A. Garay and Rosario Gennaro. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 444–461.

[Gar+12] Jesus Luna Garcia, Hamza Ghani, Tsvetoslava Vateva, and Neeraj Suri. "Quantitative Assessment of Cloud Security Level Agreements - A Case Study". In: *Proc. of the International Conference on Security and Cryptography - Volume 1: SECRYPT 2012, Rome, Italy*. 2012, pp. 64–73.

[Gar+13] Jesus Luna Garcia, Tsvetoslava Vateva-Gurova, Neeraj Suri, Massimiliano Rak, and Loredana Liccardo. "Negotiating and Brokering Cloud Resources based on Security Level Agreements". In: *Proc. of the 3rd International Conference on Cloud Computing and Services Science - Volume 1: CloudSecGov, (CLOSER 2013)*. SciTePress, 2013, pp. 533–541.

[Gar18] Gartner, Inc. *Gartner Survey Says Cloud Computing Remains Top Emerging Business Risk*. https://www.gartner.com/en/newsroom/press-releases/2018-08-15-gartner-says-cloud-computing-remains-top-emerging-business-risk. Last accessed on 10.07.2019. 2018.

[Gar19] Gartner, Inc. *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019*. https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g/. Last accessed on 10.09.2019. 2019.

[Gdp] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=EN. European Commission, Apr. 27, 2016.

[Gen+19] Daniel Genkin, Mihir Pattani, Roei Schuster, and Eran Tromer. "Synesthesia: Detecting Screen Content via Remote Acoustic Side Channels". In: *Proc. of the IEEE Symposium on Security & Privacy, S&P 2019*. 2019, pp. 853–869. DOI: 10.1109/SP.2019.00074.

[Gra+17] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Bos, and Cristiano Giuffrida. "ASLR on the Line: Practical Cache Attacks on the MMU". In: *Proc. of the 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. Feb. 2017.

[Gra91] J. W. Gray. "Toward a mathematical foundation for information flow security". In: *Proc. of the Symposium on Security and Privacy*. 1991, pp. 21–34.

[Gra93] J. W. Gray. "On analyzing the bus-contention channel under fuzzy time". In: *Proc. of Computer Security Foundations Workshop*. IEEE, 1993, pp. 3–9.

[Gru+16] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. "Flush+Flush: A Fast and Stealthy Cache Attack". In: *Proc. of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*. DIMVA 2016. San Sebastian, Spain: Springer-Verlag, 2016, pp. 279–299. DOI: 10.1007/978-3-319-40667-1_14.

[Gup+08] Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat. "Difference Engine: Harnessing Memory Redundancy in Virtual Machines". In: *Proc. of the USENIX Conference on Operating Systems Design and Implementation*. OSDI'08. San Diego, California: USENIX Association, 2008, pp. 309–322.

[HIM14] HIMMS Analytics. *2014 HIMMS Analytics Cloud Survey*. https://www.himss.org/file/1308371/download?token=CBkkly5K. Last accessed on 07.07.2018. 2014.

[Hea18] Nick Heath. *'Moore's Law is dead': Three predictions about the computers of tomorrow*. https://www.techrepublic.com/article/moores-law-is-dead-three-predictions-about-the-computers-of-tomorrow/. Last accessed on 13.08.2019. 2018.

[Hei+19] Markus Heinrich, Tsvetoslava Vateva-Gurova, Tolga Arul, Stefan Katzenbeisser, Neeraj Suri, Henk Birkholz, Andreas Fuchs, Christoph Krauß, Maria Zhdanova, Don Kuzhiyelil, Sergey Tverdyshev, and Christian Schlehuber. "Security Requirements Engineering in Safety-Critical Railway Signalling Networks". In: *Security and Communication Networks* 2019 (2019). DOI: 10.1155/2019/8348925.

[Hla+11] H. Hlavacs, T. Treutner, J. P. Gelas, L. Lefevre, and A. C. Orgerie. "Energy Consumption Side-Channel Attack at Virtual Machines in a Cloud". In: *Proc. of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, DASC 2011*. 2011, pp. 605–612. DOI: 10.1109/DASC.2011.110.

[Hu92a] W. M. Hu. "Lattice Scheduling and Covert Channels". In: *Proc. of the Computer Society Symposium on Research in Security and Privacy*. 1992, pp. 52–61. DOI: 10.1109/RISP.1992.213271.

[Hu92b] W. M. Hu. "Reducing timing channels with fuzzy time". In: *J. Comput. Secur.* 1.3-4 (1992), pp. 233–254.

[Hun+15] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari. "Understanding contention-based channels and using them for defense". In: *Proc. of the 21st International Symposium on High Performance Computer Architecture (HPCA)*. 2015, pp. 639–650.

[IK] Peggy Ireland and Shihjong Kuo. *Performance Monitoring Unit Sharing Guide*. https://software.intel.com/sites/default/files/m/2/5/1/b/4/30388-PMU-Sharing-Guidelines.pdf. Last accessed on 02.08.2019.

[Infa] Information Technology Laboratory. *Computer Security Resource Center*. https://csrc.nist.gov/glossary/term/DATA-CONFIDENTIALITY. Last accessed on 02.09.2019.

[Infb] Information Technology Laboratory. *Computer Security Resource Center*. https://csrc.nist.gov/glossary/term/Side_Channel-Attack. Last accessed on 05.09.2019.

[Infc] Information Technology Laboratory. *Computer Security Resource Center*. https://csrc.nist.gov/glossary/term/covert-channel. Last accessed on 07.09.2019.

[Int] *Intel 64 and IA-32 Architectures Optimization Reference Manual*. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf. Last accessed on 12.08.2019. 2016.

[Ira+14] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. "Wait a Minute! A fast, Cross-VM Attack on AES". In: *Proc. of the International Symposium on Research in Attacks, Intrusions, and Defenses, RAID 2014*. 2014, pp. 299–319.

[KA14] Pankaj Kadam and Nilesh Alone. "Review on KVM Hypervisor". In: *International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878* 3.4 (2014), pp. 54–59.

[KB07] B. Köpf and D. Basin. "An Information-theoretic Model for Adaptive Side-channel Attacks". In: *Proc. of the 14th ACM Conference on Computer and Communications Security*. CCS '07. 2007, pp. 286–296. DOI: 10.1145/1315245.1315282.

[KPMR12] T. Kim, M. Peinado, and G. Mainar-Ruiz. "STEALTHMEM: System-level Protection Against Cache-based Side Channel Attacks in the Cloud". In: *Proc. of the USENIX Security Symposium (USENIX Security 12)*. 2012, pp. 189–204.

[Kem02] R. A. Kemmerer. "A Practical Approach to Identifying Storage and Timing Channels: Twenty Years Later". In: *Proc. of the 18th Annual Computer Security Applications Conference*. ACSAC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 109–118.

[Ker13] Michael Kerrisk. *The Linux man-pages project*. http://man7.org/linux/man-pages/man7/sched.7.html. Last accessed on 02.07.2018. 2013.

[Kin19] Colin King. *stress-ng - a tool to load and stress a computer system*. https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html. Last accessed on 02.06.2019. 2019.

[Koc96] P. C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Proc. of the 16th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '96. 1996, pp. 104–113.

[LGR13] P. Li, D. Gao, and M. K. Reiter. "Mitigating access-driven timing channels in clouds using StopWatch". In: *Proc. of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2013, pp. 1–12. DOI: 10.1109/DSN.2013.6575299.

[LGR17] Weijie Liu, Debin Gao, and Michael K. Reiter. "On-Demand Time Blurring to Support Side-Channel Defense". In: *Proc. the 22nd European Symposium on Research in Computer Security (ESORICS) 2017*. 2017, pp. 210–228. DOI: 10.1007/978-3-319-66399-9_12.

[LL14] Fangfei Liu and Ruby B. Lee. "Random Fill Cache Architecture". In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-47. Washington, DC, USA: IEEE Computer Society, 2014, pp. 203–215.

[Lam73] Butler W. Lampson. "A Note on the Confinement Problem". In: *Commun. ACM* 16.10 (1973), pp. 613–615.

[Liu+15] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. "Last-Level Cache Side-Channel Attacks are Practical". In: *Proc. of the Symposium on Security and Privacy*. 2015, pp. 605–622. DOI: 10.1109/SP.2015.43.

[Lun+13] Jesus Luna, Tsvetoslava Vateva-Gurova, Neeraj Suri, Massimiliano Rak, and Alessandra De Benedictis. "SecLA-Based Negotiation and Brokering of Cloud Resources". In: *Cloud Computing and Services Science– Third International Conference, CLOSER 2013, Aachen, Germany, May 8-10, 2013, Revised Selected Papers*. Springer International Publishing, 2013, pp. 1–18.

[M. 09] M. Jones. *Inside the Linux 2.6 Completely Fair Scheduler*. http://www.ibm.com/developerworks/library/l-completely-fair-scheduler/. IBM. developerWorks. Last accessed on 28.10.2019. 2009.

[MDS12] R. Martin, J. Demme, and S. Sethumadhavan. "TimeWarp: Rethinking Timekeeping and Performance Monitoring Mechanisms to Mitigate Side-channel Attacks." In: *Proc. of the 39th Annual International Symposium on Computer Architecture*. ISCA '12. 2012, pp. 118–129.

[MDS99] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. "Investigations of Power Analysis Attacks on Smartcards". In: *Proc. of the USENIX Workshop on Smartcard Technology*. WOST'99. Berkeley, CA, USA: USENIX Association, 1999, pp. 17–17.

[MKS12] Keaton Mowery, Sriram Keelveedhi, and Hovav Shacham. "Are AES x86 Cache Timing Attacks Still Feasible?" In: *Proc. of the 2012 ACM Workshop on Cloud Computing Security Workshop*. CCSW '12. New York, NY, USA: ACM, 2012, pp. 19–24. DOI: 10.1145/2381913.2381917.

[MM94] I.S. Moskowitz and A.R. Miller. "Simple Timing Channels". In: *Proc. of the Computer Symposium on Research in Security and Privacy*. 1994, pp. 56–64.

[Man+18] Salman Manzoor, Tsvetoslava Vateva-Gurova, Rubén Trapero, and Neeraj Suri. “Threat Modeling the Cloud: An Ontology Based Approach”. In: *Proc. of Information and Operational Technology Security Systems, Heraklion, Crete, Greece, September 13, 2018*. 2018, pp. 61–72.

[Mau+15] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. “C5: Cross-Cores Cache Covert Channel”. In: *Proc. of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148*. DIMVA 2015. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 46–64. DOI: 10.1007/978-3-319-20550-2_3.

[Mau+17] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, K. Römer, and S. Mangard. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: *Proc. of the 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. 2017.

[Mil89] J. K. Millen. “Finite-State Noiseless Covert Channels”. In: *Proc. of Computer Security Foundations Workshop*. 1989, pp. 81–86.

[Mil99] J. Millen. “20 years of covert channel modeling and analysis”. In: *Proc. of the Symposium on Security and Privacy*. 1999, pp. 113–114. DOI: 10.1109/SECPRI.1999.766906.

[Moo00] Gordon E. Moore. “Readings in Computer Architecture”. In: ed. by Mark D. Hill, Norman P. Jouppi, and Gurindar S. Sohi. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. Chap. Cramming More Components Onto Integrated Circuits, pp. 56–59.

[Nis] *National Vulnerability Database (NVD)*. https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&search_type=all. Last accessed on 03.08.2019.

[OST06] D. A. Osvik, A. Shamir, and E. Tromer. “Cache Attacks and Countermeasures: The Case of AES”. In: *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*. CT-RSA'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 1–20. DOI: 10.1007/11605805_1.

[Ope] *OpenSSL Crypthography and SSL/TLS Toolkit*. https://www.openssl.org. Last accessed on 16.08.2019.

[PG74] Gerald J. Popek and Robert P. Goldberg. “Formal Requirements for Virtualizable Third Generation Architectures”. In: *Commun. ACM* 17.7 (July 1974), pp. 412–421. DOI: 10.1145/361011.361073.

[Per05] Colin Percival. “Cache missing for fun and profit”. In: *Proc. of BSDCan*. 2005.

[RWB04] G. B. Ratanpal, R.D. Williams, and T.N. Blalock. “An on-chip signal suppression countermeasure to power analysis attacks”. In: *IEEE Transactions on Dependable and Secure Computing* 1.3 (2004), pp. 179–189.

[Ris+09] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds". In: *Proc. of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: ACM, 2009, pp. 199–212. DOI: 10.1145/1653662.1653687.

[SA19] D. Spinellis and P. C. Avgeriou. "Evolution of the Unix System Architecture: An Exploratory Case Study". In: *IEEE Transactions on Software Engineering* (2019), pp. 1–1. DOI: 10.1109/TSE.2019.2892149.

[SWT01] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. "Timing Analysis of Keystrokes and Timing Attacks on SSH". In: *Proceedings of the USENIX Security Symposium*. SSYM'01. Berkeley, CA, USA: USENIX Association, 2001.

[Sch] *Scheduling*. http://wiki.prgmr.com/mediawiki/index.php/Scheduling. Last accessed on 03.10.2019. 2008.

[Sch+17a] Christian Schlehuber, Markus Heinrich, Tsvetoslava Vateva-Gurova, Stefan Katzenbeisser, and Neeraj Suri. "A Security Architecture for Railway Signalling". In: *Proc. of the 36th International Conference on Computer Safety, Reliability, and Security, SAFECOMP 2017, Trento, Italy, September 12*. IEEE, 2017, pp. 320–328.

[Sch+17b] Christian Schlehuber, Markus Heinrich, Tsvetoslava Vateva-Gurova, Stefan Katzenbeisser, and Neeraj Suri. "Challenges and Approaches in Securing Safety-Relevant Railway Signalling". In: *Proc. of the 2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017*. 2017, pp. 139–145. DOI: 10.1109/eurospw.2017.63.

[Sha01] C. E. Shannon. "A Mathematical Theory of Communication". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 5.1 (2001), pp. 3–55.

[Shi+11] J. Shi, X. Song, H. Chen, and B. Zang. "Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring". In: *Proc. of DSN-Workshops*. 2011, pp. 194–199.

[Spp] *40th IEEE Symposium on Security and Privacy – Program*. https://www.ieee-security.org/TC/SP2019/program.html. Last accessed on 11.09.2019. San Francisco, CA, US, 2019.

[Sta+19] Emily Stark, Ryan Sleevi, Rijad Muminovi, Devon O'Brien, Eran Messeri, Adrienne Porter Felt, Brendan McMillion, and Parisa Tabriz. "Does Certificate Transparency Break the Web? Measuring Adoption and Error Rate". In: *Proc. of the IEEE Symposium on Security & Privacy, S&P 2019*. 2019, pp. 211–226.

[Ste+13] Deian Stefan, Pablo Buiras, Edward Z. Yang, Amit Levy, David Terei, Alejandro Russo, and David Mazières. "Eliminating Cache-Based Timing Attacks with Instruction-Based Scheduling". In: *Proc. of the 18th European Symposium on Research in Computer Security (ESORICS) 2013*. Ed. by Jason Crampton, Sushil Jajodia, and Keith Mayes. Springer Berlin Heidelberg, 2013, pp. 718–735.

[Suz+11] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. "Memory Deduplication As a Threat to the Guest OS". In: *Proc. of the Fourth European Workshop on System Security*. EUROSEC '11. Salzburg, Austria: ACM, 2011, 1:1–1:6. DOI: 10.1145/1972551.1972552.

[TV05] Kris Tiri and Ingrid Verbauwhede. "Simulation Models for Side-channel Information Leaks". In: *Proc. of the 42nd Design Automation Conference, DAC 2005*. 2005, pp. 228–233.

[Tir+05] K. Tiri, D. Hwang, A. Hodjat, B. Lai, S. Yang, P. Schaumont, and I. Verbauwhede. "A side-channel leakage free coprocessor IC in 0.18 μm CMOS for embedded AES-based cryptographic and biometric processing". In: *Proc. of the 42nd Design Automation Conference, DAC 2005*. 2005, pp. 222–227.

[VG+19] Tsvetoslava Vateva-Gurova, Salman Manzoor, Ruben Trapero, and Neeraj Suri. "Protecting Cloud-Based CIs: Covert Channel Vulnerabilities at the Resource Level". In: *Proc. of Information and Operational Technology Security Systems, Heraklion, Crete, Greece, September 13, 2018*. Ed. by Apostolos P. Fournaris, Konstantinos Lampropoulos, and Eva Marín Tordera. Cham: Springer International Publishing, 2019, pp. 27–38.

[VGCS] T. Vateva-Gurova, N. Coppik, and N. Suri. "SpyAlarm: Be the Spy and Spy the Attacker". In: *Transactions on Dependable and Secure Computing* (). [under submission].

[VMwa] VMware. *Additional Transparent Page Sharing management capabilities and new default settings*. Tech. rep. 2097593. "Last accessed on 07.06.2018." VMware.

[VMwb] VMware. *Security considerations and disallowing inter-virtual machine Transparent Page Sharing*. Tech. rep. 2080735. Last accessed on 07.06.2018. VMware.

[VRS14] Venkatanathan Varadarajan, Thomas Ristenpart, and Michael Swift. "Scheduler-based Defenses against Cross-VM Side-channels". In: *Proceedings of the USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 687–702.

[VS18] Tsvetoslava Vateva-Gurova and Neeraj Suri. "On the Detection of Side-Channel Attacks". In: *Proc. of the 23rd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC) PRDC'18, Taipei, Taiwan, December 4-7, 2018*. 2018, pp. 185–186.

[VSM15] T. Vateva-Gurova, N. Suri, and A. Mendelson. "The Impact of Hypervisor Scheduling on Compromising Virtualized Environments". In: *Proc. of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, DASC 2015, Liverpool, United Kingdom, October 26-28*. 2015, pp. 1910–1917. DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.283.

[Vat+14a] T. Vateva-Gurova, J. Luna, G. Pellegrino, and N. Suri. “Towards a Framework for Assessing the Feasibility of Side-channel Attacks in Virtualized Environments”. In: *Proc. of the 11th International Conference on Security and Cryptography - Volume 1: SECRYPT 2014, Vienna, Austria*. INSTICC. SciTePress, 2014, pp. 113–124. DOI: 10.5220/0005052101130124.

[Vat+14b] Tsvetoslava Vateva-Gurova, Jesus Luna, Giancarlo Pellegrino, and Neeraj Suri. “On the Feasibility of Side-Channel Attacks in a Virtualized Environment”. In: *E-Business and Telecommunications, ICETE 2014, Vienna, Austria, August 28-30, 2014, Revised Selected Papers*. Ed. by Mohammad S. Obaidat, Andreas Holzinger, and Joaquim Filipe. Springer International Publishing, 2014, pp. 319–339. DOI: 10.1007/978-3-319-25915-4_17.

[Vat+18] T. Vateva-Gurova, S. Manzoor, Y. Huang, and N. Suri. “InfoLeak: Scheduling-Based Information Leakage”. In: *Proc. of the 23rd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC) PRDC’18, Taipei, Taiwan, December 4-7, 2018*. 2018, pp. 44–53. DOI: 10.1109/PRDC.2018.00015.

[Ves+14] Fatbardh Veseli, Tsvetoslava Vateva-Gurova, Ioannis Krontiris, Kai Rannenberg, and Neeraj Suri. “Towards a Framework for Benchmarking Privacy-ABC Technologies”. In: *Proc. of the 29th IFIP TC 11 International Conference on ICT Systems Security and Privacy Protection, SEC 2014, Marrakech, Morocco, June 2-4, 2014*. 2014, pp. 197–204. DOI: 10.1007/978-3-642-55415-5_16.

[WL05] Z. Wang and R. B. Lee. “Capacity estimation of non-synchronous covert channels”. In: *Proc. of DSN-Workshops*. 2005, pp. 170–176.

[WL08] Zhenghong Wang and Ruby B. Lee. “A Novel Cache Architecture with Enhanced Performance and Security”. In: *Proc. of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-41. Washington, DC, USA: IEEE Computer Society, 2008, pp. 83–93.

[WXW12] Zhenyu Wu, Zhang Xu, and Haining Wang. “Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud”. In: *Proceedings of the USENIX Security Symposium*. Security’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 159–173.

[Wal02] Carl A. Waldspurger. “Memory Resource Management in VMware ESX Server”. In: *SIGOPS Oper. Syst. Rev.* 36.SI (Dec. 2002), pp. 181–194. DOI: 10.1145/844128.844146.

[Wea16] Vincent M. Weaver. *Advanced Hardware Profiling and Sampling (PEBS, IBS, etc.): Creating a New PAPI Sampling Interface*. Tech. rep. UMAINE-VMW-TR-PEBS-IBS-SAMPLING-2016-08. http://web.eece.maine.edu/~vweaver/projects/perf_events/sampling/pebs_ibs_sampling.pdf, Last accessed on 11.10.2019. University of Maine. UMaine VMW Group Tech Report, 2016.

[Wra91] J. C. Wray. “An analysis of covert timing channels”. In: *Proc. of Computer Society Symposium on Research in Security and Privacy*. 1991, pp. 2–7.

[Xen] *Xen Project Software Overview*. https://wiki.xen.org/wiki/Xen_Project_Software_Overview. Last accessed on 02.08.2019.

[Xen] Xen Project. *Credit Scheduler*. http://wiki.xen.org/wiki/Credit_Scheduler. Last accessed on 03.10.2019.

[Xen] *Xen Project*. http://www.xenproject.org/. Last accessed on 13.10.2014.

[Xu+11] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. “An Exploration of L2 Cache Covert Channels in Virtualized Environments”. In: *Proc. of the 3rd ACM Workshop on Cloud Computing Security Workshop*. CCSW ’11. New York, NY, USA: ACM, 2011, pp. 29–40. DOI: 10.1145/2046660.2046670.

[YF14] Yuval Yarom and Katrina Falkner. “FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-channel Attack”. In: *Proc. of the USENIX Conference on Security Symposium*. SEC’14. Berkeley, CA, USA: USENIX Association, 2014, pp. 719–732.

[Yan+19] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher W. Fletcher, Roy H. Campbell, and Josep Torrellas. “Attack Directories, Not Caches: Side Channel Attacks in a Non-Inclusive World”. In: *Proc. of the IEEE Symposium on Security & Privacy, S&P 2019*. 2019, pp. 888–904.

[Yar+15] Yuval Yarom, Qian Ge, Fangfei Liu, Ruby B. Lee, and Gernot Heiser. “Mapping the Intel Last-Level Cache”. In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 905.

[ZD05] Y. Zhou and F. DengGuo. *Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing*. Cryptology ePrint Archive, Report 2005/388. 2005.

[ZL15] Tianwei Zhang and Ruby B. Lee. “CloudMonatt: An architecture for security health monitoring and attestation of virtual machines in cloud computing”. In: *Proc. of the 42nd Annual International Symposium on Computer Architecture (ISCA)*. 2015, pp. 362–374. DOI: 10.1145/2749469.2750422.

[ZZL16] Tianwei Zhang, Yinqian Zhang, and Ruby B. Lee. “CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds”. In: *Proc. of the 19th International Symposium on Research in Attacks, Intrusions, and Defenses, RAID 2016*. 2016, pp. 118–140.

[Zha+11] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. “HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis”. In: *Proc. of the Symposium on Security and Privacy*. SP ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 313–328. DOI: 10.1109/SP.2011.31.

[Zha+12] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. "Cross-VM Side Channels and Their Use to Extract Private Keys". In: *Proc. of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. New York, NY, USA: ACM, 2012, pp. 305–316. DOI: 10.1145/2382196.2382230.

[Zho+11] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram. "Scheduler Vulnerabilities and Coordinated Attacks in Cloud Computing". In: *Proc. of the 2011 IEEE 10th International Symposium on Network Computing and Applications*. 2011, pp. 123–130. DOI: 10.1109/NCA.2011.24.

[dAJ05] R. deGraaf, J. Aycock, and M. Jacobson. "Improved port knocking with strong authentication". In: *Proc. of the 21st Annual Computer Security Applications Conference (ACSAC'05)*. 2005, 10 pp.–462. DOI: 10.1109/CSAC.2005.32.