Ensuring Compliance with Data Privacy and Usage Policies in Online Services

A dissertation submitted towards the degree

Doctor of Engineering
of the Faculty of Mathematics and Computer Science
of Saarland University

by

Aastha MEHTA

Saarbrücken, 2020

Date of Colloquium: 03 November 2020
Dean of Faculty: Prof. Dr. Thomas Schuster

Chair of the Committee: Prof. Dr. Christian Rossow
Reporters
First Reviewer: Prof. Dr. Peter Druschel
Second Reviewer: Prof. Dr. Deepak Garg
Third Reviewer: Dr. Jonathan Mace
Fourth Reviewer: Prof. Dr. Thomas Ristenpart
Academic Assistant: Dr. Isaac Sheff

To my parents, Sonal and Ketan, and my brother, Rohan.

Abstract

Online services collect and process a variety of sensitive personal data that is subject to complex privacy and usage policies. Complying with the policies is critical, often legally binding for service providers, but it is challenging as applications are prone to many disclosure threats. We present two compliance systems, Qapla and Pacer, that ensure efficient policy compliance in the face of direct and side-channel disclosures, respectively.

Qapla prevents direct disclosures in database-backed applications (e.g., personnel management systems), which are subject to complex access control, data linking, and aggregation policies. Conventional methods inline policy checks with application code. Qapla instead specifies policies directly on the database and enforces them in a database adapter, thus separating compliance from the application code.

Pacer prevents network side-channel leaks in cloud applications. A tenant's secrets may leak via its network traffic shape, which can be observed at shared network links (e.g., network cards, switches). Pacer implements a cloaked tunnel abstraction, which hides secret-dependent variation in tenant's traffic shape, but allows variations based on non-secret information, enabling secure and efficient use of network resources in the cloud.

Both systems require modest development efforts, and incur moderate performance overheads, thus demonstrating their usability.

Kurzdarstellung

Onlinedienste sammeln und verarbeiten eine Vielzahl sensibler persönlicher Daten, die komplexen Datenschutzrichtlinien unterliegen. Die Einhaltung dieser Richtlinien ist häufig rechtlich bindend für Dienstanbieter und gleichzeitig eine Herausforderung, da Fehler in Anwendungsprogrammen zu einer unabsichtlichen Offenlegung führen können. Wir präsentieren zwei Compliance-Systeme, Qapla und Pacer, die Richtlinien effizient einhalten und gegen direkte und indirekte Offenlegungen durch Seitenkanäle schützen.

Qapla verhindert direkte Offenlegungen in datenbankgestützten Anwendungen. Herkömmliche Methoden binden Richtlinienprüfungen in Anwendungscode ein. Stattdessen gibt Qapla Richtlinien direkt in der Datenbank an und setzt sie in einem Datenbankadapter durch. Die Konformität ist somit vom Anwendungscode getrennt.

Pacer verhindert Netzwerkseitenkanaloffenlegungen in Cloud-Anwendungen. Geheimnisse eines Nutzers können über die Form des Netzwerkverkehr offengelegt werden, die bei gemeinsam genutzten Netzwerkelementen (z. B. Netzwerkkarten, Switches) beobachtet werden kann. Pacer implementiert eine Tunnelabstraktion, die Geheimnisse im Netzwerkverkehr des Nutzers verbirgt, jedoch Variationen basierend auf nicht geheimen Informationen zulässt und eine sichere und effiziente Nutzung der Netzwerkressourcen in der Cloud ermöglicht.

Beide Systeme erfordern geringen Entwicklungsaufwand und verursachen einen moderaten Leistungsaufwand, wodurch ihre Nützlichkeit demonstriert wird.

Acknowledgements

I would like to thank several people whose guidance, encouragement, and support made it possible to complete this thesis. First of all, I would like to thank my advisors, Peter Druschel and Deepak Garg, for their invaluable and timely advice throughout my PhD, and for giving me space and time to develop my skills and ideas and to complete this thesis. Their support and motivation through rejections and deadlines has helped me develop grit and resilience. I am inspired by their commitment to excellence in research and hope to emulate it in my own future research.

I would like to thank Jonathan Mace and Tom Ristenpart for agreeing to be on my thesis committee and taking the time to review my thesis.

I am thankful to Björn B. Brandenburg for his valuable inputs in the Pacer project, and his advice on leadership and team management. I was fortunate to have worked closely with fellow students: Eslam Elnikety, Katura Harvey, Mohamed Alzayat, and Roberta De Viti. Their contributions to Qapla and Pacer are deeply appreciated. I would also like to mention Anjo Vahldiek-Oberwagner with whom I collaborated on earlier projects. Anjo and Eslam taught me the ropes of systems research. Thanks to Manohar Vanga for all those conversations about Linux and Xen, and to Ahana Ghosh for explaining how to train a neural network.

I would like to thank Rose Hoberman for providing critical feedback on many paper drafts and talks, and helping me improve my writing and presentation skills. Many thanks to the current and former members of the SysNets group, and to other fellow students, postdocs, and faculty at MPI-SWS, with whom I have shared several lunches and chatted about research, academic life, and a plethora of other topics.

Research at MPI would not have been possible without the immense support of the office staff. Mary-Lou helped navigate the requirements of the PhD program. Claudia, Annika, Gretchen, and Brigitta demysitified the complex paperwork related to German administration. Carina, Christian Klein, and Christian Mickler ensured that research was never delayed due to any IT-related issues.

Special thanks to friends, Vasundhara and Saakshita, who joined for coffee and dinner breaks, and helped tide over periods of extreme frustration.

I would like to thank my parents and brother for encouraging me to be ambitious and for boosting my morale whenever I faltered. I thank my parents-in-law for understanding my ambition and for providing their unwavering support in my endeavours. Last, but not the least, I would like to thank my husband, Arpan Gujarati, who has been a fellow traveller and a partner in crime on this journey of PhD.

He has faced the trials and tribulations of not one, but two PhDs. This thesis could not have been completed without you. Thank you for being a reviewer of my worst ideas. Most importantly, thank you for being my best friend.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today, online social networks, e-commerce websites, and mobile applications collect and aggregate a massive amount of personal information. Although this data enables services to provide rich user experience online, unintended disclosure and misuse of the data may affect users personally, socially, financially, and professionally. For instance, in one case, the introduction of new features in an online social network resulted in exposure of users' old posts leading to embarrassment for users [52]. Several e-commerce and travel booking services have been accused of online price discrimination, as they use differential pricing based on personal data [2]. Credit card numbers stolen from online services are misused by cyber criminals for committing fraud, which causes financial losses to victims [70]. Users' publicly exposed social media updates have enabled robbers to plan house robberies [1]. Data has even been misused to undermine the political discourse in society [160].

Legislators have reacted to this growing threat of data disclosures and misuse with stricter laws and regulations, such as the California Consumer Privacy Act (CCPA) [22] and the European Union's General Data Protection Regulation (GDPR) [59]. However, laws alone are not enough. Service providers must have the necessary tools to ensure that their systems remain compliant with the laws, regulations, policies, and users' choices.

Ensuring compliance in contemporary services is a challenging task for two main reasons. First, services handle data that is subject to *complex and diverse policies* governing their privacy and usage. For instance, users' photos in online social networks may be limited to friends or friends-of-friends, while their click history may be used only for personalization and may require expiration; email is private to sender and recipient(s); a company's personnel records may be accessible in full only to the HR (human resources) department, while a subset may be accessible to other employees.

Second, services tend to have *complex architectures and large, rapidly-evolving codebases*. Consequently, they are prone to various types of data disclosure and misuse threats. Threats may range from application bugs, misconfigurations and human errors to exploitable vulnerabilities, malware, side channels, and malicious insiders. Ensuring continued compliance with all the complex policies in a continuously evolving codebase and in the face of a variety of threats is non-trivial.

Data disclosures and misuse can be broadly categorized into two types: *direct* and *indirect*. In a direct disclosure, sensitive data immediately becomes available to an unauthorized user or a third party (principal), who may then misuse the data. Direct disclosures may arise due to bugs or misconfigurations of the system, which accidentally leak data. They may also arise when a malicious user exploits system vulnerabilities or inject malware to extricate sensitive data from the system. Several large-scale direct data disclosures have been reported in online services [46, 53, 109].

Additionally, even if a principal cannot directly access the sensitive data from a system, (s)he may *indirectly infer* the data via side channels. Side channels arise when mutually untrusting principals share physical resources. An unauthorized principal (called the adversary) can observe the application's (victim's) use of the shared physical resource, which may be correlated with the victim's secrets, and then infer the victim's secrets from its usage pattern. For instance, it has been shown that an adversary can learn the content of a VoIP (voice-over-IP) conversation by observing just the sizes and timing of the packets transmitted via shared network elements like switches and routers [180]. Side channel disclosures are difficult to eliminate since, at system design time, it is difficult to anticipate which resources can be exploited as side channels and what inferences an unauthorized principal can make from observing the victim's usage of the shared resource.

A systematic approach to mitigating both direct and side-channel disclosures is to design *compliance systems* that enforce data policies "by design", *i.e.,* that systematically subject all data accesses in an application to policy checks. A compliance system (i) specifies a threat model describing the threats covered by the system, *e.g.,* an adversary's capabilities and the channels over which data can be disclosed to the adversary; (ii) provides a policy specification language that allows specifying the data policies of the application in a clear, concise, and auditable manner; and (iii) provides a mechanism to enforce the policies and protect the data under the specified threat model.

For a compliance system to be practical, it must add low performance overhead and require minimal changes to the application. Moreover, to ensure compliance

despite rapid changes in the application codebase, the compliance system must be independent from the application. In particular, the policy specification and enforcement should be independent of the application logic. Finally, the compliance system itself must be secure and not reduce the security of the application being protected.

1.1 Thesis contributions

In this thesis, we demonstrate that one can *build practical compliance systems that can prevent direct and side-channel disclosures in online services by ensuring compliance by design with precise data privacy and usage policies, independently of application logic.* We present the design, implementation and evaluation of two compliance systems that prevent direct and side-channel disclosures in specific settings. First, we present Qapla [113], a system that prevents direct disclosures due to accidental bugs and misconfigurations in database-backed applications. Second, we present Pacer [114], a system that prevents network side-channel disclosures in cloud applications.

1.1.1 Qapla: Policy compliance in database-backed systems

A large number of web services today store their confidential data in relational database management systems (DBMSs). This data is often subject to complex and fine-grained policies. In a personnel management system, for instance, ordinary employees may be allowed to query their own personal information but not that of others. Members of a workers' council may be allowed to query the columns containing employee names and ages separately, but not together, to prevent them from linking employees to their ages. Similarly, members of the payroll department may not be allowed to query the health history of individual employees, but they may be allowed to query for aggregates over the health histories of all employees.

Today, services attempt to ensure policy compliance by implementing policy checks inlined with the application logic. Specifically, when an application accesses sensitive data, checks are implemented to determine whether the access complies with relevant policies of the data. Such policy checks are scattered throughout the application codebase. Ensuring continued compliance in this manner is cumbersome and error-prone. As the applications or data policies evolve, developers potentially need to revisit all code paths to ensure that all data accesses continue to remain policy compliant. It is easy to miss some checks or implement incorrect checks, and accidentally disclose sensitive data to an unauthorized user.

We present Qapla, a compliance system that prevents such accidental data disclosures in database-backed applications. Qapla provides a policy specification language that can specify a rich class of application policies, including the ones described above. The policies include fine-grained access control on individual rows, columns and cells, as well as complex conditions to restrict or relax access to cells based on query operators applied to cells. For instance, Qapla can specify a policy to prevent an application from linking two columns through join or filter operations, and another policy to allow the application to access a column only in aggregated form (Section 3.4). The policies are specified in a SQL-like language, as a function of the database schema, and stored in the database itself (in separate tables).

For policy enforcement, Qapla uses a reference monitor that intercepts application queries at runtime, looks up applicable policies, and rewrites queries to make them policy compliant (Section 3.5). The reference monitor is integrated with a database adapter between the application and the underlying DBMS, and requires no changes to and no specific support from the DBMS for policy enforcement. Moreover, the integration with a database adapter makes Qapla portable across DBMSs.

Qapla's policy enforcement by query rewriting is completely transparent to application queries that are already policy compliant. And for non-compliant queries, Qapla still returns the policy-compliant subset of results, thus preserving maximal functionality even for non-compliant application queries.

To evaluate Qapla, we used it to ensure policy compliance in two applications (Section 3.8): (i) HotCRP [74], a popular conference management system that handles conference paper submissions and reviews, and (ii) APPLY, our institute's job application portal that handles candidate applications and selection process workflows. We empirically verified Qapla's security guarantees by injecting known disclosure bugs in HotCRP and checking that Qapla successfully prevents disclosures (Section 3.9). We also evaluated Qapla's performance, and observed that Qapla incurs moderate runtime overheads on applications.

Contributions summary Qapla is a policy-compliance middleware for database-backed applications that enforces policies in a DBMS-agnostic and an application-transparent manner. Our main contributions are: (i) a policy language that enables specifying a rich class of data policies, including data linking and aggregation policies, independently from both the application code and any DBMS-specific support; (ii) the design and architecture of the reference monitor that allows policy enforcement independently from application and DBMS; and (iii) an evaluation that

demonstrates Qapla's security guarantees and moderate performance overheads, indicating its practicality for database-backed applications.

1.1.2 Pacer: Network side-channel mitigation in cloud applications

Side channels are a growing vector for data disclosures, of particular concern for applications hosted in public clouds, where mutually distrusting tenants share the cloud's physical resources. Numerous side-channel disclosures have been demonstrated in public clouds, which exploit a variety of physical resources (*e.g.*, CPUs, caches, and memory) shared between tenants co-located on the same server or between the tenants and the host (hypervisor and OS) [3, 11, 48, 58, 80, 81, 95, 103, 131, 132, 135, 136, 152, 164, 186, 188, 189, 194] (see Section 2.4 for discussion).

Even if tenants rent dedicated servers or CPU sockets and use memory only within their local NUMA (Non-Uniform Memory Access) domain, they may still share network elements: the server's network interface card (NIC), a top-of-the-rack switch, or a router. By generating cross-traffic on such shared elements and observing its delay, an adversarial tenant can infer the shape of a co-located victim's encrypted network traffic [4, 144, 146].

In contrast to side channels via shared microarchitectural state and memory, side channels via shared network elements have not received much attention, particularly in the context of cloud computing. In this thesis, we present Pacer, a compliance system that focuses on mitigating network side channels in the cloud. Specifically, Pacer prevents leaks of a tenant's secrets to an adversary who can observe, directly or indirectly, the shape of the tenant's encrypted network traffic.

Pacer mitigates such leaks by making the *shape* of victim's network traffic *independent* of its secrets. Shaping involves padding each of a tenant application's outgoing messages to a secret-independent size, and transmitting all packets at secret-independent times. While Pacer actively prevents secret-dependent variations in the traffic shape, it specifically allows variations in the shape based on non-secret information of the tenant, thus reconciling security with efficiency (Section 4.3).

To implement traffic shaping, we present the abstraction of a *cloaked tunnel* that encapsulates the observable network path between the tenant application and its clients, and enforces a selected shape on the application's traffic (Section 4.4). The tunnel pads payload packets and transmits them at times defined in the transmit schedule (shape) for the application traffic, transmitting a dummy packet when the

application fails to prepare payload packets in time. Thus, the resulting traffic shape is *secret-independent by design*.

Pacer realizes the cloaked tunnel abstraction for the cloud environment, where an adversary may be co-located with the victim's VM on the same server and therefore may observe the victim's traffic at the shared network interface of the server. To handle this case, Pacer integrates the tunnel with the cloud's hosting server. Pacer relies on a paravirtualization approach to implement the tunnel, which ensures full isolation from potentially secret-dependent computations in the guest VM and requires modest changes to the cloud hypervisor and the guest OS (Section 4.5).

To generate efficient traffic shapes, the tenant partitions its workloads based on public information. For each partition, a profiler samples the distributions of the tenant's unmodified payload traffic shapes and computes a schedule, which is subsequently used to shape all the tenant's traffic within the partition (Section 4.6).

To evaluate Pacer, we used it to shape traffic of two applications (Section 4.8): (i) a document hosting service based on Mediawiki [112], and (ii) a custom video-streaming service hosting a corpus of ~1200 YouTube videos. For both applications, Pacer's shaping incurs modest overheads on bandwidth, application throughput, and client response latencies.

Contributions summary Pacer is a novel and efficient system that prevents network side-channel leaks in cloud tenants by design and with minimal support from the tenant application. Our main contributions are: (i) a traffic shaping strategy that makes traffic shapes independent of the tenant's secrets and allows variations in the shapes only based on non-secrets; (ii) a gray-box profiler that generates traffic shapes automatically from the tenant's network traces with minimal support from the tenant application; (iii) a cloaked tunnel abstraction, which ensures that the shape of network traffic in the tunnel is independent of secrets; (iv) a paravirtualized implementation of the cloaked tunnel for an IaaS cloud server that ensures full isolation from potentially secret-dependent computations of the tenant; and (v) an evaluation that demonstrates that strong mitigation of network side channels is possible with moderate overheads.

1.2 Publications and collaborations

This thesis contains text and material from the following two publications [113, 114]:

- "Qapla: Policy compliance for database-backed systems".
  Aastha Mehta, Eslam Elnikety, Katura Harvey, Deepak Garg, Peter Druschel.
  USENIX Security Symposium, 2017.

- "Pacer: Network Side Channel Mitigation in the cloud".
  Aastha Mehta, Mohamed Alzayat, Roberta De Viti, Björn B. Brandenburg, Peter Druschel, Deepak Garg. *Under submission*. *arXiv preprint available at* https://arxiv.org/abs/1908.11568.

The following paragraphs describe the origins of the text and material in this thesis in more detail.

Chapter 1 The introduction is largely written by me. Section 1.1 paraphrases some text from the introductions of the above two papers.

Chapter 2 The background is entirely written by me.

Chapter 3 The text of Chapter 3 is derived from the Qapla paper [113], which was written by me in collaboration with my advisors. The case study of APPLY (Section 3.8.2) and the corresponding experiments and results (Section 3.9.3.2) were developed in collaboration with my co-author, Katura Harvey. The experiments of sections 3.9.4 and 3.9.6 were designed with assistance from Eslam Elnikety, while the experiments were executed by me.

Chapter 4 The text of Chapter 4 is derived from the Pacer paper [114], which was written by me in collaboration with my co-authors. Sections 4.2-4.6 are mostly taken from the paper, while the remaining sections expand on the material from the paper.

The core Pacer system was designed, implemented and evaluated by me with guidance from my advisors. The clustering algorithms (Section 4.6.2), the gray-box profiler (Section 4.6.1), and the experimental results (sections 4.8.2, 4.8.4, and 4.8.5) were generated in collaboration with Mohamed Alzayat and Roberta De Viti.

Chapter 5 The conclusion and future work is written by me.

During my PhD, I have also contributed to the following publications [45, 124, 166], which are not included in this thesis.

- "Guardat: Enforcing data policies at the storage layer".
  Anjo Vahldiek-Oberwagner, Eslam Elnikety, Aastha Mehta, Deepak Garg, Peter Druschel, Ansley Post, Rodrigo Rodrigues, Johannes Gehrke.
  European Conference on Computer Systems (EuroSys), 2015.

- "Thoth: Comprehensive Policy Compliance in Data Retrieval Systems".
  Eslam Elnikety, Aastha Mehta, Anjo Vahldiek-Oberwagner, Deepak Garg, Peter Druschel. *USENIX Security Symposium*, 2016.

- "Oblivious Multi-Party Machine Learning on Trusted Processors".
  Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, Manuel Costa. *USENIX Security Symposium*, 2016.

1.3 Organization

The rest of this thesis is organized as follows. In Chapter 2, we discuss background on data policies, direct and side-channel disclosures, and prior work on mitigating disclosures. In Chapter 3 and Chapter 4, respectively, we describe the design, implementation and evaluation of Qapla and Pacer, as well as discuss more specific related work. Finally, we summarize this thesis and outline directions for future research in policy compliance in Chapter 5.

Chapter 2

Background and Prior Work

In this chapter, we first discuss data policies and compliance system components. We then give a general overview of the threats and mitigation mechanisms for the two kinds of data disclosures that are the focus of this thesis: direct and side-channel disclosures. We discuss more specific related work in Chapters 3 and 4.

2.1 Data policies

Services today access data that has complex privacy and usage policies. These policies arise from three broad sources:

(i) *User preferences.* In an online social network, for instance, users may wish to share their photos only with friends, but not with others.

(ii) *Service provider's internal policies.* For example, in an organization, only the human resource employees may have full access to all employee records, while all the other employees may have access to only a subset of the records. Members of the equal opportunities division may have access to salary statistics but may not be allowed to access salaries of individual employees.

(iii) *External legal requirements.* For example, with the General Data Protection Regulation (GDPR) [59], organizations need to ensure that personal data is protected from unintended disclosures[1], and that no personal data item is used

[1] *Article 5(1)(f) [59]: Personal data shall be processed in a manner that ensures appropriate security of the personal data, including protection against unauthorised or unlawful processing and against accidental loss, destruction or damage, [...] ("INTEGRITY AND CONFIDENTIALITY")*

for unintended purposes[2]. They may even be required to delete the collected user data after a stipulated period of time[3].

There are several challenges involved in ensuring compliance with such policies. First, the policies are diverse and complex, and often stated in a natural language like English. Mapping the natural-language policies to specific enforceable components within a system is difficult. Second, systems within which such policies must be enforced are typically complex and evolving. This makes it difficult to continuously ensure compliance with all policies across all code paths. Finally, systems are prone to numerous threats that can violate data policies. Building a system that comprehensively mitigates all threats is non-trivial.

Although an important problem, this thesis does not focus on solutions to translate natural-language specifications into enforceable system-level specifications, nor does it provide any mechanisms to compare and match the natural-language specifications and their system-level descriptions. However, as we describe later, we provide a policy specification framework that enables concise and intuitive specifications of application policies, which are easy to inspect and reason about.

In addition, to address the second and the third challenge, we present compliance systems that can enforce data policies independently from the application code, while addressing specific classes of threats, *viz.* direct and side-channel disclosures.

2.2 Compliance system

In the introduction we described a principled approach to preventing data disclosures and misuse, which is to design a dedicated compliance subsystem that is responsible for this task. Designing a compliance system involves (i) defining a precise threat model describing threats considered and assumptions made by the compliance system, (ii) specifying data policies that the application needs to comply with, and (iii) designing a mechanism to enforce the specified policies. Next, we elaborate on the three components of the compliance system design process.

Defining a threat model The threat model must precisely state an adversary's capabilities (*e.g.*, passive observation, active exploits) and the channels through which

[2] *Article 5(1)(b) [59]: Personal data shall be collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes [...] (Purpose limitation)*

[3] *Article 5(1)(e) [59]: Personal data shall be kept [...] for no longer than is necessary for the purposes for which the personal data are processed [...] (Storage limitation)*

data can be disclosed to the adversary (*e.g.,* a web, filesystem or database interface). It must also define the trusted computing base—the set of components that the compliance system relies on for ensuring compliance, including external components, such as an authentication server for verifying the identity of principals in the system. Finally, it is assumed that the adversary cannot subvert or bypass the compliance system to access the data, and that the compliance system itself does not disclose sensitive data it is designed to protect.

Policy specification Data privacy and usage policies must refer to principals that can access the data (a principal may be associated with a human user or an application component), the data object being accessed (*e.g.,* a disk block, a file, a database column) types of access (*e.g.,* a block write, a file read, or a database query), and conditions under which each principal is granted or denied each type of access (*e.g.,* time of access, the state or content of the data store, the organizational role of a user).

A compliance system must provide a policy specification language that can express the relations between the principals, the data objects, the type of access, and the conditions of access in the data policies. The specification enabled by the language must be clear, concise, and easy to both reason about and audit.

Enforcement To enforce policies, each data flow through the channels that is observable by an adversary must be subject to policy checks. Data must be allowed to flow through the channel, only if the flow is compliant with relevant policies.

Policies can often be enforced at multiple layers in the software stack. For instance, a policy controlling read of a table column in a relational database can be specified and enforced at the database layer or at the granularity of individual disk blocks constituting the table. At the database layer, the policy can be easily specified using a SQL view and enforced by executing application queries against the view. However, correct policy enforcement would require that the DBMS implements views correctly. At the disk layer, the policy must map the table column to disk blocks and the disk blocks to corresponding policies, while enforcing the policy would require intercepting read of every disk block constituting the column, which could potentially incur significant overheads. The choice of the enforcement layer depends on the granularity of data on which policies are specified and the threats being covered, and determines the efficiency of the compliance system, the size of the trusted computing base, and the usability of the system.

In summary, a compliance system guarantees that only policy-compliant data disclosures can be made through the specified channels. We now discuss the kinds of threats that lead to direct and side-channel disclosures, and the general techniques that have been developed to mitigate such threats for different application settings.

2.3 Direct data disclosures

Direct disclosures can arise in two ways: accidentally or due to active exploits. In the following, we briefly explain both types of disclosures, and then discuss why we focus on accidental disclosures in this thesis. We also give a background on different types of access control policy specification and enforcement mechanisms, which are the state-of-the-art techniques to enforce policies and prevent disclosures.

2.3.1 Accidental disclosures and active exploits

In accidental disclosures, the system itself unintentionally reveals sensitive data to unauthorized, *passive* users via its interfaces. Such disclosures may occur due to bugs or misconfigurations of the system, causing compliance failures. Indeed, accidental disclosures have exposed sensitive user data in many systems [23, 53].

On the other hand, even if a system does not have bugs that immediately disclose sensitive data, it may still have vulnerabilities that could be *actively* exploited by an adversary to circumvent policy checks. For instance, the Heartbleed vulnerability [69] in OpenSSL library could be used to extricate private keys of an application relying on the library, which in turn would allow an adversary to decrypt application's network traffic, thus violating all data policies. Alternatively, an adversary can exploit a vulnerability to install a malware that extricates authentication credentials; the adversary can then use the credentials to access sensitive data as an "authorized" user [109].

Despite the numerous incidences of large-scale accidental disclosures [23, 53, 54, 115, 116], systematic solutions to prevent them do not exist. Consequently, this thesis focuses on the problem of mitigating accidental direct disclosures.

Furthermore, this thesis focuses on accidental disclosures arising due to bugs and misconfigurations in applications. In general, disclosures can arise also in the system stack, such as the operating system, the storage subsystem (*e.g.*, file system or database management system), or the network subsystem. However, while applications are maintained by multiple teams of developers and evolve more rapidly, the

system stack is often maintained by a small team of experts in an organization's IT department and evolves slowly. Moreover, the system stack is typically composed of components from reputable vendors, and is likely to be patched regularly with security updates. As a result, there are likely to be more unintentional data disclosures from the application layer compared to the system stack.

2.3.2 Preventing accidental disclosures

The key mechanism to prevent direct disclosures is access control. Access control allows principals *authorized* for a particular access to a resource to access the resource, and prevents *unauthorized* principals from accessing the resource[4]. We next discuss various techniques to specify and enforce access control policies.

File system access control File system access control [65, 68] protects persistent files from unauthorized users or applications. Access control policies are specified in the form of an access control list (ACL), which is a set of system principals authorized to perform specific operations on a file in the file system. Principals may represent users or applications in the system, and file operations may include read, write, and execute.

File system ACLs provide only coarse-grained access control. They do not allow fine-grained control on file operations, such as preventing only parts of a file from being read, or allowing a file to be modified in certain ways, but not others. They also do not allow one to specify time-based conditions (*e.g.*, a file must not be accessible after a certain date) or conditions on principals (*e.g.*, a file can only be accessed by employees with a certain role), although the latter could be emulated with "group permissions" in the ACLs. Thus, file system ACLs are inconvenient for specifying complex data access policies on data.

Guardat [166] supports complex policies on files that go beyond the standard file system ACLs. The policies are specified in a declarative language based on Datalog [97], can express conditions on users, roles, access time, and state and content of files, and are enforced at the disk block layer. The file abstraction is too coarse-grained to efficiently specify and enforce fine-grained policies, such as those arising in database-backed applications. Consequently, filesystem ACLs and Guardat are not suitable for ensuring policy compliance in such applications.

[4] Authorization of principals refers to the mechanism of specifying the access rights of the principals to a resource.

DBMS access control Relational DBMSs store data in database tables in the form of rows and columns. Thus, access control in DBMS naturally requires restricting access to rows, columns, and cells. A large number of relational DBMSs [77, 140, 147, 155, 161] allow specifying access control policies on rows in database tables in a way that is similar to the filesystem access control. Principals and roles are defined in the DBMS and each row in a table is allowed access to a set of principals.

To specify policies on columns, there are different types of specification frameworks. A popular approach is to create separate policy-defined logical views of the database tables for each principal, such that each view allows appropriate principals access to only the policy-compliant subset of the data from underlying tables. The view definitions can express conditions on users, roles, access time, and even on values in the database tables. Application queries are then restricted to the database views rather than the underlying tables. Policy views have been proposed in several research systems [10, 14, 55, 66, 134], and is the most basic mechanism for column access control supported by all conventional DBMSs. However, views are not effective for enforcing access control by design. Policy changes require modifying not only the database views, but also application queries. In particular, when a more restrictive policy is added to the database, application queries must be updated to query the correct database view in order to enforce policies correctly. Thus, views may fail to prevent disclosures if there are bugs in the application queries.

In addition to view-based access control, certain DBMSs like IBM DB2 [147], SQL Server [156], and Oracle [130] provide explicit syntax to specify policies on columns, while Oracle [161] also provides a unified syntax for row-, column-, and cell-level access control specification. However, such support for policy specification is DBMS-specific and non-portable, and forces applications to be locked in with a specific DBMS vendor.

Policy specification and enforcement in applications Existing access control support in the storage systems is not sufficient to ensure compliance with complex policies that often arise in applications. Consequently, applications tend to implement policy checks on their own. They often implement access control operationally, *i.e.,* as code, and inlined with the application logic. However, enforcing data policies this way is cumbersome and error-prone. Instead, specifying policies using a *declarative language* and *on the data* enables a concise and intuitive specification that is easier to reason about, audit, and maintain independent of the application code. Additionally, the logic for checking declarative policies can be typically implemented

centrally and with few lines of code, which minimizes the trusted computing base of the compliance system.

Examples of declarative policy specification frameworks include EPAL (Enterprise Privacy Authorization Language) [7] and XACML (eXtensible Access Control Markup Language) [50]. EPAL is a generic policy specification framework from IBM that was designed for use in enterprises. EPAL specifies enterprise privacy policies in terms of user categories, data categories, purposes, actions, conditions, and obligations. However, EPAL has not seen widespread adoption in many applications.

XACML [50] is another platform-independent policy specification framework, which is a web standard and has been adopted by some web services for access control policy specification [183]. XACML enables declarative, fine-grained attribute-based access control policies. It specifies application-specific attributes on users, data, actions, and system environment, and a policy specifies conditions on these attributes. Both EPAL and XACML need to be implemented by each application separately. Moreover, they support policies on abstract, high-level user actions within the application, such as view, edit, or delete, but do not support fine-grained restrictions on database accesses.

In this thesis, we focus on policy compliance in database-backed applications, which need to comply with complex policies such as preventing certain data linking operations or allowing access to only aggregated data. Existing policy specification frameworks cannot support such policies. With Qapla, we provide a declarative policy specification language that can express such policies independently from the application code and without relying on any specific DBMS support. Furthermore, Qapla enforces the policies on each database query independently from the application code and the DBMS, thus eliminating all accidental disclosures due to bugs in application queries by design.

2.4 Side-channel disclosures

Recall from Chapter 1 that one of our contributions is Pacer, a compliance system to ensure compliance with data policies in the face of network side-channel disclosures. We provide below some background on side channels in general, and explain the key principles underlying the different ways to mitigate side-channel disclosures.

2.4.1 Understanding side channels

Side channels arise in shared resources, such as CPUs, cores, hardware counters, caches, memory, storage, schedulers, files, network elements, and network packets. An effective side-channel attack requires the adversary to have *a priori* knowledge of the correlations between the victim's secrets and the resource usage pattern the secrets generate. The adversary may gain this knowledge, for instance, by profiling the victim's application on its own infrastructure. Throughout this thesis, we simply assume that the adversary has the relevant knowledge.

A side-channel attack involves two steps. First, the adversary observes the usage of a shared resource, which is correlated with the victim's secrets. The adversary then uses statistical techniques to infer the victim's secrets from the side-channel observations. Next, we elaborate on the two steps.

Capturing observations over side channels Broadly speaking, side-channel observations can arise from an application in two ways: accidentally and due to an adversary having compromised the application.

In accidental disclosures, an adversary simply captures the application's usage of shared resources as an observer external to the application. For instance, side channel attacks have been demonstrated using unprivileged application processes that bypass OS isolation [3, 48, 131, 164, 188, 189], as well as unprivileged VMs in clouds that bypass hypervisor isolation [78, 80–82, 103, 136, 195]. In addition, attacks have also been demonstrated against applications running in trusted execution environments (TEEs), which are supported by strong hardware-based isolation [95, 167, 186]. In network side channels, the adversary can directly observe the victim's traffic shape at a network link [20, 26, 63], or indirectly infer the traffic shape by contending with victim's traffic at a shared link and measuring the available bandwidth or queueing delays for its own traffic [4, 144, 146].

In contrast to accidental side-channel disclosures, an adversary can exploit application vulnerabilities to gain access to its secrets, and then transmit the secrets out of the system by exploiting the shared resources from within the victim's system. We call such side-channel disclosures as *covert channel* disclosures. An adversary may encode the secrets in the timing modulation of the system's execution or its network transmissions, in the state of resources used by the application (*e.g.,* by modifying the size of a file on shared file system), or in spare bits of network packet headers

[61, 173, 191]. (The adversary relies on side channels because it may not have a direct communication channel to the outside, or it may want to avoid detection.)

In server applications, an adversary can also compromise the clients of the application; it may then send requests to the server impersonating a legitimate client and infer the server's secrets based on the server's response times. Such adversaries have been shown to exploit the execution time of cryptographic libraries to infer cryptographic keys [11, 17, 18, 89] and, more recently, the network transmission time to infer usage pattern of microarchitectural resources at the remote server [90, 152][5].

Making inferences from observations Once the adversary collects the necessary observations over a side channel, it applies analytical techniques to glean sensitive information from the observations. Depending on the side channel exploited, the analyses may range from simple aggregation and thresholding mechanisms to more sophisticated machine learning classifiers.

In certain exploits, such as those based on shared microarchitectural state, the victim's secret is inferred one bit at a time. The analysis involves simple thresholding mechanisms to identify the bits in the victim's secrets: for example, if the latency of the adversary's access to a resource is above a threshold, the victim's bit is 1, otherwise it is 0 [49, 103, 188].

In contrast to microarchitectural side channels, network side-channel exploits infer secrets from the sizes and timing of victim's packets or bursts. They rely on more sophisticated statistical analyses. The earliest statistical analyses involved measuring the similarity or distances between the distributions of observations (*e.g.,* Jaccard's similarity coefficient [101], or optimal string alignment distance [21, 174]), and then using classifiers based on Naïve Bayes [43, 71], nearest neighbours [176] or Support Vector Machines (SVM) [19, 20, 133]. With advances in machine learning, the analyses have become more sophisticated. Most recently, deep learning techniques based on Neural Networks have been used for side-channel inferences [146]. Indeed, we also demonstrate a network side-channel attack in Section 4.1, where an adversary is able to use a Convolutional Neural Network (CNN) to accurately identify video streams based only on noisy timing measurements of the streams.

With the growing number of shared resource exploits and the increasing sophistication of available analyses, side channels are becoming a major concern for service

[5]For a detailed discussion on side and covert channels based on shared microarchitectural state, we refer the reader to surveys by Ge *et al.* [58] and Szefer [159].

providers. Public clouds particularly lower the barrier to obtain the necessary observations through side channels, making accidental side-channel disclosures quite feasible. An adversarial tenant can simply rent cheap virtual machines, co-locate them with a victim tenant's virtual machine on a cloud server, or within a rack in the datacenter [78, 144], observe the victim's usage of shared resources, and thereby learn the victim's secrets. For this reason, and also owing to the large-scale use of public clouds, we focus on accidental side-channel disclosures in this thesis.

2.4.2 Mitigating side-channel disclosures

Unlike direct disclosures, the main challenge for mitigating side-channel disclosures is not specifying data policies, but rather enforcing them. Indeed, a compliance system to mitigate side-channel disclosures essentially enforces a single implicit policy:

The application's sensitive data must not be inferrable by any unauthorized principal.

This seemingly simple policy is actually non-trivial and difficult to enforce, since it is difficult to anticipate the observation and inference capabilities, as well as the background knowledge of unauthorized principals (as discussed in Section 2.4.1). In the following, we discuss prior work related to three high-level approaches for enforcing such a policy: (i) adding noise to the channel observations, (ii) partitioning the physical resource, and (iii) shaping the victim's usage of the shared resource.

Noising observations A natural approach to mitigating side channels is to rely on noise in the adversary's observations. Noise is implicitly present when unrelated computations share the physical resource being observed. However, an adversary could simply wait for an opportune moment to attack when unrelated noise in the victim's execution is low. Alternatively, additional synthetic noise can be added in the victim's execution [89] to thwart attacks. However, ad-hoc or random noise may not always be sufficient in level and entropy to overcome a sophisticated adversary's detection and inference capabilities.

In contrast to random noise, differentially private noise [42] can provide quantifiable and tunable guarantees on privacy leaks. Zhang *et al*. [193] recently proposed using differentially private noise to mitigate network side channels. They demonstrate that adding noise to network traffic shapes of a corpus significantly reduces the accuracy of machine learning classifiers in classifying the video corpus. However, their precise privacy guarantees against the adversary remain unclear. Unlike in traditional analytics settings, where the differentially private noise added to

a corpus determines the number of queries that a (adversarial) querier is allowed to execute on the corpus, a side-channel observer can execute infinite classification "queries" on the traffic shapes, which could increase the privacy loss significantly and eventually allow the identification of the videos. In general, it is difficult to anticipate, much less restrict, the number of queries an adversary will perform on the side-channel observations to make inferences. Hence, understanding the efficacy of differential privacy in mitigating side channels remains an open problem.

Zhang *et al*. [193] also used adversarial machine learning samples to defeat classifiers, but demonstrate that it is difficult to defeat multiple classifiers with the same adversarial inputs. In other words, adversarial inputs cannot prevent leaks to an adversary that can adapt its classifiers to circumvent the noise in victim's observations.

Another approach is to reduce the adversary's capabilities of performing timing measurements, for instance, by preventing access to fine-grained clock sources or by adding noise to the reported clock values [105, 110, 170]. However, Schwarz *et al*. [151] demonstrate how adversaries can generate high-precision timing measurements from coarse-grained clocks and simple counter processes, thus indicating that eliminating clock sources is an unreliable approach.

In general, one cannot anticipate the precise observation and inference capabilities of an adversary. Robust side-channel mitigation solutions must therefore assume that the adversary can learn all information contained in the resource usage pattern of the victim.

Resource partitioning A physical resource that can be used for a side-channel leak can be partitioned or replicated among different tenants, in order to completely *prevent* an adversary from observing a victim's usage of the resource. The resource can be partitioned in two ways, spatially and temporally.

Spatial partitioning divides a resource into smaller units, each of which may be assigned to different tenants. Prior work has proposed partitioning of memory, caches, and hardware performance counters to mitigate related side channels [15, 104, 197].

In contrast, *temporal partitioning* multiplexes, in time, a shared resource among different tenants. Temporal partitioning can be achieved by using a time-division multiple access (TDMA) scheduling policy [84] to schedule access to a shared resource. In TDMA scheduling, when a time slice is reserved for a tenant to access the resource, no other tenant can access the resource for the duration of that time slice. Temporal partitioning has been used to mitigate side channels due to shared cores and core-local resources, such as L1 and L2 caches [154, 169].

TDMA scheduling can also be used to mitigate network side channels. Network bandwidth can be reserved for flows at the physical layer, which would eliminate the adversary's (and in fact every other tenant's) ability to observe a co-located tenant's traffic. However, for end-to-end mitigation, TDMA scheduling must be applied *synchronously* along a victim's path, *i.e.,* the TDMA slot for a tenant's transmission must be enabled at the same time at all shared network elements in the victim's path. Implementing efficient, synchronous TDMA within a datacenter and in the Internet is difficult [171]. Synchronous TDMA with short time slots would cause high overheads for tenants' peak workloads and waste cycles in the frequent switching between tenants' traffic, while TDMA with longer time slots would significantly reduce bandwidth utilization when the tenants' traffic is below their peak workloads.

Both spatial and temporal resource partitioning generally reduce resource utilization, and are at odds with the idea of resource sharing in a public cloud.

Usage shaping An alternate approach is to *shape* the victim's resource usage pattern to be independent of its secrets, so that an adversary cannot infer the secrets despite observing the victim's resource usage pattern. For example, constant-time implementation ensures that the execution time of all program paths is secret-independent. Constant-time programming has been used to secure implementations of cryptographic libraries [31–33] against execution time based side-channel leaks. However, the technique has not been shown to generalize to large-scale software systems. Moreover, constant-time implementation cannot mitigate leaks arising via other vectors, such as the sequence of memory addresses or sizes of network packets.

Traffic shaping is a technique used to mitigate leaks via network side channels, specifically leaks through the shape of the victim's network traffic. Shaping involves padding individual network packets to a secret-independent sizes, padding bursts of fixed-sized packets to secret-independent lengths, and transmitting the packets at a secret-independent rate [73, 148, 153].

In contrast to microarchitectural and memory side channels, network side channels are not understood well in the cloud environments. Therefore, this thesis focuses on mitigating network side channels in clouds. Furthermore, existing network side-channel mitigation techniques do not consider threats from a co-located adversary contending on a shared NIC, and therefore are inherently insufficient to mitigate leaks in a cloud datacenter. They are also inefficient for the bursty workloads that arise in web services. Pacer is a system that mitigates network side channels

for cloud tenants by design. Pacer presents a secure and efficient traffic shaping strategy, as well as a system that can enforce the shaping securely on a cloud server without depending on any significant support from the tenant application.

Chapter 3

Qapla: Policy Compliance in Database-backed Systems

This chapter describes the design, implementation and evaluation of Qapla, a system that ensures policy compliance in database-backed applications. Qapla enables applications to comply with complex data linking and aggregation policies, which go beyond fine-grained cell-level access control and require constraining query operators, such as join, aggregate, and group by.

As we described in Section 2.3.2, DBMSs cannot support such complex policies in an application-transparent manner. We elaborate further on the limitations of DBMSs with a concrete example and the need for Qapla in Section 3.1. We then give an overview of Qapla's key goals and design (Section 3.2), and discuss its precise threat model (Section 3.3). We describe Qapla's policy specification framework (Section 3.4), the design of Qapla's reference monitor and the enforcement mechanism (Sections 3.5 and 3.6), and Qapla's implementation (Section 3.7). Next, we describe our two case studies on HotCRP and APPLY (Section 3.8), and report our evaluation results (Section 3.9). We discuss Qapla's limitations and possible extensions to overcome them (Section 3.10). Finally, we discuss the related work (Section 3.11).

3.1 Motivation

Consider an application with a database table with the schema `T(C1, C2)`. The application serves three users `U1`, `U2`, and `U3`, and wishes to enforce the following two policies on the data access.

(i) Users `U1` and `U2` can read both columns `C1` and `C2` independently, while only `U1` can link the two columns together.

View id	View definition	User access
V1	SELECT C1 FROM T	U1, U2
V2	SELECT C2 FROM T	U1, U2
V3	SELECT C1, C2 FROM T	U1
V4	SELECT SUM(C2) FROM T	U3

TABLE 3.1: View-based policy enforcement

(ii) User U3 is prohibited from reading the individual values of columns, but is allowed to query for aggregates (*e.g.*, SUM) on C2.

Such policies may arise, for example, in an organization's personnel management system where column C1 contains the names of users, column C2 contains their salaries, and the users U1, U2, and U3 are respectively an employee in the human resources (HR) department, an employee in a department other than HR, and a member of an external auditing committee evaluating the organization's pay scales.

Enforcing these policies in existing DBMSs requires creating logical views of the table T with appropriate subsets of columns and restricting users' accesses to the views according to the prescribed policies. Table 3.1 summarizes the views and access policies that need to be defined for the application to enforce the two policies. Policy (i) requires creating views V1, V2, and V3, while policy (ii) requires V4.

Furthermore, correct enforcement of the two policies requires the application to query the appropriate views on behalf of the users. Specifically, to enforce policy (i), the application must use views V1 and V2 when a query accesses individual columns C1 and C2, respectively, but it must use view V3 for a query accessing the two columns together. Similarly, to enforce policy (ii), the application must use V4 when a query is issued to access C2 by user U3.

As can be seen, view-based policy enforcement requires trusting the application to query the correct policy-defined views. Moreover, view-based enforcement requires modifying all application queries, including compliant ones, whenever views change due to a change in policies.

One way to address the above limitations would be to add support in DBMSs to transparently rewrite application queries with appropriate policy views. However, this solution requires support from each DBMS vendor.

Given these challenges, many applications attempt to enforce policies themselves within the application code [40, 41, 74, 112, 126, 127]. Essentially, every data access is checked *within the application* to determine whether the user, on whose behalf the

application is accessing the data, is authorized to access the data. However, inlining of policy checks is also a cumbersome and error-prone approach. As the applications or data policies evolve, developers potentially need to revisit all code paths to ensure that all data accesses continue to remain policy compliant. It is easy to miss some checks or implement incorrect checks, and trigger accidental disclosure of sensitive data to an unauthorized user.

With Qapla, we explore the design of a policy compliance system that is independent from both the application code base as well as the underlying DBMS. Qapla addresses the limitations of the DBMS access control, and enables policy compliance in a DBMS-agnostic and application-transparent manner.

3.2 Design overview

A principled approach to designing a general policy compliance system for database-backed applications requires addressing four goals.

G1 The policy specification framework must be comprehensive and provide a uniform syntax for specifying a rich class of complex and fine-grained policies.

G2 The language must be simple and intuitive for the policy administrators to adopt, and must enable specification that is easy to understand, reason about and audit.

G3 The policies should be specified directly on the data and independently from the application codebase.

G4 Enforcement of the policies should not depend on DBMS-specific support, and should be transparent to applications that issue policy-compliant queries.

Qapla's design satisfies all the above goals. Qapla's policy specification framework allows specifying a rich class of policies, including standard fine-grained row-, column-, and cell-level access control, and also complex policies that limit data linking or allow specific data aggregations and transformations. The policies are *specified* in a SQL-like language as a function of the database schema (but not of the individual applications built on the database), and are stored in the database independent from the application codebase and data. For policy *enforcement*, Qapla integrates a reference monitor with a generic database adapter, which intercepts all application queries, looks up applicable policies, and rewrites queries to ensure compliance.

Figure 3.1 depicts Qapla's architecture. Qapla's metadata and policies are stored in the database (in separate tables). The Qapla reference monitor authenticates with

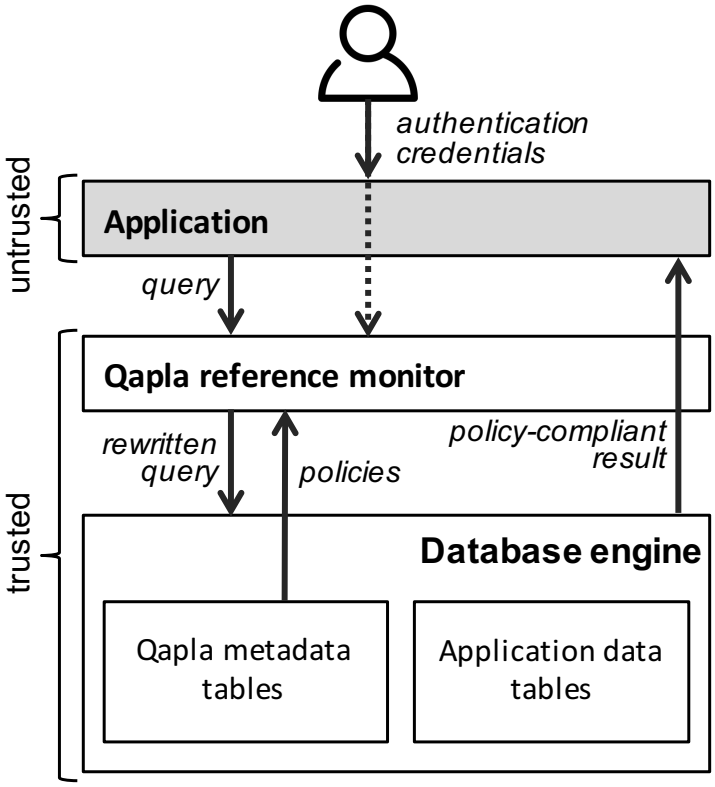


FIGURE 3.1: Qapla architecture

the database with its own unique credentials, and it has the exclusive privilege to access all tables directly. It intercepts the application's database queries, and associates each query with the authenticated end user on whose behalf the query was issued by the application. The query is rewritten to ensure its compliance with policies, and the rewritten query is executed by the database.

SQL is a natural choice for Qapla's policy language due to several reasons. First, SQL is widely understood and already used by developers to write database queries; this makes it easier for them to specify policies as additional SQL syntax. Second, SQL enables a high-level, declarative specification of policies, which makes it easier to reason about, analyze, audit, and debug the policies compared to policies written in application code. Finally, the SQL-like syntax of the policies simplifies Qapla's query rewriting for enforcement.

Because the policies are associated with the database schema, policy changes can be affected reliably based on the schema alone, without requiring inspection

of queries or modification of compliant queries by application programmer. Furthermore, Qapla's rewriting-based enforcement is transparent to application queries that are already policy compliant, so the application has to be changed only where its queries are not policy compliant. Qapla requires no changes to and no support from the underlying DBMS (although we describe in Section 3.5 how database-specific support like materialized views can be used to optimize Qapla's performance).

Qapla is essentially a policy compliance middleware for database-backed applications. It removes the often large and rapidly evolving applications from the codebase trusted for compliance, simplifies new applications by obviating the need for pervasive filtering code, and avoids compliance bugs due to incorrect or missing application checks. Moreover, since the Qapla reference monitor is integrated in a generic database adapter and does not depend on DBMS-specific access control support, it is portable across DBMSs. We demonstrate Qapla's portability by incorporating it with PHP's and Python's database adapters, and using it to enforce fine-grained policies in two applications: HotCRP and APPLY (see Section 3.8).

3.3 Threat model

Our goal is to protect confidentiality of data in the face of *application bugs* and *misconfigurations*. Specifically, we wish to prevent data leaks due to application bugs that result in non-compliant queries to the database.

We assume that the application bugs do not compromise Qapla's enforcement. Specifically, we trust the application in two ways. First, we assume that the application can authenticate a user, and forward the authentication information correctly to Qapla for policy compliance. Second, we assume that the application-level bugs or vulnerabilities cannot circumvent Qapla's reference monitor to access the database directly, or steal the reference monitor's privileged database credentials. These assumptions are not fundamental to Qapla's design, and are only limitations of its prototype. In Section 3.10 we discuss possible modifications to Qapla's prototype to relax these assumptions.

Qapla enforces link policies only on individual application queries, and not across multiple queries. We assume that individual users do not link non-overlapping parts of the database they have obtained in separate queries. We also assume that users do not collude offline (outside the application) to combine the information they are individually authorized to read. In general, preventing all types of *offline linking attacks*

(linking of information across queries) is a hard problem, and we discuss potential solutions to specific instances of the problem in Section 3.10.

We do not consider active attacks, such as SQL injection attacks that may exploit vulnerabilities in SQL's interface[1]. We trust the Qapla reference monitor, the database infrastructure comprising of the DBMS and the adapter it provides for an application to connect to the database, the operating system, and the storage layer, and assume that each of these are correctly configured. The database curator or compliance team is assumed to have installed correct policies, and any information referenced by policies is assumed to be correctly stored in the database. Under these assumptions, Qapla guarantees that only policy compliant query results are returned to the application.

3.4 Policy framework

In this section, we describe Qapla's policy specification framework, and demonstrate its use to specify confidentiality policies using specific examples.

Qapla enforces fine-grained policies on individual cells of a database. A database cell can be accessed through four kinds of operations in a query. A cell can be (i) accessed as part of a column access, (ii) linked with other cells of the row or with rows from other tables, (iii) accessed after a transformation, or (iv) accessed after aggregation with other cells, *i.e.,* an aggregated value including the cell value can be accessed. Qapla's policies can specify constraints on each kind of cell operation.

One way to specify the policies would be to enumerate all the cells in a database and associate explicit policies with each cell. However, this would blow up the number of policies that need to be specified, as well as the space overhead for storing the policies. Fortunately, cells within individual columns tend to have a similar policy template, with policies differing only in a few parameters. The parameter values may depend on the state or value of the same cell or other cells, columns, or rows in the database, or on conditions such as time. Thus, Qapla policies are specified as templates applicable to sets of columns, which describe restrictions for various query operations on the columns and individual cells.

Every Qapla policy applies to a class of queries based on the way the query operates on different columns. The policy specifies how those queries must be restricted to be compliant. The restrictions are specified as SQL WHERE clauses that are added

[1]However, Qapla can defend against a limited class of SQL injection attacks in the application layer that attempt to read unauthorized portions of the database. We discuss this further in Section 3.5.2.

to the query by Qapla before the query is executed. Qapla executes the rewritten query, thus filtering out non-compliant records. An application can obtain a tuple using a query only if (a) at least one policy applies to the query, and (b) the query rewritten under the restrictions of the applicable policies produces the tuple. If no policy applies to a query, the query is not executed. This whitelist principle ensures that data is accessed only due to some explicitly written policy and never leaked due to accidental omission of policies. We formally define when a policy applies to a query and the query rewriting procedure in Section 3.5.

We now reveal the features of Qapla's policy specification framework incrementally through a series of policies for an example personnel management system. We consider the human resources database of a fictitious company called `Acme`. The database has three tables:

(i) `Employees(empID, name, address, age, gender, dept)`,

(ii) `Payroll(empID, salary)`, and

(iii) `Benefits(empID, health_plan)`.

The first table maps employees to their home address, age, gender and department. The second table maps employees to their salary, while the third table specifies which health insurance plan each employee subscribes to.

3.4.1 Single column policies

The simplest Qapla policy protects a single database column by specifying which rows (cells) of the column can be accessed by each user, and when. It has the form:

`col` :- FILTER-CONDITIONS

Here, FILTER-CONDITIONS is of the form T:W, where T is the table containing the column `col` and W is a SQL WHERE clause that specifies which rows from `col` can be returned. FILTER-CONDITIONS may refer to the authenticated user and the wall clock time using the variables $user and $time, respectively, which are instantiated by the Qapla reference monitor when the clause is added to the query. The policy applies to any query that references only the column `col` (queries that read more than one column are subject to link policies described later).

Example 1 (name, age, health_plan) The names of `Acme`'s employees should be accessible to all other employees. The following policy specifies this.

$$\texttt{name} \text{ :- } \texttt{Employees} : \text{EXISTS}(\text{SELECT } 1 \text{ FROM } \texttt{Employees} \\ \text{WHERE } \texttt{empID} = \$\text{user})$$

The SQL fragment EXISTS(...) specifies a condition that holds only if the authenticated user exists in the table `Employees`. An identical policy applies to the columns `age` and `health_plan`. Note that employees can access each of these columns only in isolation, which only allows enumerating the names, ages or health plans of all employees separately, but does not allow, for example, knowing the employees' ages.

Example 2 (address, salary) The columns `address` and `salary` can be read only by members of the HR (human resources) department. Additionally, an employee may read his or her own address or salary. The following policy enforces this on `address`. A similar policy applies to `salary`.

$$\texttt{address} \text{ :- } \texttt{Employees} : ((\texttt{empID} = \$\text{user}) \text{ OR} \\ \text{EXISTS}(\text{SELECT } 1 \text{ FROM } \texttt{Employees} \\ \text{WHERE } \texttt{empID} = \$\text{user AND } \texttt{dept} = \text{HR}))$$

Compared to the policy of `name`, this policy allows different employees access to different entries in `address`. Note that the WHERE clause is organized as a disjunction of conditions, one for each class of users.

This is an example of a role-based access control (RBAC) policy, where an employee's role is dictated by her affiliation with a particular department. This policy relies on the availability of the mapping from users to their roles in the database itself. In applications where this mapping is outside the database (e.g., on a file system), Qapla's policy language can be easily extended to support predicates that look up this mapping outside the database. Qapla can interpret these non-database predicates in the policies using native procedures, and apply the remaining SQL policy to the database queries.

3.4.2 Link policies

When a query reads two or more columns, more information can be exposed by linking the columns to each other than what is exposed from the columns individually.

Therefore, an additional policy is required to prevent such linking. The applicable policy can even be more restrictive than the individual policies of all the columns read, as in the following example.

Example 3 (linking name and age) The policies of the columns `name` and `age` allow any employee to read these columns individually (Example 1). However, not every employee should be able to read the columns `name` and `age` *together* since that reveals every employee's age, which may be private. The right policy is that only members of HR and an employee himself/herself may read the employee's `name` and `age` together. In Qapla, this policy is expressed by mentioning both columns `age` and `name` to the left of :- in the policy.

$$\{\texttt{name}, \texttt{age}\} \text{ :- } \texttt{Employees} : ((\texttt{empID} = \$\text{user}) \text{ OR } \text{EXISTS}(\text{SELECT } 1 \text{ FROM } \texttt{Employees} \text{ WHERE } \texttt{empID} = \$\text{user} \text{ AND } \texttt{dept} = \text{HR}))$$

Such policies, which apply to simultaneous access of two or more columns, are called *link policies*. Their general form is

$$\{\texttt{col}_1, \ldots, \texttt{col}_n\} \text{ :- FILTER-CONDITIONS}$$

with FILTER-CONDITIONS of the form T_1:W_1 , . . . , T_m:W_m. Here, $\{\texttt{col}_1, \ldots, \texttt{col}_n\}$ are columns spanning the tables $T_1, \ldots, T_m$ and $W_1, \ldots, W_m$ are separate WHERE clauses for these tables, respectively. The link policy applies to any query that reads a subset of the columns $\{\texttt{col}_1, \ldots, \texttt{col}_n\}$ (for any purpose including projection, selection, joining, grouping or aggregation). The WHERE clauses of all the tables mentioned in the query are added to the query by Qapla (see Section 3.5 for details).

Columns in separate tables When the goal is to restrict the linking of data in two or more *separate* tables, the effect of a link policy can sometimes be simulated by simply restricting access to the individual columns containing the common keys of the two tables. However, when different sets of columns spanning the tables need different policies, the policies must be specified using the general form of link policies described above.

3.4.3 Transformation policies

Applications often apply functions or transformations to columns to hide sensitive information. A transformed column could have more permissive policies than the column itself. Qapla directly supports such transformations-aware policies.

Example 4 Suppose `Acme` provides a home-to-office shuttle service to its employees, run by `Acme`'s "logistics" department. The shuttle service has a fixed stop in every neighborhood that houses an employee but it is not door-to-door. In order to provide this service, members of the logistics department must know the neighborhood in which every employee lives, but not their precise home addresses. To enforce this, the privacy compliance team can create a user-defined function (UDF), `neigh`, that maps an address to a neighborhood, and add the following Qapla policy.

$$\{\texttt{name}, \texttt{address}[\texttt{neigh}]\} \text{ :- } \texttt{Employees} : ((\texttt{empID} = \$\text{user}) \text{ OR } \text{EXISTS}(\text{SELECT } 1 \text{ FROM } \texttt{Employees} \text{ WHERE } \texttt{empID} = \$\text{user AND } (\texttt{dept} = \text{HR OR } \underline{\texttt{dept} = \text{logistics}})))$$

This policy says that an employee's `name` and `neigh(address)` can be linked by the employee, members of HR, and members of logistics. The policy is strictly more permissive than the policy on $\{\texttt{name}, \texttt{address}\}$, which allows access only to the respective employee and HR, but not to logistics. The revised policy allows logistics to run the query

$$\text{SELECT } \texttt{name}, \texttt{neigh}(\texttt{address}) \text{ FROM } \texttt{Employees}$$

but not

$$\text{SELECT } \texttt{name}, \texttt{address} \text{ FROM } \texttt{Employees}$$

The general form of a Qapla transformation policy is

$$\{\texttt{col}_1[\texttt{t}_1], \ldots, \texttt{col}_n[\texttt{t}_n]\} \text{ :- FILTER-CONDITIONS}$$

The FILTER-CONDITIONS are of the same form as in a link policy. The policy applies to any query that accesses a subset of the columns $\texttt{col}_1, \ldots, \texttt{col}_n$ but only after the respective transformations $\texttt{t}_1, \ldots, \texttt{t}_n$ have been applied.

3.4.4 Aggregation policies

Many applications declassify aggregate statistics on otherwise private columns. Accordingly, Qapla provides *aggregation policies*. An aggregation policy specifies two sets of columns: 1) *LS* (link set)—columns which can be projected, used to join or group data (SQL's GROUP BY) or be aggregated in a query, and 2) *JS* (join set)—columns which can be used only to join tables in the query and nothing else. With each column in *LS* an optional transformation or aggregation operation can be specified, which restricts the use of that column to only that transformation or aggregation. The general syntax is

$$\{JS = \{\texttt{jcol}_1, \ldots, \texttt{jcol}_m\},\ LS = \{\texttt{col}_1[\texttt{t}_1], \ldots, \texttt{col}_n[\texttt{t}_n]\}\}\ \text{:-}\ \text{FILTER-CONDITIONS}$$

Example 5 Suppose `Acme` has a workers' council (WoC) that periodically computes salary statistics to ensure fairness in worker compensation. One statistic it computes is the distribution of average salary over age ranges (20-30 years, 30-40 years, etc.). Rather than provide WoC full access to the `Employees` table, the policy compliance team can selectively provide WoC rights to compute only such statistics by writing the following aggregation policy. Here, `age_range` is a function that rounds an individual's age to a 10-year range.

$$\{JS = \{\texttt{Payroll.empID}, \texttt{Employees.empID}\},\ LS = \{\texttt{age}[\texttt{age\_range}], \texttt{salary}[\texttt{AVG}]\}\}\ \text{:-}$$
$$\texttt{Payroll, Employees}:\ \text{EXISTS}\ (\text{SELECT}\ 1\ \text{FROM}\ \texttt{Employees}$$
$$\text{WHERE}\ \texttt{empID} = \$\text{user}\ \text{AND}$$
$$(\texttt{dept} = \text{HR}\ \text{OR}\ \underline{\texttt{dept} = \text{WoC}}))$$

This policy allows WoC to run any query that joins tables `Payroll` and `Employees`, and then uses only `age_range(age)` and average on `salary` (in any way). For example, it allows queries similar to the following two instances:

(i) SELECT `AVG(salary)`, `age_range(age)`
FROM `Employees` JOIN `Payroll` ON `empID`
GROUP BY `age_range(age)` HAVING `AVG(salary)` > 50000

which lists age groups with average salaries above 50000.

(ii) SELECT `AVG(salary)` FROM `Employees` JOIN `Payroll` ON `empID`
WHERE `age_range(age)` $= (30, 40)$

which lists the average salary of a specific age group.

Correctly, the policy does not allow queries that look at the `age` or `salary` columns directly. For instance, the following query is disallowed by the policy:

SELECT `AVG(salary)` FROM `Employees` JOIN `Payroll` ON `empID`
WHERE `age` $= 75$

Indeed, such queries can be used to identify the salary of individuals with unique ages by repeating the query for every age value, and therefore must be disallowed.

3.4.5 Relation between policy classes

Qapla's four policy classes—single-column policies, link policies, transformation policies and aggregation policies—are increasingly more general. Single-column policies are an instance of link policies, where the set of linked columns is a singleton. Link policies are a special case of transformation policies where the transformations are identity functions. A transformation policy S :- FILTER-CONDITIONS is the same as the aggregation policy $\{JS = \{\},\ LS = S\}$:- FILTER-CONDITIONS.

3.4.6 Policy inference heuristics

To reduce the compliance team's burden of specifying policies, Qapla provides three safe heuristics for automatically determining the policies on column sets in a schema.

Heuristic 1: A link policy for a set of columns also automatically applies to any subset of those columns since reading a subset only reveals less information than does reading the whole set. Thus, there is no need to specify a link policy on a subset unless the subset's policy is strictly more permissive than the policy of the whole set, and some application needs the permissiveness.

Heuristic 2: If a query uses column transformations or aggregations but a specific applicable transformation or aggregation policy does not exist, Qapla applies the link policy of the set of columns that occur in the query, if one exists. This is safe because transforming or aggregating a column always reduces the amount of information revealed.

Heuristic 3: In place of writing an explicit link policy on a set of columns, the designer can explicitly instruct Qapla to automatically construct a link policy for a set of columns by combining the policies of the individual columns in the set. This synthesized policy applies the conjunction of the FILTER-CONDITIONS of the individual

columns even when they are read together. This is useful in some cases. For instance, we may want to allow only HR members and an employee simultaneous access to the employee's `name` and `address`. However, this is exactly the policy of the individual column `address` (Example 2). Therefore, for such cases, the policy language can be extended to specify a keyword on the column set (here $\{$`name`, `address`$\}$) which can instruct Qapla to synthesize the link policy for the column set by combining the policies of the individual columns (here `name` and `address`).

3.5 Enforcement

Qapla's policy enforcement on a query consists of two steps: (i) Identifying the set of policies that apply to the query, and (ii) Rewriting the query to filter out tuples disallowed by all the applicable policies. We describe the two steps in detail.

3.5.1 Identifying applicable policies

Internally, Qapla treats every policy as an aggregation policy of the form $\{JS, LS\}$:- FILTER-CONDITIONS, where JS and LS are, respectively, the set of columns that may be used to (only) join two or more tables, and the set of columns that may be projected, grouped by and aggregated. As explained in Section 3.4.5, this is the most general form of policies; all single-column, link and transformation policies can be expressed in this form. Qapla parses every application query to extract the corresponding sets js and ls of columns that are used only to join and those that the query actually projects, groups by, or aggregates.

A policy applies to a query if the query's use of the columns js and ls is allowed by the corresponding sets JS and LS of the policy. Formally, the policy applies to the query when $js \subseteq JS$ and when every column `c` and every transformed column `c`[`t`] in ls is *dominated* by a column or transformed column in LS. Domination is defined as follows: Every (transformed) column dominates itself, and a column dominates any transformation of itself. Thus, the following policy with $\{JS = \{$`Benefits.empID`, `Employees.empID`$\}$, $LS = \{$`age`, `health_plan`$\}\}$ applies to a query with $js = \{$`Benefits.empID`, `Employees.empID`$\}$ and $ls = \{$`age`[`age_range`], `health_plan`[`COUNT`]$\}$. Figure 3.2 summarizes this algorithm.

To efficiently find all policies that apply to a query, Qapla maintains two data structures. The first data structure maps every pair of a column and a transformation (that applies to the column) to a bitvector representing the policies in the system. The

```
1  # Does a policy apply to a query?
2  input: Query Q
3  input: Policy pol of form {JS, LS} :- FILTER-CONDITIONS
4  output: true if pol applies to Q, false otherwise
5  {js, ls} = parseQuery(Q)
6  if (js ⊈ JS) then return false
7  for each column c in ls
8    if (c ∉ LS) then return false
9  for each transformed/aggregated column c[t] in ls
10   if (c[t] ∉ LS and c ∉ LS) then return false
11 return true
```

FIGURE 3.2: Algorithm to decide if a policy applies to a query

ith bit is set in the bitvector of the (transformed) column j if policy i's LS contains a column that dominates j. To find all applicable policies whose LS matches a given query's ls, Qapla simply takes the bit-wise AND of the bitvectors of all (transformed) columns in ls. The second data structure is similar but applies to JS and allows finding all policies whose JS matches a query's js. Finally, Qapla takes the bit-wise AND of the two bitvectors to find the policies that must be applied to the query.

3.5.2 Query rewriting algorithm

The query rewriting algorithm modifies an application query to make it compliant. In the simple and common case where only one policy applies to the query (only one policy bit set in the final bitvector generated from the identification step), the policy rewriting algorithm replaces each reference to a table in the query with a subquery of the form (SELECT $*$ FROM `table` WHERE *list-of-conditions*), where *list-of-conditions* are the FILTER-CONDITIONS for the table provided in the policy. Each subquery generates a list of rows compliant with the FILTER-CONDITIONS of the columns accessed from the table. The overall effect is that the application query is executed over joins of policy-compliant subtables of the database tables, where the subtables are created using the FILTER-CONDITIONS of the applicable policy.

Example 6 In the context of `Acme`'s database, assume that some link policy exists for the column set {`name`, `age`, `health_plan`, `Employees.empID`, `Benefits.empID`} and that it specifies the WHERE clauses f_E and f_B for filtering the tables `Employees` and `Benefits`, respectively. Consider the following query: SELECT `name`, `age`, `health_plan` FROM `Employees` JOIN `Benefits` ON `Employees.empID` = `Benefits.empID`. This query will be rewritten to:

SELECT `name`, `age`, `health_plan` FROM
(SELECT $*$ FROM `Employees` WHERE f_E) `Employees` JOIN
(SELECT $*$ FROM `Benefits` WHERE f_B) `Benefits` ON
(`Employees.empID` $=$ `Benefits.empID`)

When more than one policy applies to a query and the query does not return an aggregate, Qapla rewrites the query according to each applicable policy and takes a SQL UNION of these. This ensures that a tuple exists in the result only when at least one applicable policy allows it. If the query returns an aggregate value and more than one policy applies, Qapla picks the first applicable policy, but the application may override this to a specific applicable policy at the cost of minor changes to its code. (We have not encountered the need for such changes in our evaluation.)

Protection against SQL injection attacks Although not the focus of our work, Qapla's policy enforcement mechanism provides protection against a limited form of SQL injection attacks aimed at violating database confidentiality. Below we give an example of the kind of attacks Qapla can protect against. Consider an application query of the form:

SELECT $*$ FROM `Employees` WHERE `Employees.empID` $=$ $\$var$

The parameter $\$var$ is populated at runtime with the id of the authenticating user. An adversary may attempt to perform a SQL injection attack by passing a string of the form "X OR $(1 = 1)$". In absence of proper input sanitization, the adversary would be able to read the entire `Employees` table from the database. Qapla's reference monitor, which rewrites queries after parameter resolution, would replace the `Employees` table with a subquery consisting of the link policy for the link of all columns of the table. The link policy would be at least as restrictive as the conjunction of individual policies of all columns. Thus, the adversary's query would be

rewritten as:

SELECT $*$ FROM
(SELECT $*$ FROM `Employees` WHERE f_{empID} AND f_{name} AND ... f_{dept}) `Employees`
WHERE `Employees.empID` $=$ X OR $(1 = 1)$

The adversary's query will only be able to access a subset of the table compliant with all the columns' policies.

3.5.3 Optimizations

Next we describe three heuristics to reduce the overheads of policy enforcement during runtime. In our current prototype we implement the first optimization, and the second optimization partially. Implementing the third optimization is not difficult.

3.5.3.1 Query template cache

The Qapla reference monitor implements a query template cache to amortize the overhead of parsing and rewriting queries with the same structure. A query template is a query with all its constant values replaced with placeholder variables. The Qapla template cache maps query templates to their rewritten forms. When a query is received, Qapla converts the query to a template and checks if a query template with the same hash is cached (if the application query is already parametrized, Qapla hashes it directly). For a hit, Qapla retrieves the associated rewritten query template, and binds its variables with the values from the submitted query. For a miss, Qapla parses and rewrites the query with the applicable policies, and inserts the resulting rewritten query template into the cache.

3.5.3.2 Partial evaluation

The Qapla reference monitor often generates complex rewritten queries containing several nested subqueries accessing one or more tables, and having large filter conditions. Executing the query efficiently depends on the ability of the DBMS to generate an efficient execution plan for the rewritten query. To reduce the complexity of the rewritten query, Qapla can pre-evaluate parts of the rewritten query that do not depend on database values (e.g., parts that depend only on time, or the identity of the user on whose behalf the application makes the access) before posting the query to the database. This can significantly simplify the query since any predicates

connected by 'AND' to a pre-evaluated predicate that is false can all be replaced by a single false before the query is sent to the database. Similarly, any predicates connected by 'OR' to a pre-evaluated predicate that is true can all be replaced by a single true. In our prototype we only pre-evaluate time-based conditions.

3.5.3.3 Materialized views

To offset the cost of policy checks during query evaluation, Qapla can create materialized views, one for each (group of) user(s) with similar permissions, by applying applicable policies to the tables offline. In a group's materialized view, every cell inaccessible to the group is replaced with a special value that is not a legal value for the underlying table. At runtime, every application query is run against the materialized view appropriate for the authenticated user. The query is rewritten by Qapla to disregard any record that contains the special value in a field that is used in the query. Note that for confidentiality, it is insufficient to disregard a record only when one of its inaccessible fields is projected. It is also necessary to disregard a record if one of its inaccessible fields will be tested by the query's WHERE clause(s). Doing so prevents implicit information leaks through the WHERE clause(s).

Our preliminary evaluation suggests that this optimization can reduce runtime overheads on read-intensive workloads by an order of magnitude. However, proportional to the number of user groups with different policies, maintaining materialized views adds storage cost and runtime overhead to propagate updates to all materialized views.

3.6 Compatibility with legacy applications

The policy enforcement algorithm described in Section 3.5 drops a row during query execution if any field in the row is inaccessible according to the policy and is used in the query. This *row-suppression mode* of policy enforcement ensures that information about an inaccessible field cannot be inferred even when that information is correlated with other fields in the row. This makes row-suppression a very safe choice for policy enforcement (and, hence, Qapla uses it by default). However, row-suppression is not the only possible way of enforcing Qapla's policies. We briefly describe here a second mode of policy enforcement, the *cell-blinding mode*.

The primary consideration for the cell-blinding mode is compatibility with legacy applications, which may issue broad queries that project more columns than actually necessary, and eventually remove the extra columns in their own code. With the row-suppression mode, such broad queries may result in fewer records than expected by the application. Transitioning such applications to make them compatible with row-suppression may require effort and time, as developers may have to rewrite queries to not project unnecessary columns. This transition can be particularly difficult when the set of necessary columns depends on the application state.

The cell-blinding mode changes the semantics of policy enforcement to compromise some security and efficiency in return for accommodating overly broad queries. In this mode, Qapla rewrites the application queries to replace (blind) inaccessible cells with special values that can be returned in results, before executing the original query's logic. (This replacement is identical to the replacement of inaccessible cells in the creation of materialized views from Section 3.5.3, but here the special values must not depend on any secrets since they can be returned directly in query results.)

However, the cell-blinding mode has two drawbacks. First, if some fields of a record are inaccessible according to the policy, the record is still returned (with the inaccessible fields blinded). This leaks some information when the presence of the record in the database is sensitive and when blinded fields are correlated with other non-blinded fields. Second, the cell-blinding mode imposes significant overhead on query execution (up to two orders of magnitude for some queries with MySQL) due to the need to check policies on, and possibly blind, individual cells in every query. We believe that the use of materialized views described in Section 3.5.3 can reduce this overhead substantially. A full study of this approach remains as future work.

Due to these limitations of the cell-blinding mode, it is preferable to use the row-suppression mode and to modify the application to restrict overly general queries. Qapla uses only the row-suppression mode of policy enforcement.

3.7 Implementation

The Qapla implementation consists of ~20,000 lines of C code. It provides the API to create application-specific policies, associates policies with column identifiers in the database schema, and maintains an in-memory mapping from column identifiers to associated policies. It also provides an API for setting application-specific user authentication parameters (*e.g.,* login credentials) in the reference monitor. Qapla uses an existing SQL parser from the MySQL workbench [121] to extract accessed tables

and columns. The SQL parser consists of ~1.01 million lines of C code. A rewrite module implements the lookup for applicable policies and the query rewriting algorithm. A template cache module maintains a cache of rewritten query templates, and a customizable translation module can translate the SQL dialect of one DBMS to that of another, allowing Qapla uses across DBMSs. In our evaluation, we translate MySQL queries into a commercial DBMS's queries.

Qapla can support existing PHP and Python based applications. For PHP applications, we modified the PHP Data Objects (PDO) [137] module in the PHP interpreter. For Python applications, we rely on the Django framework [39], which provides an object-relational mapping (ORM) API for database interaction. Django provides a database-independent abstraction to the application developer. We modified this abstraction and interface with the Qapla reference monitor using the ctypes library. Both PDO and Django can be used to connect with different databases, such as MySQL, SQLite, MSSQL, and Oracle. Modifications to PDO and Django were limited to 135 and 141 lines of code, respectively.

3.8 Case studies

Next, we describe our use of Qapla to ensure compliance in HotCRP and APPLY.

3.8.1 HotCRP

Policies We studied HotCRP's schema and wrote policies based on our knowledge of its workflow. In many cases, we reverse-engineered HotCRP's policies by inspecting its code base to confirm and correct our intuition. In total, we specified 35 policies for the 22 tables and 215 columns in the schema of HotCRP version 2.99, which supports a broad range of configurations for a conference. The policies cover a single-track conference with a double-blind submission process, handling of chair conflicts with paper managers, and a review process with no rebuttal.

Table 3.2 shows a subset of the policies associated with important tables like contacts, papers, reviews, and conflicts. The full set of policies is listed in Section 3.12. The policies are explained in plain English for clarity and brevity of exposition but are actually written in the language introduced in Section 3.4. Macros abbreviate common SQL fragments that appear in many policies. Many of the policies are fine-grained access control predicates on user, time, and the content of various database tuples. There are also link and aggregation policies.

id	table	column list	allow the authenticated user U access to row R if ...
C3	Contact-Info	contactId	(U is a chair or PC) OR (R is contact information of chair or a PC member) OR (R is U's contact information) OR (R is contact of U's coauthor) OR (notification deadline has passed, U is on PC, and R is contact of an accepted paper's author)
P5	Paper	leadContactId	(U is R's paper manager) OR (U is a non-conflicted PC member and has submitted a review for R) OR (review discussion has started and U is R's paper manager or a non-conflicted PC member)
R1	Paper-Review	reviewId, paperId, <review content>, reviewSubmitted	(P5 conditions) OR (R is a sub-review and U is the reviewer who asked for it) OR (notification deadline has passed and U is R's author or a non-conflicted PC member)
R2	Paper-Review	contactId, reviewRound, requestedBy, reviewType, ...	(P5 conditions) OR (R is a sub-review and U is the reviewer who asked for it)
Con	Paper-Conflict	paperId, contactId, conflictType	(U is R's author) OR (U is a chair) OR (U is a PC member identified by R's contactId) OR (notification deadline has passed and U is a chair or a PC member)
AO	Paper	Total number of submissions and accepted papers	the notification deadline has passed
AR3	Paper-Review	Average review score across all submitted reviews	U is a PC member
AR4	Paper Review	Number of reviews by each PC member	U is a PC member and statistics excludes each row conflicted with U

TABLE 3.2: Subset of HotCRP policies

Link policy example An author can independently view the names of all PC members, her own paper submission, and the reviews for her papers after the notification date. However, the author is not allowed to see the join of the three columns, which reveals the reviewers' identities. In the HotCRP schema, these columns reside in three different tables (ContactInfo, Paper, and PaperReview). The PaperReview table can be joined with Contact via the contactId key column, and with Paper via the paperId key column. The link policy can be implemented by specifying a restrictive policy for PaperReview.contactId, which does not allow the author to read the column (R2 in Table 3.2). The policy prevents PC authors from identifying reviewers of their own papers, yet allows them to know and participate in discussions with reviewers of non-conflicted papers.

Aggregation policy example During the review and discussion process, HotCRP provides aggregate statistics to all reviewers. The statistics include the average review score across all papers as well as the number of reviews submitted by each PC member. To allow this feature to function correctly, we specify two aggregate policies (AR3 and AR4 in Table 3.2), one allowing an AVG computation on the overAllMerit score field and the other allowing a COUNT on the review field grouped by PC member. In the second case, conflicted papers must be excluded.

Type of change	**Lines of code**
Replace MySQLi with PDO adapter	96
Change paper query	110
Change review query	25
Change comment query	17
Authentication with Qapla	5

TABLE 3.3: HotCRP changes for Qapla

Implementation effort We replaced the MySQLi database adapter [138] normally used in HotCRP with our modified, Qapla-enabled PDO adapter. We modified HotCRP to forward the user authentication credentials to the Qapla reference monitor. (Apache was configured to fork a separate process for each HotCRP user session, so there is a separate instance of the adapter/reference monitor for each session.)

HotCRP uses broad queries and relies on post-filtering to remove the information the user should not see. We changed approximately 150 LoC in HotCRP's code to make these queries policy compliant so that they can work with Qapla. In most cases, we added a couple of queries to identify the contextual information required to convert the broad queries into more specific queries. With Qapla in place, we can remove the post-filtering queries, but we ignored them for now. Table 3.3 summarizes the changes we made in HotCRP.

3.8.2 APPLY

We briefly describe our use of Qapla to protect the application management system (APPLY) for managing faculty, PhD, post-doc, and internship applications in our organization. APPLY's database is similar to the fictitious `Acme` database from Section 3.4 and the confidentiality concerns are also similar. The database contains user accounts for applicants and reviewers, contact and application details of the applicants, references, and internal application review aspects such as comments. Users within the organization are assigned roles based on what application type (intern, PhD, postdoc, faculty) they are allowed to access. APPLY prevents reviewers from accessing applications created before they joined the organization. Additionally, APPLY allows explicit delegation of the right to view (sets of) applications to specific users or roles, and disallows a user from accessing an application in case of a conflict of interest. A single policy condition, listed below, covers a large number of columns across many tables. We refer to this condition with the macro `HAS_APP_ACCESS(U, A)`:

User `U` has access to application `A` if :

(`A` is `U`'s own application) OR

((`U` joined before `A` was submitted) AND

(`U` has no conflict of interest with `A`) AND

((`U` is faculty) OR (`U` has been delegated access to `A`)))

There are additional restrictions on many sensitive columns and exceptions for other roles. For example, users cannot see reference letters written for them and an applicant's country of birth and citizenship cannot be seen by reviewers until the application has been accepted (to prevent discrimination). Office staff can access all applicant names, emails, and postal addresses (to correspond with them) and CVs of accepted applicants (to prepare contracts). In total, we wrote 46 policies for APPLY. The complete list of policies is provided in Section 3.13

Implementation effort APPLY is implemented using Django and Python, and stores its data in a database comprising 36 tables and 202 columns. The modifications necessary for APPLY were quite similar to those required for HotCRP. First, we modified 10 LoC to pass user authentication credentials to the Qapla reference monitor. Second, we changed 63 LoC to remove unused columns from queries to make them compatible with our policies.

3.9 Evaluation

In this section, we present results of an experimental evaluation of Qapla's overhead when used with HotCRP. We also perform a brief security evaluation by injecting HotCRP bugs that existed in older versions.

3.9.1 Experimental setup

All experiments were performed on Dell Precision T1600 workstations with an Intel Xeon E3-1225 3.1Ghz quad core CPU, 8GB main memory, and 10Gbit Ethernet links. The client and server machines were running OpenSuse Linux 12.1 (kernel version 3.1.10-1.29, x86-64). The HotCRP server software consisted of Apache HTTP server 2.4.18, PHP 5.6.15, and HotCRP 2.99. By default, the backend database for each application was MySQL Server 5.7.11. In some experiments, we used instead a well-known commercial DBMS, which remains unnamed due to license restrictions on the publication of benchmark results. Both DBMSes were configured with a query cache of 500MB, unless stated differently. By default, the experimental results correspond to a setup with MySQL with the query cache, unless stated otherwise.

For HotCRP, we used an anonymized database snapshot of a major conference hosted on HotCRP in the past. The database included about 150 submissions, over 400 contacts, and over 700 reviews. The papers were reviewed in 3 rounds.

	Baseline query	**Policy summary**
Q1	select title, abstract from Paper where paperId=X	paper author **or** PC member and the paper is under submission
Q2	select title, overAllMerit from Paper join PaperReview where paperId=X	paper author after notification **or** PC member who is not a conflict and has submitted his/her review and the paper is under submission
Q3	select title, overAllMerit, reviewerName from Paper join PaperReview join ContactInfo where paperId=X	PC member who is not conflicted and has submitted his/her review and the paper is under submission

TABLE 3.4: Microbenchmarks queries

3.9.2 Microbenchmark

The first experiment measures Qapla's latency overhead on individual queries. Qapla introduces overheads associated with query parsing, query rewriting, and executing the rewritten query in the database. Table 3.4 lists the actual HotCRP queries we used in the experiment. Figure 3.3 shows the average query latency over 1000 trials on MySQL and on the commercial DBMS. The Qapla latency is broken down into three components: query parsing ($\textbf{Qapla}_{\textbf{parse}}$), query rewriting ($\textbf{Qapla}_{\textbf{rewrite}}$), and execution of the rewritten query ($\textbf{Qapla}_{\textbf{exec}}$). The error bars show the standard deviation. In this experiment, the query caches of the backend DBMSes and Qapla's template cache were disabled.

The contribution of query parsing and rewriting is small, particularly for the more complex queries (on MySQL, 23%, 15%, 8% of the overall query latency for Q1, Q2, and Q3 respectively). The query rewriting overhead is slightly larger with the commercial DBMS, because Qapla has to translate HotCRP queries, which were written for MySQL, to use a SQL syntax appropriate for that DBMS.

In all cases, Qapla's latency overhead is dominated by the execution time of the rewritten queries. A query rewritten with policy conditions may be significantly more complex than the original query as each table in the query is replaced with a subquery, which may access additional tables that appear in the policy. The efficiency of the rewritten query depends on the database query optimizer being able to generate an efficient query plan. The costs of executing the rewritten queries are

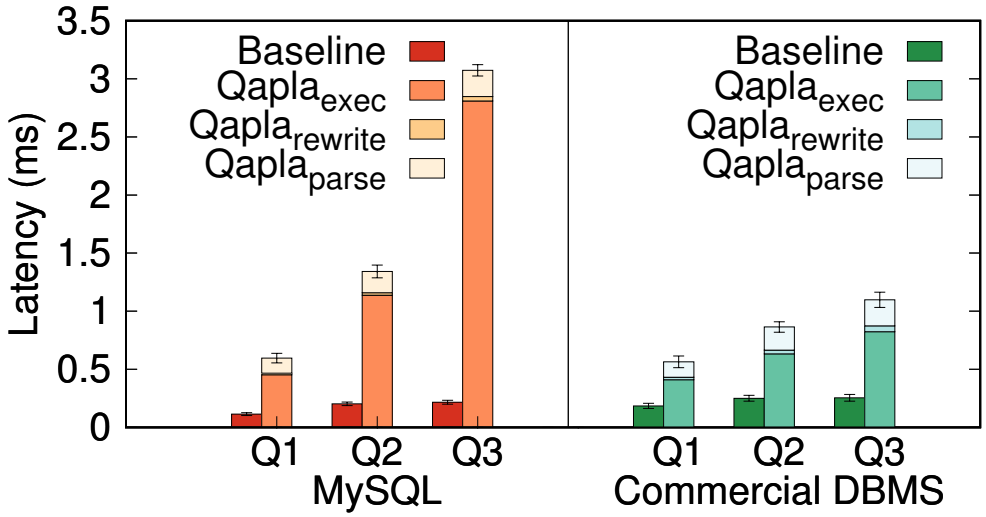


FIGURE 3.3: HotCRP query latency on MySQL and a commercial DBMS (baseline is measured without Qapla).

lower with the commercial DBMS, whose query optimizer likely is more sophisticated than that of MySQL. Thus, while the commercial DBMS has a slightly higher baseline latency, it is able to execute the rewritten queries faster than MySQL, reducing Qapla's overhead substantially for the more complex queries Q2 and Q3.

Our experiment inflates Qapla's true overheads to some extent, because the rewritten query may require accessing tables that are not mentioned in the original query to ensure compliance. HotCRP accesses these same tables in a separate query to perform the filtering in its own code. To understand this further, we measure the overheads for traces of queries corresponding to user actions in the next experiment.

3.9.3 Application latency benchmarks

A user task in HotCRP and APPLY typically involves multiple actions, such as logging in, clicking on a url to visit a page, and clicking on a button to save a form. For each action, the application in turn issues several SQL queries to get the required data for the response and for policy compliance checks. In this section, we measure the overhead for the sequence of SQL queries involved in several application user tasks. We recorded the SQL queries issued for each of the tasks, and replay the query trace with and without Qapla.

We measured the overhead for executing the query traces and the client-perceived latency overhead under various configurations of the baseline and Qapla. **Base** is the

baseline system, **Qapla** is Qapla, and **Qapla**$_{\textbf{t-cache}}$ is Qapla with the template cache enabled. In all configurations, the query cache of the backend DBMS was enabled.

3.9.3.1 HotCRP

In HotCRP, we measured four user tasks: **H1**: As an author, view reviews for a submission (resulting in two actions). **H2**: As a PC member, search for a paper with a keyword, and add a comment (resulting in four actions). **H3**: As PC chair, search for a paper with a keyword, and declare a conflict with a PC member (resulting in five actions). **H4**: As PC chair, invoke the automatic review assignment for all submissions (resulting in three actions).

Task trace execution overhead First, we measured the average time for executing the traces for tasks H1-H4 on MySQL and the commercial DBMS, respectively, under the three configurations and across 1000 trials (all standard deviations are below 5%).

With MySQL, the relative overheads of **Qapla**$_{\textbf{t-cache}}$ are 6x, 4.7x, 5.4x, and 7.8x for the tasks, respectively. With the commercial DBMS, the relative overheads of **Qapla**$_{\textbf{t-cache}}$ are 2.5x, 6.5x, 3.8x, and 2.9x. The results for **Qapla**$_{\textbf{t-cache}}$ show that Qapla's query template cache is effective in reducing the overhead resulting from Qapla's query parsing and rewriting. The template cache hit rates for each action are 25%, 71%, 82%, and 99%, respectively, yielding a reduction in Qapla's overhead of up to 22%, relative to **Qapla**, for H4 with the commercial DBMS. In the case of H1, we observe a net increase in overhead, because the cost of maintaining the template cache cannot be offset due to the low hit rate.

Client-side latency To measure latencies from the perspective of a Web client, we executed each task with a client-side driver that issues HTTP requests to HotCRP for each action involved in performing the task manually. The driver fetches the static HTML pages (but excludes dynamic content such as css, javascripts) from HotCRP and stores them locally. Thus, the experiment includes the overheads of executing PHP code, including database queries, and sending the HTML pages over the network. The template cache as well as the database query cache were flushed after each iteration of a task to fully expose worst-case latency overheads.

Figure 3.4 shows the average latency across 1000 trials of the action with the highest relative overhead in **Qapla**$_{\textbf{t-cache}}$ (all standard deviations are below 0.05%). The latency overheads for the actions are 40%, 25%, 47%, and 320%, respectively.

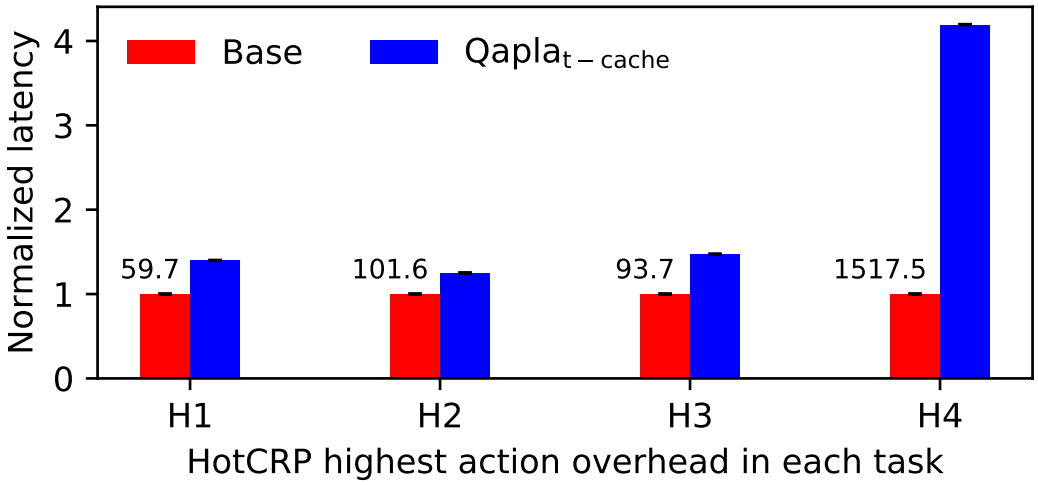


FIGURE 3.4: HotCRP client latency of highest overhead action normalized to **Base**. Labels show **Base** absolute latency numbers in ms.

Note that the overheads are conservative and could be further reduced by removing the redundant post-filtering queries in HotCRP, as discussed in Section 3.8.1.

Most of the latency is due to the PHP execution (including database queries), while the network overhead is minimal (0.2ms on average). All the actions are performed in less than 150ms, except the reviewer assignment generation in H4, which takes 1.5s in **Base** and 6.4s in **Qapla$_{\text{t-cache}}$**. The assignment algorithm invokes ~3780 queries for the given set of papers and reviewers, while the remaining actions invoke less than 45 queries. H4 is a task used by the PC chair(s) only, and normally only a few times per conference, depending on the number of reviewing rounds.

3.9.3.2 APPLY

In APPLY, we measured the following tasks: **A1**: As an applicant, view the status of a submitted application (resulting in 3 actions). **A2**: As the faculty member in charge of post-doc applications, mark the status of multiple applications to reject, and send rejection emails to the marked applications (resulting in 7 actions). **A3**: As a faculty member, search for an applicant by name, and request recommendation letters from the applicant's recommenders (resulting in 7 actions). **A4**: As a student reviewing doctorate applications, see a list of doctorate applications currently under review, and view the details of a single application (resulting in 4 actions).

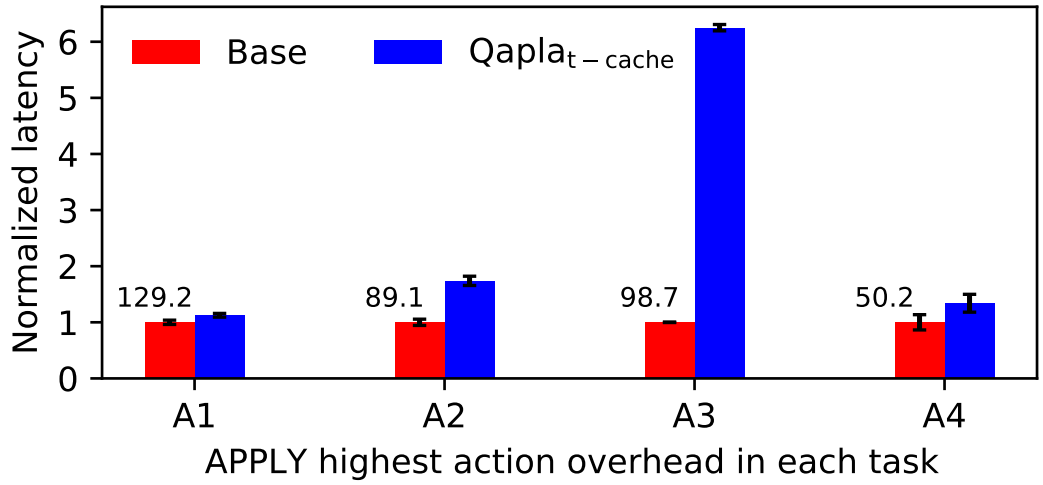


FIGURE 3.5: APPLY client latency of highest overhead action normalized to **Base**. Labels show **Base** absolute latency numbers in ms.

Task execution trace overhead With MySQL, the relative overheads of **Qapla$_{\text{t-cache}}$** are 5.35x, 5.4x, 5.2x, and 4.5x for A1-A4, respectively. With the commercial DBMS, the relative overheads are 4.2x, 3x, 3.3x, and 3.3x, respectively.

Client-side latency Figure 3.5 shows the average latency, across 100 trials, of the action with the highest relative overhead in **Qapla$_{\text{t-cache}}$** (all standard deviations are below 12%). The latency overheads for the four actions are 12.5%, 74%, 6.25x, and 34%, respectively. The high overhead in action A3 is due to a single query with very high runtime, which is the cause of nearly all the overhead. On investigating the query behavior, we found that the performance overhead is due to the MySQL query optimizer's inability to deal with a specific query pattern, possibly because this pattern is unlikely to occur in hand-written queries. When we ran the same query on the commercial DBMS, the overheads came down to approximately 50%.

3.9.4 HotCRP throughput benchmark

For most HotCRP actions, latency is the metric of interest, as it affects user-perceived delays. Right before a submission deadline, however, throughput is also a measure of interest, because many authors re-submit a final revision of their submission within the last minutes before a deadline. To examine the performance under such

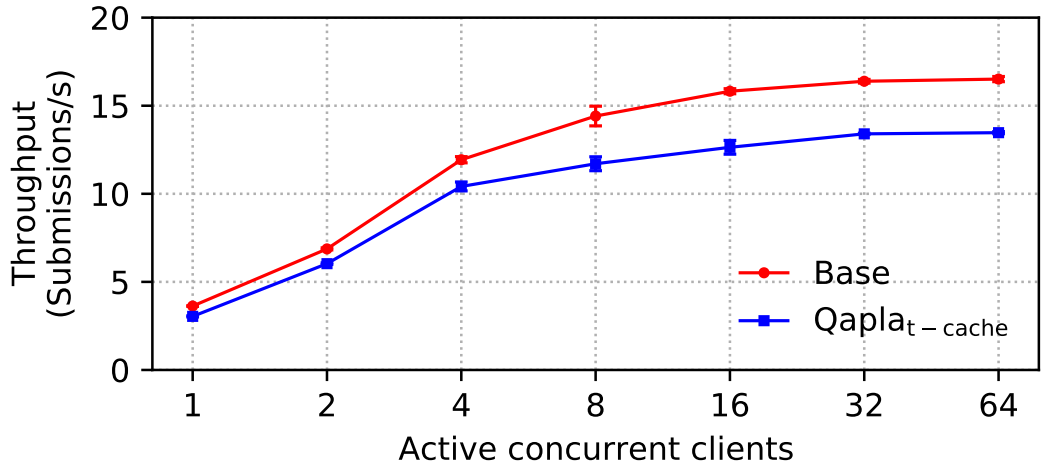


FIGURE 3.6: Submission throughput

conditions, we measured the number of submissions per second HotCRP can sustain with and without Qapla.

In this experiment, clients concurrently upload submissions of size 356KB, which is close to the average submission size in the past HotCRP conference deployment. We varied the number of concurrent clients from 1 to 64. 32 clients were sufficient to saturate the CPU. Prior to the experiment, we cached the entire conference database (∼880MB) in memory. Figure 3.6 shows the number of submissions per second our HotCRP installation can sustain for different numbers of concurrently connected clients. The results were averaged across 3 runs, each of 120s. The error bars show the standard deviation across 3 runs. The overheads are moderate (below 20.2%), and can be compensated by provisioning a somewhat faster server.

3.9.5 Comparison with DBMS access control

Some production DBMS systems support fine-grained access control over tables and views to a limited extent [140, 161]. In this section, we compare using Qapla to enforcing policies directly in our commercial DBMS, which unlike MySQL has some support for fine-grained access control. More precisely, this database supports the equivalent of our single-column policies through a special configuration mechanism. We specified many of the HotCRP policies through this mechanism. However, as shown in section 3.8.1, HotCRP requires richer policies (such as link and aggregate

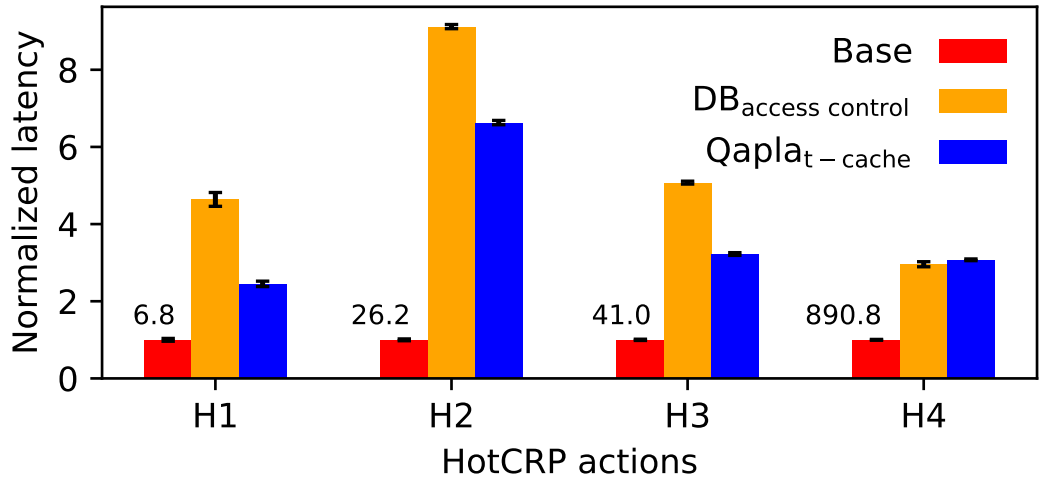


FIGURE 3.7: HotCRP action latency with policies enforced using a commercial DBMS's native support for fine-grained access control, normalized to **Base**. Labels show **Base** absolute latency in ms.

policies), which cannot be expressed using the DBMS's policy mechanism. To enforce these policies, we had to create additional views on all HotCRP tables, restrict access to those views and update all queries, whether compliant or not, to use views rather than the underlying tables.

We ran the experiments from section 3.9.3 to compare the performance of the DBMS access control mechanism with that of Qapla. Figure 3.7 shows the average latency for HotCRP actions, across 100 trials, normalized to **Base**. The error bars show the standard deviation. Qapla policy enforcement overhead is lower than the overhead of enforcing policies through the DBMS access control for most actions.

The results show that using the native support for fine-grained access control in the commercial DBMS is less efficient than Qapla's policy enforcement. Moreover, to get this level of performance from the commercial DBMS, we had to carefully tune its cache configuration for this experiment. Qapla, on the other hand, achieves better performance with both MySQL and the commercial DBMS, has a DBMS-independent policy language, and does not require the use of views and the resulting changes to compliant queries.

Log entry	High-level task reads	Count
Create/update account	User logs in, visits his/her profile	1090
Register, update, submit, or withdraw paper	User logs in, visits the submission page	2082
Added reviewer	Chair logs in, visits the paper's reviewers assignment/conflicts	1335
Set paper lead/shepherd	Chair logs in, visits the paper's page	126
Save/submit/delete review/comment	Reviewer logs in, visits the paper's page	3279
Download paper(s)	Reviewer logs in, visits the paper's page, downloads the paper	2582
Send accept/reject notification	Chair logs in, sends decisions to contact authors	2

TABLE 3.5: Trace actions for HotCRP

3.9.6 Compatibility analysis

We also verified whether HotCRP-Qapla can correctly execute all the user actions performed in a HotCRP deployment across various stages of the conference, and produce the same output as the unmodified HotCRP. We used a trace of anonymized user actions from the logfile of the HotCRP deployment. The logfile contains over 10,000 log entries that correspond to HotCRP database updates. From it, we constructed a trace by inspecting the HotCRP codebase to determine the set of SELECT queries that typically precede a specific update. For example, submitting a review for a submission must have been preceded by viewing the submission page. Since update queries are not subject to policy checks in Qapla, they are not of interest to our experiment and were not included in the trace. Table 3.5 shows the actions performed for each log entry.

The trace consists of actions corresponding to four phases: submissions, review, discussion, and post notification stage. We replayed the entire trace against the original HotCRP and HotCRP-Qapla and compared the outputs. Because the trace is read-only, we replayed it against the final state of the HotCRP database at the end of the conference review period. As a result, several policies were not exercised

the way they would be in a real deployment and, consequently, the outputs of approximately 27% of the actions differed with and without Qapla enforcement (e.g., withdraw link enabled or not, papers may have been withdrawn at a later stage of the conference). Most of these actions were in the first phase. We verified separately that the relevant policies are enforced as expected.

We found that approximately 3% of action outputs differed for other reasons. These reasons are: (i) some non-compliant queries we have not yet modified (e.g., chair unable to make assignments to conflict papers), (ii) policies that are more restrictive than HotCRP assumes (e.g., conflicted PC members unable to download the paper), and (iii) missing policies (e.g., external reviewers not considered).

3.9.7 Security validation

To verify that Qapla is effective at preventing accidental data leaks, we performed fault injection experiments. We manually reviewed HotCRP's change logs for bugs that caused data leaks and other policy violations [75]. We are confident that Qapla can prevent any data leaks that are related to missing or incorrect filtering code in HotCRP, which appear to account for the majority of cases.

As a sanity check, we reproduced two sample bugs HotCRP had in the past. One bug notified authors about changes to PC-only fields during response periods. Another bug allowed PC members to search for papers based on their acceptance status and learn of the acceptance of their papers prematurely. We simulated these bugs by making changes to the policy check functions implemented in HotCRP, or by removing the invocations of these functions at certain places in the application. We executed user actions on the buggy HotCRP application with and without Qapla and manually examined the outputs. We verified that Qapla prevents the data from being revealed to unauthorized parties.

There is one class of data leaks that Qapla cannot prevent by itself, namely when a policy depends on incorrect data recorded in the database. For instance, if HotCRP failed to record the conflicts declared by users correctly in the database, Qapla could not prevent the associated leak. We have not found instances of such bugs in HotCRP's change logs, but it is possible that such bugs might occur.

3.10 Discussion

We discuss some limitations of Qapla's current threat model and ideas on how to strengthen the design to eliminate these threats. We also discuss how Qapla can be used for logging policy violations.

3.10.1 Isolation of the reference monitor

Currently, we assume that the application, which runs in the same address space as the reference monitor, cannot circumvent the reference monitor or steal its authentication credentials. However, this is not a fundamental limitation. To provide guarantees against a malicious application, we can also isolate the reference monitor in a separate process [14, 55], or co-locate it with the DB servers. There are also efficient ways of isolating the reference monitor within the application address space, such as using light-weight contexts [102].

3.10.2 User authentication

Qapla's current design requires the application to specify on which user's behalf it is acting. An application may specify the wrong user to Qapla due to a bug, thus breaking Qapla's policy enforcement. This problem can be easily addressed by having the user authenticate to the reference monitor instead of the application. The application can then ask for the authenticated user's identity from the reference monitor.

3.10.3 Protection against offline linking attacks

Qapla does not protect against offline linking attacks that span multiple queries. For two queries whose results can be linked offline (such as in example 3), randomizing the order of query results may mitigate the attack in some cases. However, randomizing the order of query results cannot eliminate linking attacks in all cases. In particular, some linking may be possible due to information contained in the data itself (e.g., names may have high correlation with the nationality of users, or fine-grained aggregate queries may reveal individual records). We expect the policy designer to be aware of potential data leaks of this type, and design the policies such that compliant queries return a minimum threshold number of results (similar to k-anonymity [158]). Tools to check such conditions on policies can be easily designed.

3.10.4 Support for logging

A natural question is whether we can modify Qapla's reference monitor to detect and log non-compliant queries (e.g., for debugging or auditing). While this is not a design goal, Qapla can be used to detect non-compliant queries to a limited extent – by re-running a query twice, with and without policy checks and comparing the results for any differences. Non-compliant queries can then be logged.

3.11 Related work

We compare Qapla with related work on database access control support in both research and industry, and information flow control systems handling fine-grained database data in applications.

3.11.1 Database access control

The database community has explored fine-grained access and disclosure control within databases, using SQL conditions [96], queries against restricted authorization views [145], and data-derived security views [10]. A formal framework for the design of database access control is presented by Guarnieri *et al.* [66]. In contrast to these systems, Qapla's goal is to provide a portable policy layer that works with existing DBMSs and applications, without relying on any support for policies within the DBMSs.

DataLawyer [165] is a database middleware system that analyzes and rejects non-compliant queries to a relational DBMS. Policies are stated as SQL queries on the database and a usage log, which contains provenance information. DataLawyer supports rich policies, motivated, for instance, by medical databases. Since policies are associated with the entire database, each query must be checked against all policies, each requiring a separate query. Qapla policies are more restricted (e.g., they cannot refer to provenance), but Qapla is much more efficient because policies are indexed by columns. Also Qapla policies are expressed directly as filter conditions, making them easy to write and understand.

In the context of link policies, DiMon [16], its extension D^{2}Mon [162] and Biskup's work [13] enforce access policies by relying on an explicit, complete specification of information that a querier can infer from past queries. These systems deny a query when the query would allow the inference of policy-prohibited information. Qapla's

approach is complementary and easier to implement and use; we require the specification of only access rules, abstracting away the inferences those accesses would allow. If indeed a complete specification of possible inferences were to exist, it could be used to assist the policy designer understand the consequences of Qapla policies.

Turan *et al.* [163] present an algorithm to partition a database schema such that two pieces of data that should not be linked (according to a policy) lie in separate logical subschemas. This could be a useful optimization for certain policies in a Qapla deployment. However, it cannot be used for policies where, of three columns, any two may be linked together, but all three may not be linked simultaneously.

IVD [108] is an authorization system deployed in Facebook, which automatically learns write access control rules on their graph database system from production logs, and enforces them at runtime. Qapla's focus, on the other hand, is on read access control and link policies in relational DBMSs.

SafeD [51] enforces access control policies in a database adapter, similar to Qapla. However, the scope of policies is different in SafeD and Qapla. SafeD supports row-level access control policies on both read and write queries in database applications. Qapla does not support write policies, but it supports fine-grained cell-level access control, linking and aggregate policies.

3.11.2 Access control in production DBMSs

Current production DBMSs support access control at various levels of granularity. However, the extent of support and the language used to express policies varies among DBMSs and, as far as we know, no DBMS can support all of Qapla's policies without requiring changes to either the schema or queries (including queries that are policy compliant). Qapla enforces fine-grained policies without requiring changes to the schema or policy compliant queries, and requires no support for such policies in the backend DBMS. Moreover, as shown in section 3.9.5, Qapla's overhead is lower than a commercial database's native support for fine-grained policies.

Oracle VPD [161] provides extensive support for cell-level access control on tables and views. However, a policy on a table cannot depend on the results of a query on the table itself. Such policies occur in our applications. For instance, the first clause in policy C1 in Table 3.2 checks that the user is the chair, which is defined using the table that the policy protects. Such policies can be enforced in VPD only by either changing the schema or creating additional views. The use of views, in general, also requires changing queries to use the views instead of the underlying

tables. On the other hand, automatic query rewriting as in Pacer is transparent to applications that issue policy compliant queries.

IBM DB2 [147] and SQL Server [155] require a combination of row-level (data-dependent) access control and column masking policies to specify fine-grained policies, which can obscure the policy specification. PostgreSQL [140] has support for row-level policies, but they apply to all columns of a table uniformly. A policy on a subset of columns requires the creation of a view containing only those columns. MySQL and MariaDB do not support data-dependent access control. Fine-grained access control in these DBMSs requires creating a separate view for every group of users with the same privilege, or creating stored procedures and granting privileges to users to execute the procedures [77, 141].

In all production DBMSs we know of, enforcing link policies requires creating a separate view for each policy. Transformation and aggregation policies require separate views or stored procedures. As mentioned above, creating additional views or using stored procedures requires significant changes even to applications that issue only policy-compliant queries.

3.11.3 Database interposition

Interposing on database queries to improve security is a common technique. Perhaps most closely related to our work is CLAMP [134], which has the same goals as Qapla. However, CLAMP's architecture and policy language are different from Qapla's. In CLAMP, when a user initiates a session, the enforcement framework performs user authentication, instantiates a logical view of the database restricted only to data that the user can access (based on applicable policies), and isolates a fresh instance of the application in a virtual machine, restricting it to only communicate with the authenticated user and giving it access to only the logical view of the database via query interposition (as in Qapla). CLAMP's design supports a stronger threat model than Qapla's current prototype—CLAMP isolates user sessions from each other and from the reference monitor, and does not rely on the application to authenticate the user (see Section 3.10)—but the expressiveness of policies, which is really the focal point of our work, is limited in CLAMP. CLAMP only supports per-table policies, which specify the rows that each user has access to. Support for finer policies that differentiate columns of a table from each other or take into account linking, transformation and aggregation is missing in CLAMP. Qapla can be

strengthened with CLAMP's isolation and authentication techniques in a straightforward manner.

Diesel [55] is a framework for applying the principle of least privilege on relational databases. Diesel policies specify subsets of a database that each application *module* can access. For example, a policy may specify that a user-facing module can only access the Users table, but not administrative tables, thus limiting damage in the event of a user session compromise. This is very different from Qapla's (and CLAMP's) goal of specifying what data each *user* can access. Nonetheless, Diesel also relies on query interposition (as in Qapla) to enforce its policies.

Passe [14] hardens the web framework Django to isolate application modules from each other. Like Diesel, it uses query interposition to enforce least privilege on data accessible to each module. Unlike Diesel, but like CLAMP and Qapla, Passe's policies are sensitive to the authenticated user. However, Passe's policies are fundamentally different from those of Qapla, CLAMP and Diesel—they enforce data-dependency relations on query parameters. For example, a Passe policy may enforce that the third parameter of the second query made by a specific application module is always a value returned for the first query of the module. Moreover, Passe's policies are not specified by administrators. Instead, they are learnt by automated testing in an offline phase. This learning can have both false positives (it may learn a policy that is too restrictive) and false negatives (it may not learn a required policy). Due to the very different nature of Passe's policies, it is not possible to directly compare their expressiveness to that of Qapla's policies.

3.11.4 Policy specification frameworks

EPAL [7] and XACML [50] are platform-independent frameworks that enable designing declarative policy specifications for enterprise and web applications, respectively. They enable specifying fine-grained policies in terms of users, data, actions, system environment. However, both EPAL [7] and XACML [50] are frameworks, which need to be implemented by each application separately. In contrast, Qapla is a full system with a generic specification framework that can be used for any application backed by a relational DBMS. Unlike EPAL and XACML, Qapla does not support purpose as a first-class element in the policy specification, and instead relies on authentication-based access control. Finally, EPAL and XACML support policies on high-level user actions in the system, such as view, edit, delete, whereas Qapla specifies polices on operations within individual application queries.

Other systems, such as Soutei [139], SecPAL [9], and Guardat [166] have used Datalog [97] language to develop declarative specifications. For instance, Guardat uses a domain-specific policy language based on Datalog to support policies on files based on users, roles, system time, and state and content of files. Qapla instead uses SQL syntax to specify policies, similar to [29, 96]. SQL is a natural choice to specify policies for database-backed applications, since it enables specifying complex policies on query operators easily, and developers are already familiar with it.

3.11.5 CMS confidentiality

CoCon is a new conference management system whose confidentiality properties were verified formally in the Isabelle proof assistant [85]. On the other hand, Qapla is a general, language-independent runtime compliance layer for database queries, which we have used to enforce compliance in an existing and widely used conference management system, HotCRP.

3.11.6 Privacy in statistical databases

Differential privacy [42] and privacy-preserving queries [24, 111] are focused on statistical databases, where only statistical information, but no information about individual records, should be revealed. Qapla instead focuses on applications that require access to specific database records, subject to fine-grained policies.

3.11.7 Information Flow Control

UrFlow [29], Hails [60], Jacqueline [187], DBTaint [37], RESIN [190], LabelFlow [28] and Nemesis [35] use language-based techniques to enforce information flow control in web applications written in specific languages. In contrast, Qapla can be ported to any language easily but it enforces access policies, not information flow control. Qapla can be integrated with a language-based technique to control information flow with fine-grained policies.

IFDB [150] enforces authorization policies by modifying the PostgreSQL database engine, as well as the application environments in PHP and Python. For enforcing column policies, IFDB relies on declassifying views. Row policies are specified with secrecy and integrity labels, which are associated with database records. IFDB enforces row policies by tracking the labels through the application process and stored

procedures. Qapla specifies all policies using one mechanism. Qapla's enforcement uses query rewriting and is database-agnostic.

Sif [30], SeLINQ [149], and Li *et al*. [99] assign labels or security types to database columns, and use security-typed programming languages to write restricted query interfaces to the database and the application code. However, these systems cannot enforce data-dependent policies. Furthermore, some of these systems [99, 149] rely on programming applications in languages that integrate database query mechanisms. While the current prototype of Qapla focuses on applications using SQL to query databases, it can be easily extended to protect applications using other programming paradigms for database queries. Qapla does not impose any restrictions on the programming language for the applications themselves.

3.12 HotCRP policies specified in Qapla

Table 3.6 below lists the complete set of policies that we implemented on the schema of HotCRP 2.99 application and used in Qapla's evaluation. The schema consists of 22 tables, but the workflows in our evaluation did not involve two tables. So we only specified policies on 20 tables. The first column provides a policy identifier, which is used only for ease of exposition and reference. The second and third columns respectively indicate the table and the set of the table's columns in the HotCRP schema that the policy applies to. The last column explains the policy in English. The actual policy is written in Qapla's SQL-like syntax (Section 3.4) and follows the structure of the English description.

id	**table**	**column list**	**allow the authenticated user U access to row R if ...**
C1	Contact-Info	name, affiliation	(U is a chair) OR (R is U's contact information) OR (R is contact of U's coauthor) OR (U and R are on the PC) OR (R is contact of a PC member) OR (notification deadline has passed, U is on PC, and R is contact of an accepted paper's author)

id	table	column list	allow the authenticated user U access to row R if ...
C2	Contact-Info	email	(U is a chair) OR (R is U's contact information) OR (R is contact of U's coauthor) OR (U and R are on the PC) OR (notification deadline has passed, U is author of an accepted paper, and R is contact of a paper's shepherd)
C3	Contact-Info	contactId	(U is a chair or PC) OR (R is contact information of chair or a PC member) OR (R is U's contact information) OR (R is contact of U's coauthor) OR (notification deadline has passed, U is on PC, and R is contact of an accepted paper's author)
C4	Contact-Info	password	(R is U's contact information) OR (U is a chair)
C5	Contact-Info	passwordTime, fax number	(R is U's contact information)
C6	Contact-Info	last login, contact tags, collaborators	(R is U's contact information) OR (U is on the PC and R is info of a PC member)

id	table	column list	allow the authenticated user U access to row R if ...
P1	Paper	paperId, title, abstract, timeSubmitted, timeWithdrawn	(U is R's author) OR (submission deadline has not passed and U is on the PC) OR (submission deadline has passed, U is on PC, and R was submitted fully)
P2	Paper	authorInformation, collaborators	(U is R's author or chair) OR (notification deadline has passed, R was accepted and U is on the PC)
P3	Paper	outcome	(notification deadline has passed and U is R's author or chair or PC) OR (U is R's paper manager or a non-conflicted PC member)
P4	Paper	shepherdContactId	(notification deadline has passed and U is R's author or chair) OR (U is R's paper manager or a non-conflicted PC member)
P5	Paper	leadContactId	(U is R's paper manager) OR (U is a non-conflicted PC member and has submitted a review for R) OR (review discussion has started and U is R's paper manager or a non-conflicted PC member)
P6	Paper	managerContactId	(U is a chair or R's manager or a non-conflicted PC member)

id	table	column list	allow the authenticated user U access to row R if ...
P7	Paper	paperStorageId, size, paper, <other paper metadata>	(U is R's author or chair) OR (submission deadline has passed, R was submitted fully, and U is chair or PC)
AO	Paper	{*JS*: {}, *LS*: {outcome, timeSubmitted, timeWithdrawn, Paper.*[COUNT]}}	the notification deadline has passed
PS	Paper-Storage	paperId, paper, <other paper metadata>	(U is R's author or chair) OR (submission deadline has passed, R was submitted fully, and U is chair or PC)
R1	Paper-Review	reviewId, paperId, <review content>, reviewSubmitted	(P5 conditions) OR (R is a sub-review and U is the reviewer who asked for it) OR (notification deadline has passed and U is R's author or a non-conflicted PC member)
R2	Paper-Review	contactId, reviewRound, requestedBy, reviewType, reviewEditVersion, reviewToken, timeRequested, commentsToPC, <other fields only for PC>	(P5 conditions) OR (R is a sub-review and U is the reviewer who asked for it)
AR3	Paper-Review	{*JS*: {}, *LS*: {contactId, overAllMerit[AVG]}}	(U is R's paper manager or chair or a PC member)

id	table	column list	allow the authenticated user U access to row R if ...
AR4	Paper-Review	{*JS*: {}, *LS*: {contactId, reviewSubmitted, reviewNeedsSubmit, PaperReview.{*}[COUNT]}}	(U is R's paper manager or chair or a PC member and statistics excludes each row conflicted with U)
AR5	Paper-Review	{*JS*: {}, *LS*: {paperId, reviewSubmitted, reviewNeedsSubmit, PaperReview.{*}[COUNT]}}	same as AR4
Con	Paper-Conflict	paperId, contactId, conflictType	(U is author of the paper identified by R's paperId) OR (U is a chair) OR (U is a PC member identified by R's contactId) OR (notification deadline has passed and U is chair or a PC member)
PT	Paper-Tag	paperId, tag, tagIndex	(U is a chair or a PC member or manager of the paper identified by paperId)
RR	Review-Rating	<all columns>	similar to R2, but specified on ReviewRating table
Com1	Paper-Comment	contactId, replyTo	(R is a submitted comment and U is R's paper manager or U wrote R) OR (review discussion has started, R is a submitted comment, and U is R's paper manager or non-conflicted PC member)

id	table	column list	allow the authenticated user U access to row R if ...
Com2	Paper-Comment	commentId, comment, paperId, paperStorageId, commentType, commentTags, commentRound, commentFormat, timeModified, timeNotified	(Com1 conditions) OR (notification deadline has passed, R is a submitted comment, R is marked for authors, and U is author of the paper identified by R's paperId)
TI	Topic-Interest	contactId, topicId, interest	(U is chair) OR (R is U's interest entry)
Top	Paper-Topic	topicId, paperId	(U is author of R's paper) OR (R's paper is fully submitted and U is chair or a PC member)
AL	Action-Log	logId, contactId, paperId, time, ipaddr, action	U is R's paper manager or a non-conflicted chair
ML	Mail-Log	<all columns>	U is chair
RReq	Review-Request	<all columns>	U is R's paper manager or a non-conflicted chair or a PC member who requested for the review R
PRR	Paper-Review-Refused	<all columns>	Similar to RReq, but on PaperReviewRefused table
PRP	Paper-Review-Preference	<all columns>	U is R's paper manager or a non-conflicted chair or a non-conflicted PC member who submitted preference R

id	table	column list	allow the authenticated user U access to row R if ...
PO	Paper-Option	<all columns>	U is R's paper manager or chair or a PC member or author of the paper with options R
PW	Paper-Watch	<all columns>	U is chair or a PC member
S	Settings	<all columns>	public
F	For-mula	<all columns>	public
T	Topic-Area	<all columns>	public

TABLE 3.6: Set of HotCRP policies

3.13 APPLY policies specified in Qapla

Table 3.7 below lists the complete set of policies that we implemented on the schema of APPLY application and used in Qapla's evaluation. A common policy fragment that appears in many policies is denoted by the macro `HAS_APP_ACCESS`(U, A), which expands to the following conditions:

User `U` has access to application `A` if :

(`A` is `U`'s own application) OR

((`U` joined before `A` was submitted) AND

(`U` has no conflict of interest with `A`) AND

((`U` is faculty) OR (`U` has been delegated access to `A`)))

id	**table**	**column list**	**allow the authenticated user U access to row R if ...**
A1	fa_application	cv	`HAS_APP_ACCESS`(U, R) OR (U is a member of the office staff and the application is accepted)
A2	fa_application	user_id, phone, address	`HAS_APP_ACCESS`(U, R) OR U is a member of the office staff
A3	fa_application	status, app_type, archived	U is a member of the review committee or the office staff
A4	fa_application	id, created_date, updated_date, minor_status, url, rejec-tion_sent_date	`HAS_APP_ACCESS`(U, R)
Deg	fa_degreeinfo	<all columns>	(R corresponds to an inter app and `HAS_APP_ACCESS`(U, app)) OR (R corresponds to a PhD app and `HAS_APP_ACCESS`(U, app))
DA	fa_doctorate-application	<all columns>	`HAS_APP_ACCESS`(U, R)
FA	fa_faculty-application	<all columns>	`HAS_APP_ACCESS`(U, R)
IA1	fa_internship-application	application_id, birthday, under-graduate_id, field_id	`HAS_APP_ACCESS`(U, R)
IA2	fa_internship-application	birth_country, birth_city, citizenship	R corresponds to an application, the application is accepted, and `HAS_APP_ACCESS`(U, app)

id	table	column list	allow the authenticated user U access to row R if ...
Tag	fa_personaltag	id, application_id, user_id	R's user_id corresponds to U and HAS_APP_ACCESS(U, R)
Ref1	fa_reference	id, applicant_id, reference_id, status, first_sent_date, last_sent_date, tempname	R is a reference for an app and HAS_APP_ACCESS(U, app)
Ref2	fa_reference	letter	U joined before R's corresponding application was submitted and U has no conflict of interest with app and (U is faculty or U has been delegated access to app)
Ref3	fa_reference	direct_web_key	U is the sysadmin
EB1	fa_email-binding	activation_key	R's user_id corresponds to U
EB2	fa_email-binding	id, user_id, email	(R is the contact info of U's referrer) OR (R is the contact info of an application's referrer and HAS_APP_ACCESS(U, app)) OR (U is R's owner)
RER	fa_reference-emailrequest	id, requestor_id, reference_id, created_date	U is a faculty and R is a reference for an application and HAS_APP_ACCESS(U, app)

id	table	column list	allow the authenticated user U access to row R if ...
SF1	fa_specificfield	id, abbrev, fullname, faculty_pot, intern_phd	public
SF2	fa_specificfield	priority	U is a member of the review committee
TA	fa_trackapp	id, user_id, application_id	`HAS_APP_ACCESS`(U, R)
AS1	fa_apptype-settings	id, app_type, from_addr, reference_sub, reference_body, rejection_sub, rejection_body	U is a member of the review committee or office staff
AS2	fa_apptype-settings	letter_deadline, reference-_deadline_type	U is a member of the review committee
Role	fa_role	id, user_id, role_id	(U is R's owner) OR U is a faculty
RR1	rbac_rbacrole	id, name	same as Role
RR2	rbac_rbacrole	desc, order	public
Conf	fa_conflicts	id, user_id, application_id	U is a faculty
CF	django-_comment-_flags	id, user_id, comment_id, flag, flag_date	U is a sysadmin OR (U is R's owner and R is a comment on an app and `HAS_APP_ACCESS`(U, R))

id	table	column list	allow the authenticated user U access to row R if ...
DF	fa_delegation-_filters	id, delegatee_id, delegator_id, is_revoked, ... <all columns>	U is a faculty OR (R is delegated to U and R is not revoked)
AD	fa_approved-_delegations	id, delegatee_id, is_revoked, app_id, ... <all columns>	similar to DF
AU1	auth_user	id, first_name, last_name, email	U is R's owner OR U is a member of the office staff OR (R is info of an applicant and HAS_APP_ACCESS(U, app)) OR (R is info an applicant's referrer and HAS_APP_ACCESS(U, app))
AU2	auth_user	username, password, date_joined, ... <other columns>	U is R's owner OR U is the sysadmin
DCom	django-_comments	id, object_pk, user_name, user_email, comment, submit_date, is_public	R corresponds to an app and HAS_APP_ACCESS(U, app) and (U is a faculty or R is a public comment)
Sys	<all other tables>	<all columns>	U is the sysadmin

TABLE 3.7: Set of APPLY policies

Chapter 4

Pacer: Network Side-Channel Mitigation in the cloud

This chapter describes the design, implementation, and evaluation of Pacer, a system to mitigate a specific class of side-channel disclosures, namely the network side-channel disclosures. We start with a discussion of how network side-channel disclosures work (Section 4.1). We then describe our threat model (Section 4.2), and our key ideas for designing a secure and efficient mitigation solution (Section 4.3). We present our novel *cloaked tunnel* abstraction (Section 4.4), Pacer, our paravirtualized cloaked tunnel implementation for an IaaS cloud that requires modest changes to a hypervisor and guests (Section 4.5), and a gray-box profiler that automatically generates traffic shapes for a guest (Section 4.6). We describe Pacer's implementation in Section 4.7 and our experimental evaluation in Section 4.8. In Section 4.9, we discuss potential extensions to Pacer's design to mitigate additional leaks and for computing more efficient traffic shapes. Finally, we discuss related work in Section 4.10.

4.1 Network side channels

We first discuss the high-level concept of a network side-channel attack. We then demonstrate a proof-of-concept attack in cloud settings, which can leak sensitive information, such as the videos streamed.

4.1.1 Background

As briefly discussed in Section 2.4, a network side-channel disclosure, like all other side-channel disclosures, involves two steps. Here we elaborate on how network side-channel disclosures work.

First, an adversary observes the shape of the victim's traffic. An adversary with access to network elements like links, switches, or routers can observe the traffic *directly*. An adversary without direct access can still observe victim traffic *indirectly* if they can control attack traffic that shares bandwidth with the victim's traffic. Specifically, when bandwidth is shared, then regardless of the queueing discipline, available bandwidth and queueing delays observed by one flow are influenced by concurrent flows, and therefore can be exploited for a side-channel attack. An adversary can easily rent VMs in an IaaS cloud to indirectly measure the victim's traffic at a shared server's network card or a top-of-the-rack switch. Such indirect measurements significantly lower the barrier for mounting network side-channel attacks, which would otherwise be limited to adversarial network operators or other parties with direct access to a victim's network traffic. Indeed, prior work has demonstrated attacks based on indirect observations of the victim's traffic [4, 144, 146].

Extensive prior work has shown that various features of traffic shape can be observed thus; these features may include total number and sizes of packets [26, 36, 157], their timing [20, 63], more coarse-grained features such as burst lengths, frequency of bursts and burst volumes [43, 176, 180], as well as a combination of such features [67, 100, 148].

Once an adversary has the traffic shape, it can then infer the victim's secrets. Traffic shape has been shown to reveal rich information about the underlying communication, including what a user is typing [153], what webpage is being visited [73, 172] and which video is being streamed [146], what phrases are being used in VoIP conversations [180], and even the private keys of communication partners [17, 18]. Chen *et al*. [25] demonstrate that even more sensitive information like users' medical conditions, family income, and investment secrets [25] can be gleaned from the shape of the encrypted traffic of healthcare, taxation, investment, and web search services implemented on software-as-a-service offerings.

4.1.2 Attack demonstration

To demonstrate the viability of network side channel attacks through indirect measurements in cloud settings, we set up an attack where an adversary exploits the signals in the queueing delays for its own traffic to infer the victim's traffic shape. Further, using a convolutional neural network (CNN) [64], the adversary can recognize streamed videos from the traffic shapes with high accuracy.

4.1.2.1 Experimental setup

We set up two VMs, a victim and an attack VM, on two separate CPU sockets of a Dell PowerEdge R730 server machine (S_1). The VMs use Xen's virtualized network stack; thus all traffic is routed through the netback driver and the TCP stack in dom0 of the hypervisor. We configure S_1's shared NIC with a bandwidth of 1Gbps, and the hierarchical token bucket (HTB) queueing discipline [72]. We further create two separate HTB traffic classes for (i) the attack traffic, and (ii) the victim traffic and rest of the traffic through the host. We configure the attack traffic to have a lower priority than all other traffic.

The victim hosts a custom video streaming service on top of Apache, which serves video segment files in response to client requests. A custom video client runs on a second server (S_2), and requests the video segments sequentially over HTTPS. The attack VM runs a UDP client that sends short payloads (56 bytes) to a UDP server on a third machine (S_3), which logs the packet arrival timestamps and echoes back the packets to the client. S_2 and S_3 have 10Gbps NICs and all machines are connected to a 10Gbps switch; thus the bottleneck link is the shared NIC at S_1. The attack client maintains a send window of 4,500 packets (computed based on the bandwidth-delay product for the NIC), which ensures that some attack packets are always queued at the bottleneck link without overflowing the queue.

We streamed 10 videos at 720p resolution from a YouTube dataset (a detailed description of the dataset is given in Section 4.8) for upto 30 segments. Segments take less than 0.02s to download, and segments within a video are requested at an interval of 5s. We streamed each video 150 times. During each video stream, we log the series of arrival timestamps of the adversarial client's packets at the adversarial server. We label each time series of the adversary's packet arrival timestamps with the id of the video streamed by the victim. Thus, we have 1500 time series of adversary's packet arrival timestamps with 10 distinct labels.

4.1.2.2 Analysis

We aggregated each time series into windows of 50ms, and generated a time series of the adversary's transmitted packet count in each window. The packet count is the number of packet arrival timestamps recorded in each time window. Finally, we normalized each packet count time series using min-max normalization [117].

Next, we implemented a CNN classifier to train on the time series of normalized packet counts. Figure 4.1 shows the architecture of our classifier, which consists

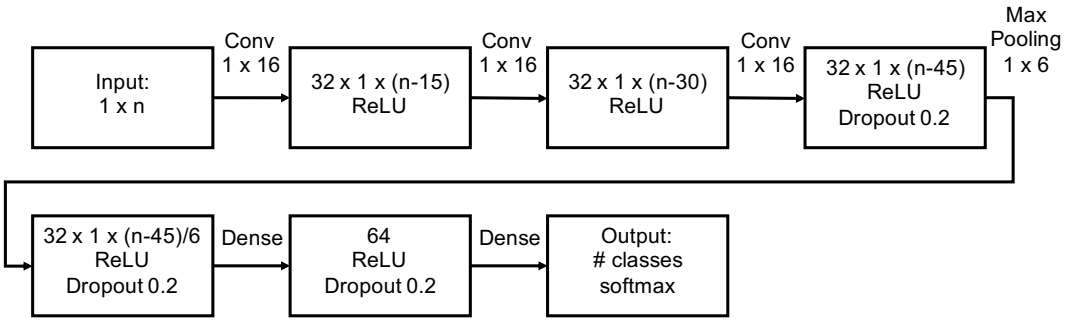


FIGURE 4.1: CNN architecture. n denotes the number of elements of one time series, which is the total time of the time series divided by the window size (50ms).

of three convolution layers, a max pooling layer, and two dense layers. We use a dropout of 0.2 between each pair of hidden layers of the classifier as shown in the figure. We train the classifier with an Adam optimizer [88], categorical cross-entropy error function, and with input batches of 64 samples. Our CNN classifier is similar to the one built by Schuster *et al.* [146, section 7.2], with the difference that we used a dropout of 0.2 between the model's hidden layers and 64 epochs for training.

We implemented the classifier using Tensorflow 2 API and with the Keras [87] frontend. We used 70% of the time series data for each label (video) for training and the remaining for evaluating the classifer. The classifier achieves an overall precision and recall of 81.8% each, and an overall accuracy of 96.4%.

Our experiment confirms prior work [4, 17, 18, 73, 133, 146, 153, 172, 180] and shows that a network side-channel attack can identify videos in a collection with good accuracy. While an attack in a production environment faces additional challenges like achieving co-location with the victim, prior work has shown that it is easy to attain co-location [78, 79, 144]. Hence, cloud tenants that require strong confidentiality have to consider that network side channels are a realistic threat.

4.2 Threat model

Figure 4.2 summarizes Pacer's threat model. Pacer's goal is to prevent network side-channel leaks of a cloud tenant's secrets to anyone able to rent other VMs in the cloud. Prior work has shown that it is easy to attain co-location with a victim VM [78, 79, 144]. Accordingly, we assume a strong adversary that may co-locate its VMs with

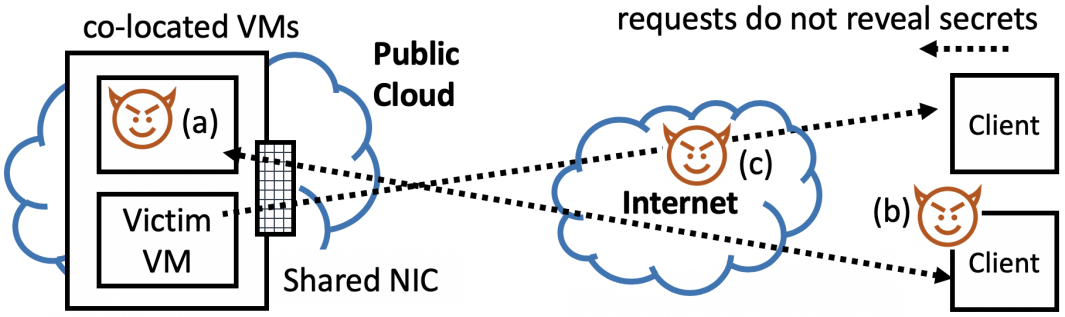


FIGURE 4.2: The adversary can (a) co-locate VMs with victim's VM in the cloud, (b) control clients of its own VMs, and (c) use cross-traffic between any pair of these to infer the shape of the victim's traffic at shared network links.

the victim's VM and indirectly infer the shape of the victim's outbound traffic by observing contention with its own cross-traffic. In particular, the adversary may use this method to infer the traffic shape of the victim at shared network elements in the common server, rack, or datacenter. The adversary has access to all the services available to IaaS guests, including the ability to time the transmission and reception of its own network packets with high precision, which would enable it to perform the indirect measurements as described in Section 4.1. The adversary controls network clients, which may communicate freely with the adversary's VMs. However, the adversary cannot access the victim's private network or impersonate/compromise the victim's clients. While not the goal of our work, Pacer's design also protects against powerful adversaries with the ability to directly observe the victim's traffic as well as delay, drop, and inject network packets.

The *victim* in the public IaaS cloud is a tenant executing arbitrary computations in one or more guest VMs but not invoking other guest VMs or cloud backend services. The victim's goal is to protect its secrets, which can be reflected in parameters of client requests (*i.e.*, secret inputs, such as the name of a requested file), or in the victim's internal state (*e.g.*, which request handlers are cache hot because they were accessed recently). The victim serves a set of trusted clients that connect to its VMs from outside the cloud via a secure Virtual Private Network (VPN) or from within the cloud via a Virtual Cloud Network[1]. Tenants may require a second level of authentication to separate clients' privileges, but this is not relevant for Pacer's security.

[1]In principle, Pacer can support multi-tier guests and also guests that provide an open service to untrusted clients, but these extensions are beyond the scope of this thesis and not implemented in our prototype. Section 4.9 discusses the extension to multi-tier guests.

We assume that the victim rents an entire server socket for exclusive use and uses main memory within the NUMA domain associated with this socket to mitigate leaks to co-located tenants via shared memory buses, caches, and CPU microarchitectural state (these side channels are not the focus of this work)[2]. We also assume that the victim partitions its workload independent of secrets.

Our focus is on server-side security; protecting client privacy is a non-goal. Moreover, we assume that client traffic reveals no secrets through its shape (its length, number of packets, or timing)[3]. This implies that the size and time of *client requests* do not depend on any secrets or the actual completion times of previous responses. A victim can design the content being served to meet this requirement, e.g., in the case of a web server by in-lining embedded objects, and using Javascript to decorrelate (in time) requests for embedded links and user requests that may depend on previously served content.

Hiding the identity of the service requested [43, 67, 73] or the communication protocol used [44, 179] is also not the focus of our work, although Pacer can be used to address these goals too.

4.3 Key ideas

A principled approach to avoiding network side-channel leaks is to shape the network traffic so that it cannot reveal secrets. If done naïvely, shaping can be very costly in terms of bandwidth or latency when the payload traffic is bursty. Pacer exploits the following key ideas to reconcile security and efficiency.

Per-guest dynamic shaping Pacer shapes each guest's network traffic dynamically according to the guest's prevailing workload, thus enabling dynamic sharing of the available network capacity among different guests for efficiency. This requires that the presence and time of requests reveal no secrets, as assumed in our threat model.

Secret-independent shaping Instead of insisting on a uniform traffic shape for all of a guest's network traffic, Pacer allows the shape to vary, as long as the variations don't depend on secrets. For instance, if the type of content being requested from a server (e.g., document vs. video) is not a secret, then a different traffic shape can be

[2]In principle, this assumption can be relaxed by using complementary work to mitigate side-channel leaks through shared memory buses, caches, and CPU microarchitectural state [15, 169].

[3]This assumption can be avoided by shaping client traffic. In principle, Pacer can support this by running a hypervisor on the client side, as discussed in Section 4.9.

used for the two. This additional degree of freedom helps minimize overhead for variable network traffic while preventing leaks of secrets.

Gray-box profiling Dynamic traffic shaping requires an understanding of how a guest's public inputs affect its network traffic. This information can be obtained via program analysis, but it is difficult to perform program analysis on arbitrary binaries running in a VM. Black-box profiling can be performed on arbitrary guests, but cannot reliably discover all dependencies and, therefore, is not secure. Pacer instead relies on gray-box profiling, which requires no knowledge of a guest's internals beyond an explicit traffic indicator from the guest. This indicator partitions the guest's possible network interactions independent of secrets and indicates the onset of a particular interaction. It is used by Pacer in two ways: (i) to profile the guest's network interactions and generate a transmit schedule for each partition, and (ii) to instantiate a transmit schedule for a network interaction. As long as a guest computes the indicator independent of secrets, the choice of indicator may affect performance but not security. We discuss Pacer's profiling in further detail in Section 4.6.

Paravirtualized support for traffic shaping Pacer provides paravirtualized hypervisor support that enables guests to implement a cloaked tunnel, while adding only a modest amount of code to the hypervisor. A performance-isolated shaping component in the hypervisor initiates transmissions based on a schedule. If no payload is available at the time of a packet's scheduled transmission, the shaping component transmits a dummy packet instead. To the adversary, this dummy is indistinguishable from a payload packet as the traffic is encrypted. Pacer's cloaked tunnel abstraction and paravirtualized design are discussed in Sections 4.4 and 4.5, respectively.

4.4 Cloaked tunnel

Pacer's key abstraction is a *cloaked tunnel*, which ensures that the shape of network traffic inside the tunnel is secret-independent, thus defending against adversaries who can observe, directly or indirectly, traffic inside the tunnel. In this section, we describe the tunnel and its security independent of a specific application setting, implementation, or placement of tunnel entry and exits. In Section 4.5, we evolve the design to work within the constraints of a realistic cloud environment.

4.4.1 Tunnel requirements

We begin with the requirements for a cloaked tunnel.

T1: Secret-independent traffic shape Packet transmissions must follow a schedule that does not depend on secrets; actual transmission times must not be delayed by potentially secret-dependent computations.

T2: Unobservable payload traffic The traffic shape must not reveal, directly or indirectly, an application's actual time and rate of payload generation and consumption. This implies that flow control must not affect the traffic shape; that padded content must elicit the same response (e.g., ACKs) from receivers as payload data; and that packet encryption must encompass the padding. This in turn requires that padding be added at or above the transport layer, while encryption be done below the transport layer.

T3: Congestion control The tunnel must react to network congestion. It must pause transmissions when the network is congested, and resume only when the congestion is eliminated. Congestion control is needed only for network stability and fairness; it does not reveal secrets, since the tunnel reacts to network conditions, which themselves depend only on already shaped and third-party traffic.

4.4.2 Architecture

Figure 4.3 shows the cloaked tunnel's architecture, which addresses the requirements of the previous subsection. The tunnel protocol stack runs on both tunnel endpoints. (Only one of the two symmetric endpoints is shown in the figure.) The tunnel protocol stack consists of a *shaping* layer on top of a modified transport layer (e.g., TCP) on top of the encryption layer[4]. These three layers rest on top of conventional IP and link layers. Each tunnel is associated with a flow identified by a 5-tuple of source and destination IP addresses and ports, and the transport protocol.[5]

The shaping layer initiates transmissions according to a schedule and pads packets to a uniform size. It interacts with applications via a set of shared, lock-free queues. The layer takes application data from a per-flow outbound queue and transmits it in the tunnel. It places incoming data from the tunnel into a per-flow inbound queue. Finally, it receives traffic indicators and per-flow cryptographic keys (to be used by the encryption layer) via a per-application command queue.

[4]The encryption could be done using IPSec protocol [86].
[5]We describe the tunnel in terms of TCP; however, another stack like QUIC [91] can also be used.

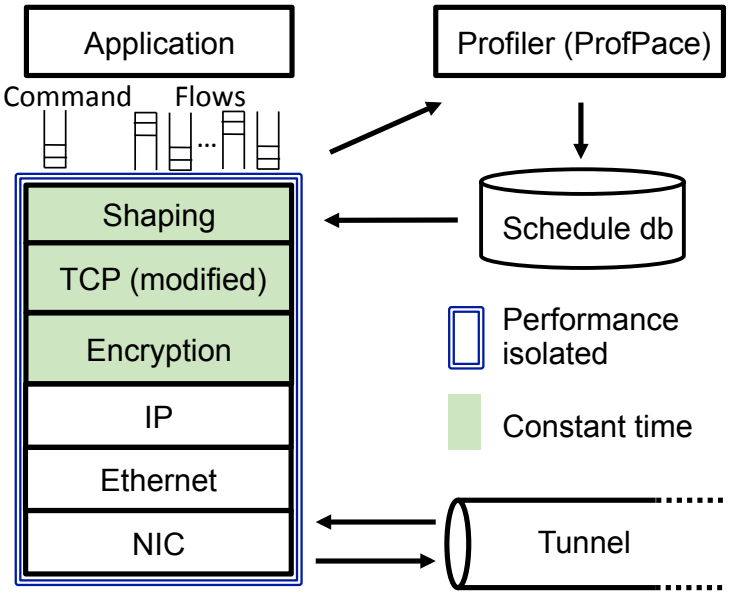


FIGURE 4.3: Cloaked tunnel (one endpoint)

A separate, user-level *gray-box profiler* (ProfPace) analyzes timestamps and traffic indicators collected by the tunnel, and generates and updates transmit schedules in the schedule database. ProfPace is described in Section 4.6. Below, we describe how the tunnel enforces pre-computed transmit schedules.

Assumptions The tunnel design presented in this section relies on some idealized assumptions, which will be relaxed in the practical design of Section 4.5. Specifically, we assume here that the processing delays in the tunnel network stack are not influenced by secrets directly or indirectly. This requires that: (i) the tunnel's layers—especially the shaping, transport, and encryption layers, which operate on cleartext data—execute in *constant time*, *i.e.,* they avoid data-dependent control flow and memory access patterns in their data path implementation; and (ii) the execution of the tunnel network stack is *performance-isolated* from the application and any other computation outside the stack.

Transmit schedules A transmit schedule is a finite series of times at which packets within a flow are transmitted. A schedule is typically associated with a type of packet train, e.g., a file transfer or the response to a service request. There is at most

one active schedule on a flow at a time; successive schedules on the same flow are non-overlapping in time.

Outbound data processing A timestamp is taken whenever data is queued by the application; these timestamps and the recorded traffic indicators are shared with the gray-box profiler. The shaping layer retrieves a chunk of available data from the flow's outbound queue whenever a transmission is due on a flow according to the active schedule (if any) and TCP's congestion window is open (see transport layer below). The layer removes a number of bytes that is the minimum of (i) the available bytes in the queue, (ii) the receiver's flow control window (see transport layer below), and (iii) M, the network's maximal transfer unit (MTU) minus the size of all headers in the stack. If fewer than M bytes (possibly zero) were retrieved from the queue due to payload unavailability or flow control, the shaping layer pads the chunk to M bytes. It adds a header indicating the number of padding bytes added.

Transport layer The transport layer operates as normal, except for two tunnel-related modifications. First, when the congestion window closes, the transport layer signals the shaping layer to suspend the flow's transmit schedule until the congestion window reopens. Schedule suspension ensures network stability and TCP-friendliness, and does not leak information because it depends only on network conditions, which are visible to the adversary anyway. Second, the flow control is modified to make it unobservable to the adversary. The transport layer signals to the shaping layer the size of the flow control window advertised by the receiver. This window controls how much payload data is included in packets generated by the shaping layer (see above). The transport layer transmits packets irrespective of the flow control window, sending dummy packets while the window is closed, which are discarded at the other end of the tunnel.

The transport layer passes outbound packets to the encryption layer, which adds a message authentication code (MAC) keyed with the flow's key to a header and encrypts the packet with the flow's key. Finally, encrypted packets are passed to the IP layer, where they are processed as normal down the remaining stack and transmitted by the NIC.

Inbound packet processing Packets arriving from the tunnel are timestamped; the stamps are shared with the profiler. Packets pass through the layers in reverse order, causing TCP to potentially send ACKs. The encryption layer decrypts and discards

packets with an incorrect MAC. The shaping layer strips padding and places the remaining payload bytes (if any) into the inbound queue shared with the application.

Schedule installation A transmit schedule must be installed on a flow before data can be sent via the tunnel. A guest application does so indirectly by sending traffic indicators. The application provides the flow's 5-tuple f, a traffic indicator *sid* (which maps to a transmit schedule), and a type. The shaping layer looks up the schedule associated with *sid* in the schedule database and associates it with flow f.

There are two types of schedules: *default* and *custom*. A default schedule is installed when the flow is created. This schedule acts as a template, which is instantiated automatically by the shaping layer whenever an incoming packet arrives that indicates the start of a new network exchange (e.g., a GET request on a persistent HTTP connection), identified by the TCP PSH flag. The schedule starts at a time equal to the arrival time of the packet that causes the schedule's instantiation.

A default schedule active on a flow can be extended by a custom schedule in response to an application traffic indicator. For instance, a default schedule that allows a TLS handshake might be extended with one that is appropriate for the response to the first incoming network request. The new schedule can extend the currently active schedule only if the new schedule's prefix matches the prefix of the currently active schedule that has already been played out. Because the new schedule is anchored at the same time as the profile it extends, the time of a schedule extension is unobservable to the adversary. A traffic indicator that would require a custom profile that does not match the played-out prefix of the active schedule is ignored.

4.4.3 Tunnel security

Next, we justify the cloaked tunnel's security, summarized by property:

S0 *The shape of traffic in the tunnel does not depend on secrets.*

This property follows from the seven properties described next.

S1 *Transmit schedules are chosen based on public information.*

This property holds by the assumption about applications' choice of schedules.

S2 *The traffic in the tunnel is independent of the payload traffic.*

This holds because packets are (i) padded and transmitted independently of the application's rate of payload generation and consumption; (ii) elicit a transport-layer

response from the receiver independently of payload traffic; and (iii) the packet contents, including headers that reveal padding, are encrypted.

S3 *All packet transmissions follow a schedule.*

This holds because the tunnel initiates transmissions according to a schedule.

S4 *Delays between scheduled and actual packet transmission times do not reflect secrets.*

This follows from the fact that the tunnel stack, from the shaping layer down, is performance-isolated from any secret-dependent computation and layers that operate on cleartext are constant-time.

S5 *Transmit schedules are activated, paused, and re-activated at a secret-independent delay from any observable event that causally precedes the pausing or (re-)activation.*

This holds because (i) pausing, reactivation, and instantiation of default schedules is performed within the performance-isolated tunnel stack; and (ii) by assumption, custom schedule installations that take immediate effect are not causally preceded by an observable event.

S6 *Transmit schedules are suspended and resumed only in response to and according to the network's congestion state.*

This follows from the tunnel's transport layer congestion control mechanism.

S7 *Modifications of active transmit schedules do not reveal secrets.*

This holds because the time of schedule replacement is unobservable to the adversary (matching prefix).

4.5 Pacer design

In the previous section, we described the conceptual design of a cloaked tunnel. In this section, we describe Pacer, a concrete and practical cloaked tunnel design in the context of a public IaaS cloud.

We begin with a discussion of constraints on the design space in the context of a IaaS cloud. There are three constraints that need to be addressed while designing the tunnel for the cloud tenants.

First, the tunnel must cover shared network links observable by an adversary. In an IaaS cloud, co-located tenants typically share the network link attached to the

server and can therefore indirectly observe each others' traffic. Therefore, the tunnel entry must be in the IaaS server to ensure that the attached link lies inside the tunnel.

Second, *shaping requires padding, which must be done at the transport layer to ensure that it is unobservable* (requirement **T2** as defined in section 4.4.1).

Third, the conceptual tunnel design from the previous section requires that the network stack is *performance-isolated from secret-dependent computations and layers that deal with cleartext are constant-time* (as described in the assumptions in section 4.4.2). However, performance-isolating the network stack within the IaaS guest is difficult, since it shares resources with the application and the rest of the kernel, whose execution may be secret-dependent and can affect the execution of the network stack. For instance, the allocation of packet buffers or preparation of packet headers may be preempted due to another secret-dependent execution, or delayed while accessing the shared physical memory or cache which contained application secrets. This suggests that shaping should be implemented in the IaaS hypervisor, where it can be executed with dedicated resources and tightly controlled.

One way to meet all the requirements could be to place the entire network stack into the hypervisor, performance-isolate it from guests, and implement it so that it executes in constant time. This approach, however, has significant limitations. First, ensuring performance-isolation for an entire network stack is technically challenging even in the hypervisor. Second, implementing the tunnel layers as constant time is not trivial. Third, the approach defeats NIC virtualization, such as SR-IOV (single root input/output virtualization), and instead requires that guests and their network peers use the network stack provided by the IaaS platform. Finally, it adds significant complexity to the hypervisor.

Pacer addresses the tension introduced by the above constraints using a paravirtualized approach. We describe Pacer's architecture in detail next.

4.5.1 Pacer architecture

In Pacer, the hypervisor cooperates with the guest OS to implement the cloaked tunnel. The responsibilities are divided in such a way that (i) the hypervisor can ensure tunnel security with only weak assumptions about a guest's rate of progress, (ii) the performance-isolated hypervisor component is small, and (iii) required changes to the guest OS are modest. Effectively, we extend the IaaS hypervisor to provide a small set of functions that allows guests to implement a cloaked tunnel, while guests retain the flexibility to use custom network stacks on top of a virtualized NIC.

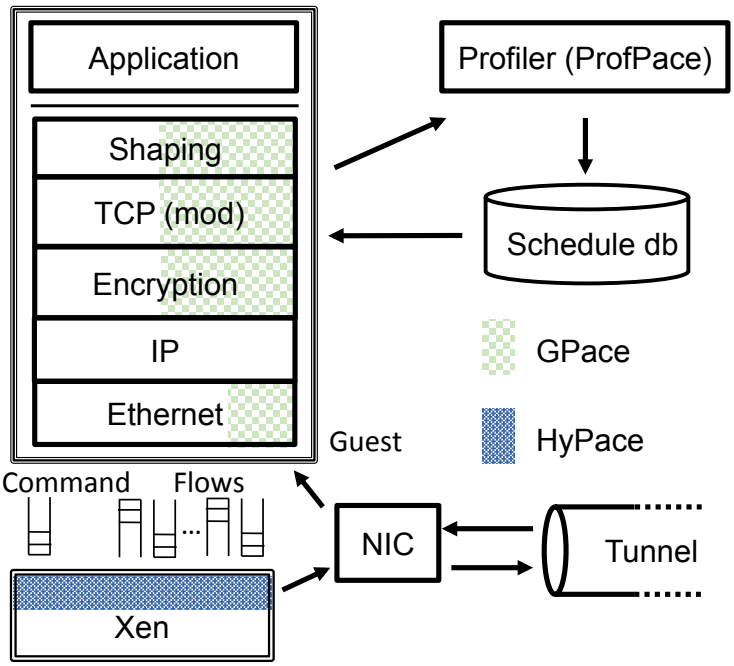


FIGURE 4.4: Pacer architecture

Figure 4.4 shows Pacer's architecture. Unlike the strictly layered tunnel stack from Section 4.4, Pacer factors out a small set of functions that inherently require performance-isolation into the lowest layer, implemented in the IaaS hypervisor. The *HyPace* component plugs into the Xen hypervisor [184] and provides these functions. The *GPace* component, a Linux kernel module, plugs into the guest OS and the OS of any network clients that interact with the guest. It implements the cloaked tunnel in cooperation with HyPace.

HyPace instantiates transmit schedules, encrypts and adds MACs to packets, and initiates their transmissions, while masking potentially secret-dependent delays in its execution. It can generate padded (dummy) packets subject to congestion control independently from the guest network stack, thereby avoiding the need to performance-isolate the guest. GPace pads payload packets, and exposes each flow's congestion window, sequence number, and crypto key to HyPace. The guest has direct access to a SR-IOV virtual NIC (vNIC) configured by the hypervisor, which it uses to receive but not to transmit packets. As we show in section 4.5.2, Pacer's security properties remain equivalent to those of the conceptual cloaked tunnel design.

4.5.1.1 HyPace

Similar to the shaping layer in the conceptual tunnel design, HyPace receives traffic indicators from applications (via GPace), instantiates template schedules in response to incoming packets (signaled by GPace), and initiates transmissions. To ensure tunnel security despite potentially secret-dependent delays in the guest, however, HyPace performs additional functions and there are differences, which we discuss next.

HyPace implements padding, encryption, and congestion control in cooperation with the guest. HyPace pauses a transmit schedule when a flow's congestion window closes, and resumes the schedule when it reopens. When a transmission is due on a flow and the congestion window is open, HyPace checks whether the guest has queued a payload packet. If not, it generates a dummy packet with proper padding, transport header, and encryption, using the flow's next available TCP sequence number and the crypto key shared with the guest. Finally, HyPace initiates the transmission of the payload or dummy packet and reduces the congestion window accordingly.

Interface with guests HyPace shares a memory region pairwise with each guest. This region contains a data structure for each active flow. The flow structure contains the following information: the connection 5-tuple associated with the flow, a sequence of transmit schedule objects, the current TCP sequence number *seq* and the right edge of the congestion window *cw*, the flow's encryption key, and a queue of packets prepared for transmission by the guest. Each transmit schedule object contains the traffic indicator *sid* and a starting timestamp. HyPace and the guest use lock-free synchronization on data they share.

Packet transmission HyPace transmits packets according to the active schedule in the packet's flow. From a security standpoint, packets need not be transmitted at the exact scheduled times; however, any deviation between scheduled and actual time must not reveal secrets.

On general-purpose server hardware, it is challenging to initiate packet transmissions such that their timing cannot be influenced by concurrent, secret-dependent computations. Using hardware timers, events can be scheduled with cycle accuracy. However, the activation time and execution time of a software event handler is influenced by a myriad of factors. We call these factors *internal timing leak* factors, which

may include, among other factors, (i) frequency and voltage scaling; (ii) disabled interrupts at the time of the scheduled event; (iii) non-maskable interrupts during the handler execution; (iv) the CPU's microarchitectural, cache, and write buffer state at the time of the event; and (v) concurrent bus traffic. Many of these factors are influenced by the state of concurrent executions on the IaaS server and may therefore carry a timing signal about secrets in those executions.

We disable hyperthreading, dynamic voltage and frequency scaling (DVFS), and power management in the hosts, and we pin HyPace to run on a dedicated core, where no VM is scheduled. These steps allow to mitigate the effects of (i)—(iii). We implement a *masking* approach to mask the effects of all other factors on HyPace's execution. We next describe the masking approach.

Masking event handler execution time HyPace masks hardware state-dependent delays to make sure they do not affect the actual time of transmissions. A general approach is as follows. First, we determine empirically the distribution of delays between the scheduled time of a transmission and the time when HyPace's event handler writes to the NIC's *doorbell register*, which initiates the transmission. We measure this distribution under diverse concurrent workloads to get a good estimate of its true maximum. We relax this estimate further to account for the possibility that we may not have observed the true maximum and call this resulting delay δ_{xmit}. Second, for a transmission scheduled at time t_n, we schedule a timer event at $t_n - \delta_{xmit}$. Third, when the event handler is ready to write to the NIC doorbell register, it spins in a tight loop reading the CPU's clock cycle register until t_n is reached and then performs the write. By spinning until t_n, HyPace masks the event handler's actual execution time, which could be affected by secrets.

Unfortunately, the measured distribution of event handler delays has a long tail. We observed that the median and maximum delay can differ by three orders of magnitude (tens of nanoseconds to tens of microseconds). This presents a problem: With the simple masking approach, a single core could at most initiate one transmission every δ_{xmit} seconds, making it infeasible to achieve the line rate of even a 10Gbps link. Instead, we rely on *batched transmissions*.

Batched transmissions The solution is based on two insights. First, our extensive empirical observations indicate that the instances in the tail of the event handler

delay distributions tend to occur very infrequently and never in short succession[6]. As a result, the maximal delay for transmitting n packets in a single event handler activation does not increase much with n. Therefore, we can amortize the overhead of masking handler delays over n packets. Second, actual transmission times can be delayed as long as the delay does not depend on secrets. Hence, it is safe to batch transmissions.

We divide time into *epochs*, such that all packet transmissions from an IaaS server scheduled in the same epoch, across all guests and flows, are transmitted at the end of that epoch. An event handler is scheduled once per epoch. The handler prepares all packets scheduled in the epoch, spins until the batch transmission time, and then initiates the transmission with a single write to the NIC's doorbell register. We discuss the choice of epoch lengths and batch sizes in Section 4.8.3.

To understand the security of the batching mechanism, let us consider factors that could delay the actual packet transmission time once the spinning core issues the doorbell write. Buffers containing the packets were read before the spin, so the state of caches plays no role. Similarly, the memory write buffer should be empty after the spin. Interference from concurrent NIC DMA transfers reflects shaped traffic and is therefore secret-independent. Similarly, any delays in the NIC itself due to concurrent outbound or inbound traffic cannot depend on secrets. Thus, the actual packet transmission times are independent of secrets.

However, the doorbell write itself could be delayed by traffic on the memory bus, PCIe bus, or bus controller/switch. We next discuss the prevalence of such delays and ways to eliminate them.

Hardware interference and NIC support A remaining source of delays are concurrent bus transactions caused by potentially secret-dependent computations. Empirically, we have not been able to find clear evidence of such delays. Nonetheless, such delays cannot be ruled out completely on general-purpose hardware.

A principled way to rule out such interference would require additional hardware support. For instance, a *scheduled packet transmission* function provided by the NIC would be sufficient. Software would queue packets for transmission with a future transmission time t. At time $t - \delta_{bus}$, the NIC DMAs packets into onboard staging buffers in the NIC. Here, δ_{bus} would be chosen to be larger than the maximal possible delay due to bus contention. At time t, the NIC would initiate the

[6]Without the knowledge of Intel CPU internals, it is difficult to determine the exact cause of the tail latencies, but their frequency suggests that they may be caused by system management interrupts.

transmission automatically. With such NIC support, HyPace would prepare packets for transmission as usual, but instead of spinning until t_n it would immediately queue packets with $t = t_n$. Incidentally, NIC support for timed transmissions is also relevant for traffic management, and a similar "transmit-on-timestamp" feature is already available on modern smart NICs [122].

HyPace summary HyPace is a minimal component implemented in the hypervisor, which is performance-isolated from the guest and enables guests to implement a cloaked tunnel. HyPace's careful design masks any potentially secret-dependent delays in the transmission of a packet, obviating the need for a constant-time implementation of any part of the tunnel's network stack or a performance-isolated guest network stack. At the same time, the batched transmission design amortizes the high cost of masking and helps to sustain packet transmission througput close to the NIC's line rate.

4.5.1.2 GPace

GPace is a Linux kernel module that implements a cloaked tunnel jointly with HyPace. On the client-side of a network connection, GPace extends the kernel to terminate the tunnel. GPace pads outgoing TCP segments to MTU size and removes the padding on the receive path. It modifies Linux's TCP implementation to share its per-flow congestion window and sequence number with HyPace, and to notify HyPace of retransmissions so that HyPace can extend the active schedule by one transmission. Unlike in the generic tunnel, where the shaping occurs above the transport layer, this schedule extension is necessary to allow for retransmission; it does not leak information because it depends only on network state.

Note that TCP's flow control window is not advertised to HyPace, causing HyPace to send dummies if the receiver's flow control window is closed, as required. GPace timestamps outbound data arriving from applications and inbound packets from the tunnel in the vNIC interrupt handler. All timestamps and recorded traffic indicators are consumed by the profiler (Section 4.6).

GPace allows applications to install session keys and provide traffic indicators on flows via IOCTL calls on network sockets. Recall that applications specify a flow, a traffic indicator *sid*, and a type as arguments when indicating traffic. GPace passes this information into the per-flow queue shared with HyPace, which uses the *sid* as an index to look up the corresponding transmit schedule in the database.

Packet processing With GPace, the guest OS generates TCP segments as usual, but pads them to the MTU size before passing them to the IP layer[7]. Instead of queuing packets in the vNIC's transmit queue, GPace queues them in per-flow transmit queues shared with HyPace. The guest OS processes incoming packets as usual by accepting interrupts and retrieving packets directly from its vNIC.

Schedule (re-)activation delays Unlike the conceptual tunnel design, Pacer processes inbound network packets in the guest, which is not performance-isolated. Therefore, care must be taken to ensure that the time of activation or re-activation of a transmit schedule in response to an inbound packet does not reveal the guest kernel's execution time, which could depend on secrets. There are four events that trigger schedule (re-)activation. Below we list the events and describe how we ensure schedule (re-)activations at defined, secret-independent delays from the corresponding causally-preceding events. Let ϵ be HyPace's epoch length and δ_{recv} be the guest OS's empirical maximal inbound packet processing time.

(i) *The arrival of the first packet of a request.* GPace instantiates a default schedule with a start time equal to the packet's arrival time. To make sure the first transmission occurs in time, we require that the initial response time of any default schedule be larger than $\epsilon + \delta_{recv}$.

(ii) *The arrival of an ACK that opens the congestion window.* GPace ensures that the ACK does not enable a transmission that is scheduled within $\epsilon + \delta_{recv}$ of the ACK's arrival.

(iii) *The arrival of an ACK that causes a retransmission.* GPace ensures that the ACK does not enable a transmission that is scheduled less that $\epsilon + \delta_{recv}$ from the ACK's arrival.

(iv) *A timeout that causes a retransmission.* GPace ensures that TCP's timeout does not enable a transmission that is scheduled within $\epsilon + \delta_{recv}$ of the timeout. Here, we use δ_{recv} as a conservative upper bound on the execution delay of the timeout event handler.

These four rules make the guest's actual processing time for incoming packets and timeouts unobservable to the adversary.

[7]Note that ACKs are not padded as Pacer does not need to hide client traffic shape. However, ACKs are paced to hide guest's interference with their transmission

4.5.2 Pacer security

We now justify Pacer's overall security. Pacer's threat model rules out side channels via shared CPU state, caches, and memory bandwidth, as well as shared cloud back-end services. Therefore, in its attempt to learn the victim's secrets, the adversary is limited to (i) trying to connect to the victim as a client and observe the timing and content of responses, or (ii) measuring the shape of the victim's traffic by observing packet sizes and timings on a shared network link.

Attack (i) is not possible because the adversary cannot elicit a response from the victim. This is because Pacer relies on encryption and a MAC keyed with pre-shared keys and GPace silently ignores incoming packets that cannot be authenticated.

Attack (ii) is unproductive because the victim's incoming traffic shape is secret-independent by assumption (Section 4.2), and its outgoing traffic is shaped to be secret-independent as discussed in this section. Next, we justify that the victim's outgoing traffic shape is indeed secret-independent *by design*. In other words, we justify that Pacer's tunnel has property S0 of the cloaked tunnel from section 4.4.3.

S1 and **S3** hold trivially, because the relevant behavior of Pacer is equivalent to the conceptual tunnel's. **S2** holds because Pacer, like the conceptual tunnel, pads packets above the transport layer, encrypts packets below the padding layer, and makes flow control unobservable. **S4** follows from GPace's rules on the pausing and (re-)activation of transmission schedules. **S5** holds because HyPace's batch transmission mechanism masks the execution time of its transmission event handler. **S6** holds because HyPace cooperates with GPace to pause and resume schedules in response to the network's congestion state. **S7** holds because the schedule extension happens in Pacer only in response to a packet loss, which is a public event.

4.6 Generating schedules

By default, Pacer can use the same transmit schedule for all of a guest's network traffic. This approach does not require any guest support and is perfectly secure. In practice, however, tenants can significantly reduce bandwidth and latency overhead by using different schedules for different partitions of their workload. As long as those partitions are chosen by public information, no information is leaked. Moreover, by profiling guests' network traffic, Pacer can generate transmit schedules that closely reflect the guest's network requirements and minimize inefficiencies in packet padding and pacing.

In this section, we discuss ProfPace, Pacer's gray-box profiler that profiles guests and generates transmit schedules automatically by analyzing the guest's recorded network interactions and the explicit traffic indicators provided by guest application. For content-serving guest applications, we then analyze the application's content corpus to suggest a clustering that maximizes bandwidth efficiency given the application's privacy needs.

4.6.1 Gray-box profiling

At each site in the guest's application where a message is sent to the network, an IOCTL call is invoked that provides a traffic indicator *sid*. The *sid* identifies segments of the same equivalent class, *e.g.*, a TCP handshake, a TLS handshake, or a response to a request within a given workload partition. GPace logs the traffic indicators along with the arrival times of incoming packets and the times at which the guest OS queues packets for transmission, and shares the logs with ProfPace.

ProfPace bins the recorded network interaction segments by *sid*. The set of observed segments in a bin are considered samples of the associated equivalence class of network interactions. ProfPace characterizes the traffic shape for each class with a set of three random variables: (i) the delay between the first incoming packet and the first response packet d_i, (ii) the time between subsequent response packets d_s, and (iii) the number of response packets p. For each class, the profiler samples the distribution of these random variables from the segments in the associated bin.

Finally, ProfPace generates a transmit schedule for *sid* based on the sampled distributions of the random variables. Specifically, it generates a schedule with the 100th percentile of the number of packets p, the 99th percentile of the initial delay d_i, and the 90th percentile of the spacing among subsequent packets d_s. We have determined empirically that transmit schedules generated thus work well.

Recall that as long as applications choose *sid* values based on public information, transmit schedules are relevant only for performance not security. An inadequate schedule could increase delays and waste network bandwidth due to extra padding, but cannot leak secrets. For good performance, during profiling runs, the guests should sample the space of workloads with different values of the public and private information, as well as different guest load levels, so that the resulting profiles capture the space of network traffic shapes well.

Next, we consider how a content-serving guest can partition its workload to minimize overhead given its privacy needs.

4.6.2 Corpus analysis

Pacer minimizes overhead by allowing the shape of network traffic to reveal information deemed public by the guest. Towards this end, a guest may partition its workload so as to minimize padding overhead while ensuring that no secret information is revealed. For instance, consider a guest that serves a corpus of objects with a skewed size distribution. Using a single schedule for the entire corpus requires padding every requested object to the largest object in the corpus, possibly incurring a large overhead. Suppose the guest can partition the corpus based on public information such that each partition contains objects of similar size; now each object is padded to the largest object in *its* cluster, which may reduce overhead significantly without revealing which object within a partition is being requested. Below we briefly describe a clustering algorithm for videos and static HTML documents that can minimize padding overhead subject to a guest's privacy needs.

However, we note that determining what information can be considered public and private in the context of a specific application and the corpus's size and popularity distributions may be challenging in general, and is beyond the scope of this thesis. Our goal here is to highlight the large efficiency gains possible when clustering content with skewed size distributions. We present specific overhead results when clustering real videos and document corpuses in section 4.8.2.

Video clustering In many popular video streaming services, videos are streamed as a sequence of segments of fixed duration, typically between 1s and 15s [38, 129]. The number of segments in a video, therefore, equals total duration of the video divided by segment duration. Furthermore, the video streams are dynamically compressed using a variable bitrate encoding scheme, such as MPEG-DASH [119] or HLS [76], which leads to video segments compressed to different sizes, *i.e.*, number of bytes. Consequently, to hide a video within a cluster, we need to pad the number of segments in the video as well as the sizes of the segments in the sequence.

Initially, we over-approximate the shape of each video v_i by its maximal segment size $smax_i$ and its number of segments l_i. For each distinct video length l and each distinct maximal segment size s in the entire dataset, we compute the set of videos that are dominated by $\langle l, s \rangle$. A video v_i is dominated by $\langle l, s \rangle$ if $l_i \leq l$ and $smax_i \leq s$.

Let c be a desired minimum cluster size. Our algorithm works in rounds. In each round, we select every $\langle l, s \rangle$ dominating at least c videos, and we choose as cluster

the set of videos minimizing the average relative padding overhead per video, *i.e.*,

$$\frac{1}{c_i}\sum_{j=1}^{c_i}\sum_{k=1}^{l_i}\left(\frac{\left(s_k - s_{kj}\right)}{s_{kj}}\right)$$

where c_i is the cardinality of the set of videos, l_i is the maximal length across all videos in the set and s_k is the maximal size of the k-th segment across all videos in the set (*i.e.*, $\max_{1\leq j\leq c_i}(s_{kj})$). The sequence of segment sizes $\langle s_1, s_2, ..., s_{l_i}\rangle$ is the ceiling of the cluster c_i. Once a cluster is formed, videos in it are not taken into account in later rounds. The algorithm terminates when all videos are clustered. If the last cluster has less than c videos, it is merged with the one formed before it.

Document clustering We use a similar but simpler clustering algorithm to cluster static HTML documents. In contrast to videos, HTML documents contain only one data object. Therefore the clustering problem simplifies to one-dimensional clustering based on the single size parameter of individual documents, and the largest document in a cluster constitutes the cluster's ceiling.

4.7 Implementation

We implemented HyPace for the Xen hypervisor in ~8,100 lines of C. We imported an additional 4,458 lines of AESNI assembly code from OpenSSL 1.1.1b [128] to encrypt packets in HyPace[8].

We implemented GPace's Linux kernel module in ~15,000 lines of C. GPace intercepts the kernel network stack at a few points in the TCP layer and the NIC driver layer, but does not modify the application's interface with the network stack. GPace modifies the semantics of TCP's largely unused URG flag and urgent pointer to indicate the number of padding bytes in a packet[9].

At each site in an application's code where a message is sent to the network, we add 15 LoC to send a traffic indicator via IOCTL to the guest kernel. We identified and modified these sites manually; automating the instrumentation is possible. No other changes were required to guest applications.

Finally, we implemented ProfPace in 1,200 lines of C code which collects packet logs from GPace, and 1,800 lines of Python code which computes traffic shapes.

[8] Alternatively, the guest could terminate an IPSec session and share the key with HyPace.

[9] Alternatively, a separate padding protocol could be implemented above TCP to indicate padding in a separate header.

4.8 Evaluation

In this section, we present results of an empirical evaluation of Pacer's prototype.

4.8.1 Experimental setup

All experiments were performed on Dell PowerEdge R730 server machines with Intel Xeon E5-2667, 3.2 GHz, 16-core CPU (two sockets, 8 cores per socket), 512 GB RAM, and a Broadcom BCM 57800 10Gbps Ethernet card. The NIC supports SR-IOV, and was configured to export virtual NICs (vNICs). We disabled hyperthreading, dynamic voltage and frequency scaling, and power management in the hosts.

We run the Xen hypervisor (version 4.10.0) on the hosts. The hypervisor is assigned one of the CPU sockets and 40GB of RAM. Up to two cores are configured to execute the HyPace transmit event handler in parallel; guests' flows are statically partitioned among the HyPace cores. The guest runs an Ubuntu 16.04 LTS kernel (version 4.9.5, x86-64) in a VM with 8 cores and 64 GB RAM, and has access to a vNIC. The VCPUs of the guest VM were pinned one-to-one to cores on the second socket of the host CPU, and we used Xen's "Null" scheduler [185] for VM scheduling. This is in line with our threat model, which assumes that guests rent dedicated CPU sockets. Pacer requires less than 10MB of additional main memory in the Xen hypervisor and less than 20MB of additional memory in each guest that uses Pacer. The network clients run Ubuntu 16.04 LTS without a hypervisor.

We evaluated Pacer's impact on client latencies and server throughput in the context of two guest applications: (i) a video streaming service, and (ii) a web service serving static documents. Both services use Apache HTTP Server 2.4.33 [5]. For video streaming, we wrote a custom application in PHP, which works like a simple file server serving video segments in response to client requests. The application serves videos from an ext4 file system on a VM disk. The document service is based on the Mediawiki server, version 1.27.1 [112]. Documents are stored in a database hosted on MySQL 5.7.16 [120] within the VM. We used a modified wrk2 [181] client to issue HTTPS GET requests for various pages to the document server.

With both services, we use real-world datasets, so that the videos or documents have real size distributions. For videos, we use a corpus of videos that we downloaded from YouTube in March 2018. For the document service, we use two different real data sets: static HTML pages of the English Wiktionary and the English Wikipedia [57]. Note that even though Youtube videos, Wiktionary pages, and Wikipedia pages are not sensitive and may not need protection with a system

like Pacer in practice, all that matters for our evaluation are the file sizes and size distributions. The content of the documents is irrelevant as it is encrypted during transmission anyway.

4.8.2 Spatial padding overhead

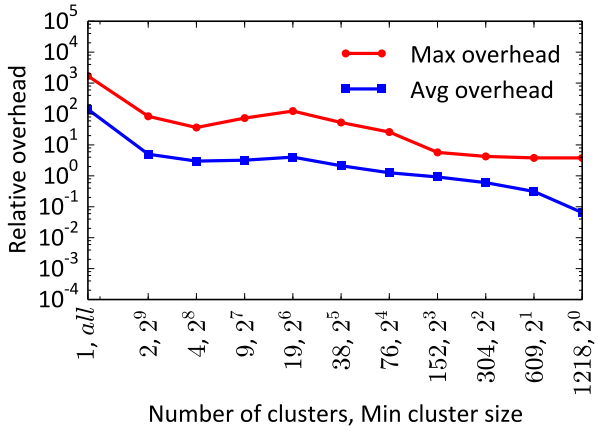
First, we measure the spatial padding overhead when clustering content as described in Section 4.6. This overhead corresponds to the network bandwidth overhead for an application due to Pacer's traffic shaping.

We clustered three different datasets: (i) a set of videos that we downloaded from YouTube in March 2018 (1218 videos, including music, sports, interviews, movie trailers, news, and tv series, 240p bitrate, max duration 4.2 hours, median duration 7 minutes, max size 468.7MB, median size 6.2MB), (ii) a 2016 snapshot of the English Wiktionary corpus (5,027,344 documents, max 521.9kB, median 4.7kB) [57], and (iii) a 2008 snapshot of the English Wikipedia corpus (14,257,494 documents, max 14.3MB, median 83.5kB) [57].

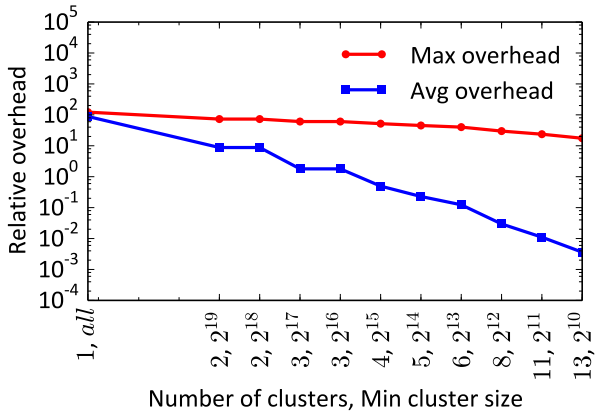
Figure 4.5 shows the reduction in the average and maximum padding overhead with increasing number of clusters and decreasing minimum cluster size (*i.e.*, the minimum number of objects in each cluster). Compared to the two document datasets, the overhead reduction is less for videos due to the smaller dataset with a narrower range of video sizes, and due to the multi-dimensional clustering necessary for videos. Nonetheless, there is a significant reduction in the average overhead with increasing number of clusters in each dataset, and even clustering the corpuses into just two clusters leads to at least two orders of magnitude reduction in the average padding overhead.

4.8.3 Microbenchmarks

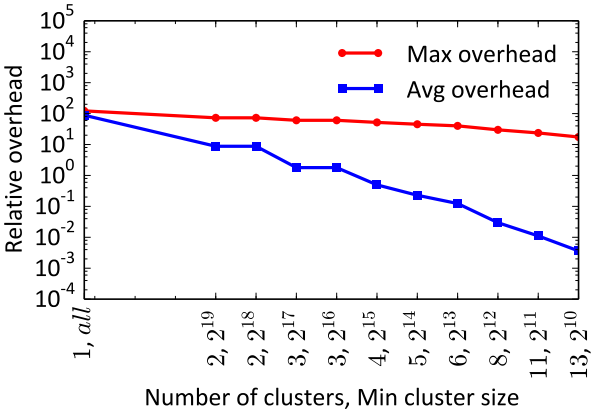
We empirically select a suitable HyPace epoch length ϵ, the maximum batch size B (number of packets to be prepared by a HyPace handler) in each epoch, and the parameters δ_{xmit} and δ_{recv} from Section 4.5. To this end, we ran multiple 12-hour experiments with varying network workloads. We requested 100KB-sized documents from the document server using concurrent clients. As background workload, we ran large matrix multiplications on the server's dom0. The workloads were configured to drive CPUs to near 100% utilization, and had a total working set of ~12GB of RAM.



(A) Video dataset



(B) English Wiktionary



(C) English Wikipedia

FIGURE 4.5: Relative padding overhead vs number of clusters and minimum cluster size (log-log scale) for (A) Youtube videos, (B) English Wiktionary, and (C) English Wikipedia

To determine δ_{xmit}, ϵ and B, we measured the cost of preparing batches of packets for transmission in HyPace. Over many observations in the presence of the background load described above, we first determined the number of packets that can be safely prepared with different epoch lengths with a single HyPace handler. Within epochs of length 30μs, 50μs, 100μs and 120μs, the number of packets that could be prepared was 5, 14, 33, and 42, respectively, which allows HyPace to achieve 22%, 28%, 41% and 42% of the NIC line rate with a single core. We configured ϵ to be 120μs for all HyPace handlers.

Based on these results, we use two parallel HyPace handlers running on two separate cores. In this configuration, we repeated our measurements and chose $B = 38$ packets and δ_{xmit} = 35μs for each handler. δ_{recv} is independent of the number of HyPace threads, and its average and maximum values observed across all experiment configurations were 3.9ms and 15.8ms, respectively. We conservatively configured δ_{recv} to 20ms.

4.8.4 Video streaming

Next, we measure the impact of Pacer's traffic shaping on the latencies of clients using the video streaming service.

Experimental setup We used the clustering configuration of 19 clusters with each cluster containing at least 64 videos. The clustering yields an average padding overhead of 4x for the dataset (Figure 4.5a).

We wrote a Python streaming client that simulates a MPEG-DASH player: when a user requests a video, the client initially downloads a few segments (covering 5s of video each) in succession to fill up a local buffer. After reaching 50% of the initial buffer (rebuffering goal), the player starts "playing" the video by consuming the segments from the buffer. The client downloads subsequent segments sequentially whenever space is available in the buffer. Once the local buffer is filled (*i.e.,* buffering goal is reached), the client typically downloads one segment every 5s. In all our experiments, we set the client's buffering goal to 60s and the rebuffering goal to 30s.

With Pacer, the player does not request a segment until the transfer of the previous segment, including any padding, has finished; otherwise, the timing of the client's request would reveal the actual size of the previous segment. In our experiments, we set the interval between segment requests at 1s.

```
1  {    clusterid: 10,
2       bursts: 2, # n = 2
3       # n lines of { p, d_i, d_s }
4       { 1, 0.5, 0.0 }, # b0
5       { s_i, 30.0, 0.5 }, # b1
6  }
```

FIGURE 4.6: Transmit schedule for i^{th} video segment in a cluster

We measure the impact of Pacer's traffic shaping on: (i) the initial delay until a video starts playing, (ii) the frequency and duration of pauses (video skipping) experienced by the player, if any, and (iii) the download latency for individual video segments. Towards this end, we use a number of concurrent clients that sequentially request four randomly selected videos each from the video server, and play each video for up to 5 minutes. The clients establish a new connection for each video. We ran experiments with 10Gbps and 10Mbps bandwidth configurations on the client side.

Transmit schedules Figure 4.6 shows a compact representation of the transmit schedule for the i^{th} segment of a video cluster generated by ProfPace. Each schedule comprises two bursts. Lines 4 and 5 in the figure show the 3-tuple of $\langle p, d_i, d_s \rangle$ (*i.e.*, number of packets, initial delay, and packet spacing as described in section 4.6.1) for the two bursts. Burst b_0 consists of a single packet that will be transmitted 0.5ms after receiving the client request. This burst is used to account for the single ACK that the server may send in response to client requests prior to transmitting the actual response. Burst b_1 consists of s_i packets, where the first packet will be transmitted 30ms after receiving the client request, and the remaining packets will be transmitted at a spacing of 0.5ms. s_i is the number of packets required to transmit the i^{th} ceiling segment in the given cluster. All profiles for the video corpus differ only in the value of s_i.

Streaming for 10Gbps clients With the above schedules, there is no significant impact on the user experience for streaming videos while using Pacer. The average and maximum initial video startup delay (*i.e.*, time until the rebuffering goal is reached and the client starts playing a video) in the baseline are 0.016s and 0.13s respectively,

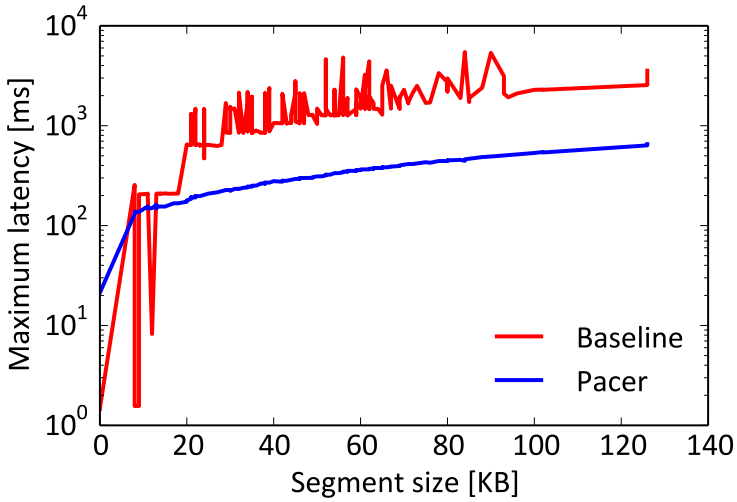


FIGURE 4.7: Download latency for a 10Mbps client for segments of increasing sizes with and without Pacer's shaping

which increase to 6.006s and 6.007s respectively with Pacer. The increase in the delays is due to the interval between segment requests, which is conservatively set to 1s. Note that this gap is essential to ensure that segment request times are independent of each other during traffic shaping.

As can be seen from the segment transmit schedules (Figure 4.6), the download latency for each segment is in the order of hundreds of milliseconds. This is well below the 5s deadline for segment download after the initial buffering. Thus, despite Pacer's traffic shaping we observe no video skipping in any of the experiments. When serving 128 clients, the maximum CPU utilization on the server increases from 3.73% to 6.26% with Pacer.

Streaming for 10Mbps clients Pacer's shaping also provides an opportunity to use domain knowledge to optimize transmit schedules for better *performance*. Downloading the largest segment in our collection of 240p videos is ~325KB in size, which is transmitted in ~250 MTU-sized TCP packets. Downloading this segment within 5s (the deadline when a video player can request only one segment at a time due to full buffers) requires its packets to be sent at an interval of max 20ms. Conservatively increasing the inter-packet spacing in the schedules to even 6ms still allows downloading all the segments within 5s. However, for 10Mbps clients, the paced schedule avoids packet losses and reduces the segment download latency significantly. Figure 4.7 shows the download latency for a 10Mbps client for different segment sizes in

```
{   clusterid: ``small'',
    bursts: 2,
    { 1, 20.0, 0.00 },  # b0
    { 20, 80.0, 0.50 }, # b1
}
```

A: Small cluster

```
{   clusterid: ``large'',
    bursts: 5,
    { 1, 20.0, 0.00 },    # b0
    { 128, 100.0, 0.35 }, # b1
    { 40, 160.0, 0.20 },  # b2
    { 225, 180.0, 0.20 }, # b3
    { 7, 2200.0, 0.10 },  # b4
}
```

B: Large cluster

FIGURE 4.8: Transmit schedules for the two clusters used in the document server: (A) small cluster, (B) large cluster

the baseline, and after applying Pacer's shaping with 6ms inter-packet spacing. Note that this schedule optimization does not affect security; it only takes advantage of Pacer to reduce network contention, which is a known benefit of traffic shaping [6].

4.8.5 Document server

Next, we measured Pacer's impact on the throughput of the document server and the response latency of the clients when serving the English Wiktionary corpus.

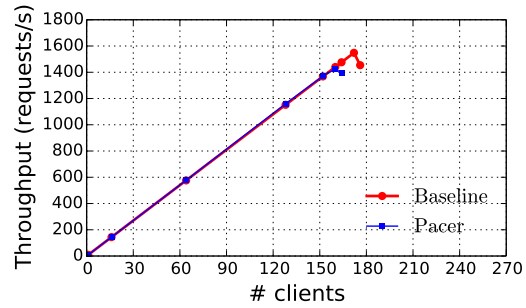
Experimental setup We used a coarse-grained clustering of the corpus with one cluster of 5,010,856 files up to 12KB in size and another with the remaining 16,488 files, which yields an average padding overhead of 150%. Clients request different Wiktionary pages concurrently and synchronously for a period of 120s. Prior to the measurement, we ran the workload for 10s to warm up the caches. We ran a workload trace where clients request a total of 1,000 files randomly chosen from both clusters. For comparison, we stressed the server with requests only to the largest file in the corpus (521.9KB).

Transmit schedules A compact representation of the transmit schedules for the two clusters is shown in Figure 4.8. The schedule for the small cluster is similar to that for video segments: responses are sent in two bursts, with the first burst consisting of a single packet sent 20ms after the client request, and the second burst consisting of 20 packets sent with a spacing of 0.5ms starting at 80ms after the client request. Responses for the large cluster are sent in 5 bursts. The schedule for the large cluster is hand-optimized to reflect the bursty transmission of large files during the slow-start phase of TCP flows. Current ProfPace prototype does not capture the variance in the inter-packet spacing due to closing of congestion window (*e.g.*, during the slow-start phase of TCP flows), and generates transmit schedules with a d_s that is used uniformly for all packets sent on a given schedule. Such a schedule leads to rapid transmission of dummy packets during slow-start phase, and causes transmission of large files to fail.

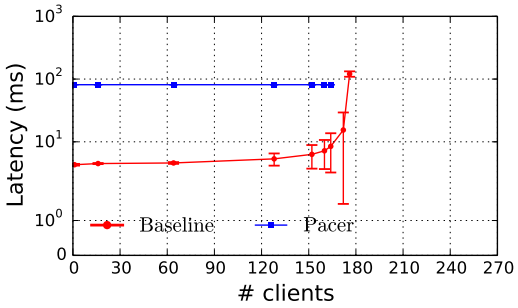
Server throughput and client latency Figures 4.9a and 4.9b show the impact of Pacer's traffic shaping on server's throughput and clients' average response latencies for the request trace workload, respectively. Figures 4.9c and 4.9d show similar results for the workload comprising only the largest file. The errors bars in the plots on the right side show the standard deviations of the average latencies.

With Pacer (blue), once the server is at capacity, it fails to serve additional requests and clients time out. Unlike the baseline, client latencies remain constant until the maximal throughput with Pacer (Figures 4.9b and 4.9d), because the latency is determined by the initial response delay in the transmission schedule. The latencies are higher with Pacer than in the baseline because the profiler generates conservative schedules based on all traffic samples observed during profiling, including samples captured when the application is saturated. Nevertheless, the latencies remain within hundreds of milliseconds, and could be optimized substantially using different schedules for different load conditions.

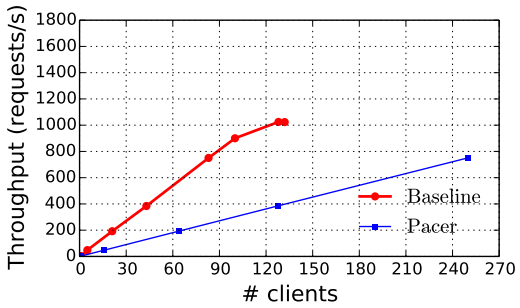
Pacer incurs a 6.8% and 30% overhead on peak throughput for the trace workload and the large file, respectively (Figures 4.9a and 4.9c). These values reflect Pacer's total overhead because they compare to a saturated baseline server. With the large file, the baseline operates at over 40% of the line rate, and we believe that Pacer's performance in this challenging experiment is limited by the accuracy of transmit schedules, which can be improved substantially.



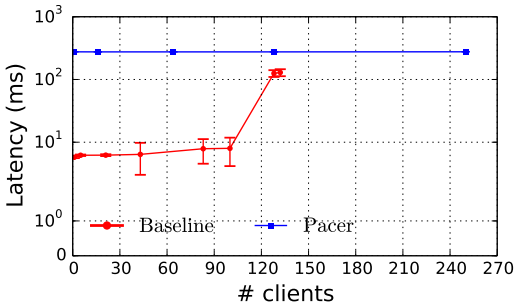
(A) Throughput for trace workload



(B) Client latencies for trace workload



(C) Throughput for largest file



(D) Client latencies for largest file

FIGURE 4.9: Document server throughput and client latencies

4.9 Extensions

In this section, we discuss extensions to Pacer's design to provide network side-channel mitigation for additional classes of applications. We also discuss how Pacer's profiling can be enhanced to automatically identify parameters for efficient workload partitioning. Exploring the design and evaluation of these extensions is left to future work.

4.9.1 Interactive client requests

Pacer currently shapes only the server-side traffic of an application, and assumes that the shape of client traffic reveals no secrets. This is sufficient for services with limited client-server interaction such as file downloading and video streaming. However, in more interactive applications, a client's request may depend on previous responses from the server. In such cases, the client request and even the shape of the client traffic may reveal secrets and must, therefore, be paced and padded. To handle such applications, Pacer's traffic-shaping support can be extended to clients

by modifying the client browser or OS, or by running a Pacer-enabled hypervisor on the client. A transmit schedule for the application would then cover an *entire interaction sequence* between the client and server, from the first request of the client to the last response of the server.

4.9.2 Multi-tier services

Our current prototype of Pacer supports single-tier services, where a single server processes client requests without relying on other backend servers. Multi-tier services can be supported by running Pacer on all servers (frontend and backend) and extending transmission schedules to cover the entire communication graph of a request. However, in a multi-tier service, it may be challenging to determine the best-fitting communication graph from just the initial request. So, we anticipate that, as in our current single-tier design, we would start processing a request using a default profile, which will be replaced with a custom profile after the structure of the remaining communication becomes clear. This is secure as long as the already played part of the default profile at the point of the replacement is a prefix of the new profile.

4.9.3 Dynamic content

In applications like VoIP, the amount of traffic in each direction during a session depends on user actions. In these cases, the structure of the transmission schedule will be application-specific, but we believe that the structure can be determined for many applications. For instance, in VoIP sessions, the duration of VoIP calls can be bucketized, and for any given duration, traffic can be transmitted for that duration at a uniform rate. The granularity of buckets would determine the trade-off between security and transmission-volume overhead, while the rate of transmission would determine the trade-off between call quality and bandwidth overhead.

4.9.4 Private VPN services

Network side-channel disclosures are relevant even for services hosted within an organization's premises and communicating via a secure VPN over the public Internet. Even though an adversary may not achieve co-location with the organization's service inside its premises, a powerful adversary could still observe the VPN traffic directly at network elements in the public Internet. However, because the adversary is outside the organization premises, Pacer's cloaked tunnel design can be simplified

significantly to provide network side-channel mitigation for VPN services. Specifically, the tunnel endpoints can be integrated in a separate middlebox, such as a gateway node of the organization's network. This would allow building a cloaked tunnel without modifying the OS on the application servers, and would provide mitigation for multiple applications with minimal effort from the application developers and system administrators.

4.9.5 Automated discovery of workload partitions

Currently, application developers need to manually identify public parameters that can be used to partition workloads to reduce shaping overheads. For applications with diverse workload characteristics, partitioning the workloads along multiple parameters may yield more efficiency gains. ProfPace can be enhanced to explore the tradeoff between privacy and overheads by generating multiple workload partitions. It can also provide hints to the application developer on parameters that may be most relevant for reducing overheads while retaining privacy guarantees for the application data.

4.9.6 Schedule adaptation

Although the choice of transmit schedules is not relevant for security, it is essential to have efficient schedules that do not incur significant bandwidth and latency overheads. The efficiency of the schedules depends on the coverage of public and private inputs during the profiling stage. If the input coverage is low, or if the application data and workload evolve over time, the schedules may become inefficient. ProfPace can be run periodically to re-compute schedules based on recent workload conditions. The schedule recomputation period must be sufficiently large to ensure that the recomputation does not reveal secrets from recent requests.

4.10 Related work

In this section, we compare Pacer to existing mitigation techniques, and also discuss technically related work with other threat models or goals.

4.10.1 Mitigating network side channels in clouds

Contention on NICs in a cloud can be mitigated by time-division multiple access (TDMA) scheduling in a hypervisor [84] as this eliminates the adversary's ability to observe a co-located victim's traffic. However, this approach is inherently inefficient when the payload traffic is bursty. Statistical multiplexing, which only caps the total amount of data transmitted by a VM in an epoch, is fundamentally insecure as the bandwidth available to a flow depends on the bandwidth used by other flows [62].

Another defense is to restrict the adversary's ability to measure time precisely and at fine granularity [105, 110, 170]. However, this approach is akin to relying on noise in adversary's observations, and cannot completely eliminate the possibility of a side-channel leak. Moreover, this approach forces coarsening the time source for all tenants, which may break the functionality of tenants relying on fine-grained timing measurements.

StopWatch [98] and Deterland [182] replace a cloud VM's access to fine-grained "real time" with a "virtual time" based only on the VM's (deterministic) execution. To prevent a VM from accessing "real time" using an external colluder, StopWatch replicates each tenant's VM $3\times$, co-locates each replica with different guests, and delivers an external interrupt at a virtual time that is the median of the 3 virtual times at which interrupt is received at each replica. This prevents a guest from consistently observing I/O interference with any specific co-located tenant. Deterland, in contrast, simply batches the interrupts and delivers them at boundaries of coarse-grained time intervals.

Both StopWatch and Deterland only reduce the granularity of time that an adversary can observe, but cannot completely mitigate timing channels. Moreover, both systems do not consider network side channels based on packet sizes. Pacer does not prevent an adversary from performing fine-grained timing measurements of victim's traffic, and instead shapes the victim's traffic to eliminate any observable secret-dependent signal in its shape. Additionally, Pacer requires far fewer cloud resources (compared to StopWatch), and fewer modifications to the cloud hypervisor and guests (compared to Deterland).

Bilal *et al*. [12] generate multicast traffic to shape the *pattern* of queries to different backend nodes in multi-tier stream-processing applications in a cloud, but they do not consider leaks due to packet sizes and timing, which is what Pacer addresses.

4.10.2 Traffic-shaping systems to mitigate network side channels

Dependence of packet *size* on secrets can be eliminated by padding all packets to a fixed length [73, 180]. This standard technique is also used by Pacer. Making packet *timing* independent of secrets is substantially harder. A straw man is to send packets continuously at a *fixed* rate independent of the actual workload, inserting dummy packets when no actual packets exist [148, 153]. However, this either wastes bandwidth or incurs high latencies when the workload is bursty.

BuFLO [43] reduces this overhead by shaping response traffic to evenly-spaced *bursts* of a fixed number of packets for a certain minimum amount of time after a request starts. However, this leaks the size of responses that take longer than the minimum time. Tamaraw [21], CS-BuFLO [19], and DynaFlow [107] pad each response to some factor of the original size, *e.g.,* the nearest power of 2. They offer no control over how many objects end up with the same traffic shape. In fact, a traffic shape may correspond to a *single* sensitive object, which would be afforded no protection at all. In contrast to these systems, Pacer allows precise control over cluster size.

Additionally, CS-BuFLO and DynaFlow adapt traffic shapes based on the application's actual transmission rate. However, this rate may depend on application secrets. Thus, the choice of the traffic shape in these systems may leak secrets. In contrast, Pacer allows traffic shape adaptation based only on public inputs, so the traffic shape is secret-independent.

Walkie-Talkie [175], Supersequence [176], and Glove [123] cluster responses, and generate a traffic shape for the cluster that envelopes each response in the cluster. However, the traffic is shaped only for packet size and inter-packet spacing but not for the server response latency. This allows leaks via the application's delay up to the first response packet. Traffic morphing [178] makes sensitive responses look like non-sensitive responses, but only shapes packet sizes and ignores packet timing. In contrast, Pacer shapes packet size, inter-packet spacing, and the server response latency, thus eliminating all leaks by design.

Additionally, the systems described above do not defend against interference between the shaping component (enforcement) and the rest of the application stack, which is required in our context (Section 4.5). Hence, they would allow for potential leaks if integrated directly within the cloud host. Pacer, on the other hand, is carefully designed and implemented to mask all secret-dependent delays in its pacing component (HyPace), as described in section 4.5.1.1.

Zhang *et al.* [193] mitigate leaks in video streaming traffic on the client side. A

proxy in the client browser adds differentially-private noise in the sequence of sizes and timing of the video bursts requested. However, the proxy does not guarantee differential privacy of the sequence of packet sizes and timing, and does not hide video lengths. Pacer clusters videos by sequence of segment sizes, and shapes sizes and timing of packets, segments, and full videos in each cluster deterministically.

4.10.3 Predictive mitigation

Predictive mitigation [8, 192] mitigates network timing side channels and covert channels, but in a threat model fundamentally different from Pacer's. In Pacer, the threat is from a co-located tenant or a network adversary that cannot compromise (or authenticate as) legitimate clients of the victim. In contrast, in predictive mitigation the threat is from legitimate (or authenticated) clients of the victim who may have been compromised. In this setting, the adversary can always distinguish real packets from dummy packets, so predictive mitigation does not rely on dummy packets and uses a fundamentally different shaping strategy: at each scheduled transmission, the enforcement mechanism transmits a packet *only if* the application has actually provided one. Otherwise, nothing is transmitted and an information leak (up to 1 bit) is incurred due to the absence of a packet on the wire. After any such leak, the schedule is *adjusted* using a prediction algorithm (based only on public inputs) to reduce the chances of a leak in the future. In contrast, owing to its different threat model, Pacer is able to send a dummy packet when the application does not provide a real packet before a scheduled transmission. This prevents leaks completely.

Nonetheless, Pacer and predictive mitigation share a key technical idea: Both partition application workloads based on public inputs and compute a traffic shape for each partition ahead of time. The difference is that a bad shape for a partition can leak information in predictive mitigation, but it only affects performance in Pacer.

Finally, the prototype implementation of predictive mitigation does not prevent or mask interference between the pacing logic and the application, which may result in timing leaks from the application to the paced traffic. Hence, that implementation cannot be used in our IaaS context without significant changes (Section 4.5).

4.10.4 Related work with other threat models

Metadata privacy Herd [94], Vuvuzela [168], Karaoke [92], and Yodel [93] provide metadata privacy: they prevent information about who is communicating with

whom from leaking via network side channels. However, these systems do not address leaks via metadata such as the lengths of application messages or calls.

Pacer's goal is to prevent sensitive data from leaking via network side channels, which is fundamentally different from that of the above systems. To address its goal, in addition to shaping the sizes and timing of individual packets, Pacer shapes the lengths of application messages. Further, although metadata-privacy systems and Pacer share some underlying techniques (*e.g.*, use of fixed size packets and dummy packets to shape traffic), Pacer additionally masks interference between the application and HyPace, thus preventing leak of sensitive data via timing channels.

Censorship circumvention systems Systems like Format-Transforming Encryption (FTE) [44], SkypeMorph [118] and ScrambleSuit [177] use a tunnel abstraction similar to Pacer's conceptual design (Section 4.4) to modify payload traffic to bypass a traffic censor's filters. This goal is different from Pacer's goal of decorrelating observable traffic from secrets, and the tunnel's design depends on the assumptions about the censor's filters. For example, FTE [44] circumvents filters that use only deep-packet inspection, but not packet size or timing information. Hence, FTE does not shape traffic and offers no protection against threats that Pacer defends against.

ScrambleSuit [177] seeks to bypass censors that may inspect packet sizes and timing. It shapes packet size within the tunnel to a distribution picked ahead of time, independent of any secrets. However, it only weakly obfuscates packet timing by adding a bounded random delay to every payload packet. This does not hide long (secret-dependent) inter-packet gaps and may leak information, unlike Pacer.

SkypeMorph [118] has goals similar to ScrambleSuit. It extends traffic morphing [178] to sample both the inter-packet gap and the packet size from a fixed distribution, which mimics the distribution of some target protocol that the censor is assumed to allow. Dummy traffic is sent when the application does not produce sufficient traffic in time. While this approach could securely mitigate network side-channel leaks as well, it transmits traffic *continuously* at the average transmission rate of the target protocol. This either wastes bandwidth or causes significant latency overhead when the payload traffic is bursty. In contrast, Pacer allows adapting the transmission rate for every request based on public parameters of the request, thus limiting overheads on both bandwidth and latency.

Finally, unlike Pacer, the implementations of SkypeMorph and ScrambleSuit do not mask interference between the application and the pacing component. Hence, both implementations could suffer from leaks when integrated with the cloud server.

Oblivious computing Oblivious computing systems [34, 47, 106] generally prevent accessed memory *addresses* or accessed database *keys* from depending on secrets. Pacer addresses the orthogonal problem of making packet size and timing independent of secrets. As such, the techniques used in the two lines of work are completely different—oblivious computing generally relies on ORAM techniques, while Pacer relies on traffic shaping. However, Fletcher *et al.* [56] address *timing* leaks in ORAM accesses by pacing ORAM accesses. While this is superficially similar to Pacer's pacing of general network traffic, the pacing mechanism used by Fletcher *et al.* changes the pacing rate periodically based on the past actual request rate of the program, which may be secret-dependent. In contrast to Pacer's design, this leaks information.

4.10.5 Related work with non-security goals

Some work has focused on the *performance* effects, but *not the security* consequences of contention on network resources from co-located tenants. Pu *et al.* [142] measure the performance consequences of many side-channel interactions between co-located tenants. In particular, they conclude that co-locating I/O-intensive (specifically, network-intensive) tenants together degrades performance due to contention on switches and network events. Ongaro *et al.* [125] describe scheduler designs to minimize such problems. Chiang *et al.* [27] show how such interferences can be exploited adversarially by deliberately timing I/O at fine-granularity (in a black-box manner) to contend with a co-located victim's I/O and cause performance problems for it. To defend against such attacks, Richter *et al.* [143] propose to rate-limit the traffic from each VM by modifying the NIC's firmware. Pacer's traffic shaping can be similarly implemented in the NIC.

Silo [83] aims to improve remote access latency in a data center but also implements a traffic pacer in the hypervisor and uses dummy packets like Pacer. However, the purpose of Silo's pacer is to prevent flooding of the network (which can increase latencies). For this, Silo injects dummy packets into the NIC's queue; these packets are dropped at the next hop. In contrast, Pacer does not rate-limit NICs. Instead, it shapes the traffic just *before* it reaches the NIC.

MITTS [196] "shapes" memory traffic on CPU cores for performance and fairness, whereas Pacer shapes network traffic for security, so the goals and approaches are again very different.

Chapter 5

Conclusion

Contemporary online services collect and process diverse sensitive personal information. An important concern for the online services is to prevent unintended disclosure and misuse of sensitive data. This thesis outlined several challenges involved in building practical solutions against data disclosures and misuse. The challenges arise from the diversity of data policies, complexity of application systems, and the numerous threats of data disclosures and misuse.

This thesis presents systems that ensure compliance by design with data privacy and usage policies in online services. A compliance system specifies a threat model describing the adversary capabilities and channels over which data can be leaked, provides a framework to specify the application policies, and provides a mechanism to enforce the policies and prevent data leaks subject to the threat model. We present two compliance systems—Qapla and Pacer—that prevent data disclosures and misuse due to application bugs and network side channels respectively. We summarize the key ideas and contributions of the two systems below.

5.1 Summary of results

Qapla (Chapter 3) prevents direct disclosures in database-backed applications arising due to application bugs, specifically due to queries that violate data policies. These disclosures arise because the access control support in existing DBMSs is insufficient to support complex application policies; as a result, applications attempt to enforce the policies within their own code, which is an error-prone approach.

Qapla enables specification and enforcement of a rich class of data policies (including data linking and aggregation policies) in a DBMS-agnostic and application-transparent manner. It removes the often large and rapidly-evolving application

from the codebase trusted for compliance, and simplifies the design of both legacy and new applications by obviating the need for pervasive policy-enforcement code.

Pacer (Chapter 4) prevents network side-channel disclosures in cloud-hosted services, specifically leaks of data from a tenant to an adversary co-located on the same server or rack within the datacenter. Such disclosures are an important concern owing to the widespread use of public cloud systems; however, they haven't received much attention.

Pacer essentially enforces a single policy for a tenant that disallows any principal unauthorized to access sensitive data from even inferring the data indirectly, and specifically prevents data leaks through the network traffic shape (packet sizes, number, and timing). Pacer shapes the victim's traffic to make the traffic shape independent of the victim's secrets, and allows variations in the shapes based only on non-secrets, thus reconciling security with efficiency. Pacer's gray-box profiler computes the traffic shapes with minimal support from the tenant application. Pacer's novel abstraction of a cloaked tunnel eliminates network side-channel leaks by design, while its tunnel implementation integrated with a cloud server overcomes practical challenges regarding performance-isolation of the tunnel from the secret-dependent computations and handling network congestion conditions. Thus, Pacer demonstrates the design of a secure, efficient, and practical system that mitigates network side-channel leaks in cloud tenants by design and with minimal support from the tenant application.

5.2 Future work

Online services continue to evolve in response to new hardware, programming paradigms, and applications. Ensuring compliance with data policies in such a dynamic environment remains a challenging open problem. Below we discuss some challenges and opportunities for designing efficient compliance solutions specifically in the context of future cloud datacenter applications.

5.2.1 Compliance for next-generation cloud applications

cloud applications are evolving under two trends. First, the monolithic applications are being refactored into smaller *microservices* that communicate with each other to perform a single task (*e.g.*, processing a single client request). Second, hardware

vendors, such as Intel, are pushing towards *disaggregated hardware* that replaces traditional CPUs with separate compute, memory, and storage servers, which will result in further refactoring of datacenter applications. These changes present several interesting research challenges for ensuring end-to-end compliance in an application in terms of both specifying and enforcing data policies.

Applications may comprise several microservices from potentially untrusted vendors. To ensure end-to-end compliance in such applications, a system would need to enforce policies not only on what data individual microservices can access, but also on what data they send to or receive from other microservices. Tracking data flows at runtime would incur significant performance overheads due to the need to track several fine-grained data flows within individual microservices as well as across the different microservices of an application. Fortunately, the simpler design of individual microservices could make them more amenable to auditing and static analysis techniques, which can be used to reduce the runtime checks within the microservices. Thus, a hybrid solution combining static analyses and runtime data-flow tracking may provide an efficient end-to-end policy compliance solution for future cloud applications.

5.2.2 Efficient mitigation of side channels

Microservices and hardware disaggregation fundamentally also increase fine-grained sharing of physical resources among microservices of multiple tenants, making the threat of side channels even more pertinent, as an adversary can now observe individual microservices and infer information about an application's intermediate states that was previously hidden. Pacer's conceptual idea of traffic shaping can be extended to shape traffic along the entire communication graph of microservices of an application. However, Pacer's software implementation of traffic shaping is unlikely to scale to the high network loads resulting from the disaggregated software and hardware architecture. Specialized hardware, such as programmable NICs and switches could help to implement a system to prevent network side-channel disclosures securely and efficiently.

To conclude, as providers continue to build newer powerful, efficient, and scalable services, compliance systems need to evolve to provide principled solutions that prevent unintended data disclosures and misuse.

Bibliography

[1] *80% of robbers check Twitter, Facebook, Google Street View*. https://www.zdnet.com/article/infographic-80-of-robbers-check-twitter-facebook-google-street-view/. 2011.

[2] *A Special Price Just for You*. https://www.forbes.com/sites/neilhowe/2017/11/17/a-special-price-just-for-you/. 2017.

[3] O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert. "Predicting Secret Keys via Branch Prediction". In: *Cryptographers' Track at the RSA Conference*. 2007.

[4] Y. Agarwal, V. Murale, J. Hennessey, K. Hogan, and M. Varia. "Moving in Next Door: Network Flooding as a Side Channel in Cloud Environments". In: *International Conference on Cryptology and Network Security (CANS)*. 2016.

[5] *Apache HTTP Server*. http://httpd.apache.org/. Accessed 31 Aug 2020.

[6] M. Aron and P. Druschel. *TCP: Improving Startup Dynamics by Adaptive Timers and Congestion Control*. Tech. rep. 1998.

[7] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. *Enterprise Privacy Authorization Language (EPAL 1.2)*. http://www.w3.org/Submission/2003/SUBM-EPAL-20031110. 2003.

[8] A. Askarov, D. Zhang, and A. C. Myers. "Predictive Black-Box Mitigation of Timing Channels". In: *ACM Conference on Computer and Communications Security (CCS)*. 2010.

[9] M. Y. Becker, C. Fournet, and A. D. Gordon. "SecPAL: Design and Semantics of a Decentralized Authorization Language". In: *Journal of Computer Security (JCS)* 18.4 (2010).

[10] G. M. Bender, L. Kot, J. Gehrke, and C. Koch. "Fine-grained disclosure control for app ecosystems". In: *ACM International Conference on Management of Data (SIGMOD)*. 2013.

[11] D. J. Bernstein. *Cache-timing attacks on AES*. 2005.

[12] M. Bilal, H. Alsibyani, and M. Canini. "Mitigating Network Side Channel Leakage for Stream Processing Systems in Trusted Execution Environments". In: *ACM International Conference on Distributed and Event-based Systems (DEBS)*. 2018.

[13] J. Biskup. "History-Dependent Inference Control of Queries by Dynamic Policy Adaption". In: *Annual IFIP WG 11.3 Conference Data and Applications Security and Privacy (DBSec)*. 2011.

[14] A. Blankstein and M. J. Freedman. "Automating Isolation and Least Privilege in Web Services". In: *IEEE Symposium on Security and Privacy (S&P)*. 2014.

[15] B. A. Braun, S. Jana, and D. Boneh. *Robust and Efficient Elimination of Cache and Timing Side Channels*. http://arxiv.org/abs/1506.00189. 2015.

[16] A. Brodsky, C. Frakas, and S. Jajodia. "Secure Databases: Constraints, Inference Channels, and Monitoring Disclosures". In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 12.6 (2000).

[17] B. B. Brumley and N. Tuveri. "Remote Timing Attacks are Still Practical". In: *European Symposium on Research in Computer Security (ESORICS)*. 2011.

[18] D. Brumley and D. Boneh. "Remote Timing Attacks are Practical". In: *Computer Networks* 48.5 (2005).

[19] X. Cai, R. Nithyanand, and R. Johnson. "CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense". In: *ACM Workshop on Privacy in the Electronic Society (WPES)*. 2014.

[20] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. "Touching from a Distance: Website Fingerprinting Attacks and Defenses". In: *ACM Conference on Computer and Communications Security (CCS)*. 2012.

[21] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. "A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses". In: *ACM Conference on Computer and Communications Security (CCS)*. 2014.

[22] *California Consumer Privacy Act*. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375. 2019.

[23] *Capital One data breach*. https://www.cnet.com/news/capital-one-data-breach-involves-100-million-credit-card-applications. 2019.

[24] R. Chen, I. E. Akkus, and P. Francis. "SplitX: High-performance Private Analytics". In: *ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 2013.

[25] S. Chen, R. Wang, X. Wang, and K. Zhang. "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow". In: *IEEE Symposium on Security and Privacy (S&P)*. 2010.

[26] H. Cheng and R. Avnur. *Traffic Analysis of SSL Encrypted Web Browsing*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.1201. 1998.

[27] R. C. Chiang, S. Rajasekaran, N. Zhang, and H. H. Huang. "Swiper: Exploiting Virtual Machine Vulnerability in Third-Party Clouds with Competition for I/O Resources". In: *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 26.6 (2015).

[28] G. Chinis, P. Pratikakis, S. Ioannidis, and E. Athanasopoulos. "Practical Information Flow for Legacy Web Applications". In: *Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (ICOOOLPS)*. 2013.

[29] A. Chlipala. "Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications". In: *USENIX Conference on Operating Systems Design and Implementation (OSDI)*. 2010.

[30] S. Chong, K. Vikram, and A. C. Myers. "SIF: Enforcing Confidentiality and Integrity in Web Applications". In: *USENIX Security Symposium*. 2007.

[31] J. V. Cleemput, B. Coppens, and B. De Sutter. "Compiler Mitigations for Time Attacks on Modern x86 Processors". In: *ACM Transactions on Architecture and Code Optimizations (TACO)* 8.4 (2012).

[32] J. V. Cleemput, B. D. Sutter, and K. D. Bosschere. "Adaptive Compiler Strategies for Mitigating Timing Side Channel Attacks". In: *IEEE Transactions on Dependable and Secure Computing (TDSC)* 17.1 (2020).

[33] B. Coppens, I. Verbauwhede, K. D. Bosschere, and B. D. Sutter. "Practical Mitigations for Timing-Based Side-Channel Attacks on Modern x86 Processors". In: *IEEE Symposium on Security and Privacy (S&P)*. 2009.

[34] N. Crooks, M. Burke, E. Cecchetti, S. Harel, R. Agarwal, and L. Alvisi. "Obladi: Oblivious Serializable Transactions in the Cloud". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2018.

[35] M. Dalton, C. Kozyrakis, and N. Zeldovich. "Nemesis: Preventing Authentication & Access Control Vulnerabilities in Web Applications." In: *USENIX Security Symposium*. 2009.

[36] G. Danezis. *Traffic Analysis of the HTTP Protocol over TLS*. 2009.

[37] B. Davis and H. Chen. "DBTaint: Cross-application Information Flow Tracking via Databases". In: *USENIX Conference on Web Application development (WebApps)*. 2010.

[38] *Delivering Live YouTube Content via DASH*. https://developers.google.com/youtube/v3/live/guides/encoding-with-dash. Accessed 31 Aug 2020.

[39] *Django*. https://www.djangoproject.com/. Accessed 31 Aug 2020.

[40] *Django-CRM*. https://github.com/MicroPyramid/Django-CRM. Accessed 31 Aug 2020.

[41] *Drupal Commerce*. https://www.drupal.org/project/commerce. Accessed 31 Aug 2020.

[42] C. Dwork. "Differential Privacy". In: *International Colloquium on Automata, Languages and Programming (ICALP)*. 2006.

[43] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. "Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail". In: *IEEE Symposium on Security and Privacy (S&P)*. 2012.

[44] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. "Protocol Misidentification Made Easy with Format-Transforming Encryption". In: *ACM Conference on Computer and Communications Security (CCS)*. 2013.

[45] E. Elnikety, A. Mehta, A. Vahldiek-Oberwagner, D. Garg, and P. Druschel. "Thoth: Comprehensive Policy Compliance in Distributed Data Retrieval Systems". In: *USENIX Security Symposium*. 2016.

[46] *Equifax data breach*. https://www.cnet.com/news/equifax-data-leak-hits-nearly-half-of-the-us-population. 2017.

[47] S. Eskandarian and M. Zaharia. *ObliDB: Oblivious Query Processing for Secure Databases*. http://arxiv.org/abs/1710.00458. 2019.

[48] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh. "Jump over ASLR: Attacking Branch Predictors to Bypass ASLR". In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2016.

[49] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev. "Branchscope: A New Side-Channel Attack on Directional Branch Predictor". In: *ACM SIGPLAN Notices* 53.2 (2018).

[50] *eXtensible Access Control Markup Language (XACML) v3.0*. http://www.oasis-open.org/committees/xacml. Accessed 31 Aug 2020.

[51] K. Eykholt, A. Prakash, and B. Mozafari. "Ensuring Authorized Updates in Multi-User Database-Backed Applications". In: *USENIX Security Symposium*. 2017.

[52] *Facebook: Improved Search Could Surface Embarrassing Old Posts*. https://abcnews.go.com/Technology/facebook-improved-search-surface-embarrassing-posts/story?id=27469264. 2014.

[53] *Facebook Photo API bug*. https://www.zdnet.com/article/facebook-bug-exposed-private-photos-of-6-8-million-users. 2018.

[54] *Facebook says it 'unintentionally uploaded' 1.5 million people's email contacts without their consent*. https://www.businessinsider.com/facebook-uploaded-1-5-million-users-email-contacts-without-permission-2019-4. 2019.

[55] A. P. Felt, M. Finifter, J. Weinberger, and D. Wagner. "Diesel: Applying Privilege Separation to Database Access". In: *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. 2011.

[56] C. W. Fletchery, L. Ren, X. Yu, M. Van Dijk, O. Khan, and S. Devadas. "Suppressing the Oblivious RAM Timing Channel while Making Information Leakage and Program Efficiency Trade-offs". In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2014.

[57] W. Foundation. *enwiki HTML dump*. http://dumps.wikimedia.org/. Accessed 31 Aug 2020.

[58] Q. Ge, Y. Yarom, D. Cock, and G. Heiser. "A Survey of Microarchitectural Timing attacks and Countermeasures on Contemporary Hardware". In: *Journal of Cryptographic Engineering (JCEN)* 8 (2018).

[59] General Data Protection Regulation. "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". In: *Official Journal of the European Union* 59 (2016), pp. 1–88. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[60] D. B. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. C. Mitchell, and A. Russo. "Hails: Protecting Data Privacy in Untrusted Web Applications". In: *USENIX Conference on Operating Systems Design and Implementation (OSDI)*. 2012.

[61] V. D. Gligor. *A Guide to Understanding Covert Channel Analysis of Trusted Systems*. Vol. 30. https://apps.dtic.mil/dtic/tr/fulltext/u2/a477651.pdf. National Computer Security Center, 1994.

[62] X. Gong and N. Kiyavash. "Quantifying the Information Leakage in Timing Side Channels in Deterministic Work-conserving Schedulers". In: *IEEE/ACM Transactions on Networking (TON)* 24.3 (2016).

[63] X. Gong, N. Borisov, N. Kiyavash, and N. Schear. "Website Detection Using Remote Traffic Analysis". In: *Privacy Enhancing Technologies Symposium (PETS)*. 2012.

[64] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[65] A. Grünbacher. "POSIX Access Control Lists on Linux". In: *USENIX Annual Technical Conference (ATC), FREENIX Track*. 2003.

[66] M. Guarnieri, S. Marinovic, and D. A. Basin. *Strong and Provably Secure Database Access Control*. http://arxiv.org/abs/1512.01479. 2015.

[67] J. Hayes and G. Danezis. "k-fingerprinting: A Robust Scalable Website Fingerprinting Technique". In: *USENIX Security Symposium*. 2016.

[68] T. Haynes and D. Noveck. "Network File System (NFS) v4 Protocol". In: *RFC 7530* (2015).

[69] *Heartbleed (CVE-2014-0160)*. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160. 2014.

[70] *Here's everything a cyber criminal can do if they steal your credit card*. https://www.cnbc.com/2019/09/26/heres-everything-cyber-criminals-can-do-if-they-steal-your-credit-card.html. 2019.

[71] D. Herrmann, R. Wendolsky, and H. Federrath. "Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-bayes Classifier". In: *ACM Cloud Computing Security Workshop (CCSW)*. 2009.

[72] *Hierarchical Token Bucket for Linux*. http://luxik.cdi.cz/~devik/qos/htb. Accessed 31 Aug 2020.

[73] A. Hintz. "Fingerprinting Websites Using Traffic Analysis". In: *International Conference on Privacy Enhancing Technologies (PET)*. 2002.

[74] *HotCRP Conference Management Software*. http://www.read.seas.harvard.edu/~kohler/hotcrp. Accessed 31 Aug 2020.

[75] *HotCRP release news*. http://read.seas.harvard.edu/~kohler/hotcrp/news.html. Accessed 31 Aug 2020.

[76] *HTTP Live Streaming (HLS)*. https://en.wikipedia.org/wiki/HTTP_Live_Streaming. Accessed 31 Aug 2020.

[77] *Implementing row level security in MySQL*. https://www.sqlmaestro.com/en/resources/all/row_level_security_mysql. Accessed 31 Aug 2020.

[78] M. S. Inci, B. Gülmezoglu, G. I. Apecechea, T. Eisenbarth, and B. Sunar. "Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud". In: *IACR Cryptology ePrint Archive* 2015.1-15 (2015).

[79] M. S. İnci, G. Irazoqui, T. Eisenbarth, and B. Sunar. "Efficient, adversarial neighbor discovery using logical channels on Microsoft Azure". In: *Annual Conference on Computer Security Applications (ACSAC)*. 2016.

[80] G. Irazoqui, T. Eisenbarth, and B. Sunar. "S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing–and Its Application to AES". In: *IEEE Symposium on Security and Privacy (S&P)*. 2015.

[81] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar. "Fine grain Cross-VM Attacks on Xen and VMware are possible!" In: *IEEE International Conference on Big Data and Cloud Computing (BDCLOUD)*. 2014.

[82] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar. "Wait a minute! A fast, Cross-VM attack on AES". In: *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. 2014.

[83] K. Jang, J. Sherry, H. Ballani, and T. Moncaster. "Silo: Predictable Message Latency in the Cloud". In: *ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 2015.

[84] S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam. "Mitigating Timing Side Channel in Shared Schedulers". In: *IEEE/ACM Transactions on Networking (TON)* 24.3 (2016).

[85] S. Kanav, P. Lammich, and A. Popescu. "A Conference Management System with Verified Document Confidentiality". In: *International Conference on Computer Aided Verification (CAV)*. 2014.

[86] S Kent and K Seo. *Security Architecture for IP*. https://tools.ietf.org/html/rfc4301. 2005.

[87] *Keras API*. https://keras.io/. Accessed 31 Aug 2020.

[88] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. http://arxiv.org/abs/1412.6980. 2014.

[89] P. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology – CRYPTO*. 1996.

[90] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi. "NetCAT: Practical Cache Attacks from the Network". In: *IEEE Symposium on Security and Privacy (S&P)*. 2019.

[91] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. "The QUIC Transport Protocol: Design and Internet-Scale Deployment". In: *ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 2017.

[92] D. Lazar, Y. Gilad, and N. Zeldovich. "Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2018.

[93] D. Lazar, Y. Gilad, and N. Zeldovich. "Yodel: Strong Metadata Security for Voice Calls". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2019.

[94] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. "Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems". In: *ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. 2015.

[95] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. "Inferring Fine-Grained Control Flow Inside SGX Enclaves with Branch Shadowing". In: *USENIX Security Symposium*. 2017.

[96] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y.-r. Xu, and D. DeWitt. "Limiting Disclosure in Hippocratic Databases". In: *International Conference on Very Large Data Bases (VLDB)*. 2004.

[97] N. Li and J. C. Mitchell. "Datalog with Constraints: A Foundation for Trust Management Languages". In: *International Symposium on Practical Aspects of Declarative Languages (PADL)*. 2003.

[98] P. Li, D. Gao, and M. K. Reiter. "Stopwatch: A Cloud Architecture for Timing Channel Mitigation". In: *ACM Transactions on Information and System Security (TISSEC)* 17.2 (2014).

[99] P. Li and S. Zdancewic. "Practical Information-flow Control in Web-Based Information Systems". In: *IEEE Workshop on Computer Security Foundations (CSFW)*. 2005.

[100] S. Li, H. Guo, and N. Hopper. "Measuring Information Leakage in Website Fingerprinting Attacks and Defenses". In: *ACM Conference on Computer and Communications Security (CCS)*. 2018.

[101] M. Liberatore and B. N. Levine. "Inferring the Source of Encrypted HTTP Connections". In: *ACM Conference on Computer and Communications Security (CCS)*. 2006.

[102] J. Litton, A. Vahldiek-Oberwagner, E. Elnikety, D. Garg, B. Bhattacharjee, and P. Druschel. "Light-Weight Contexts: An OS Abstraction for Safety and Performance". In: *USENIX Conference on Operating Systems Design and Implementation (OSDI)*. 2016.

[103] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. "Last-Level Cache Side-Channel Attacks are Practical". In: *IEEE Symposium on Security and Privacy (S&P)*. 2015.

[104] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee. "Catalyst: Defeating Last-Level Cache Side Channel Attacks in Cloud Computing". In: *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2016.

[105] W. Liu, D. Gao, and M. K. Reiter. "On-Demand Time Blurring to Support Side-Channel Defense". In: *European Symposium on Research in Computer Security (ESORICS)*. 2017.

[106] J. R. Lorch, B. Parno, J. Mickens, M. Raykova, and J. Schiffman. "Shroud: Ensuring Private Access to Large-Scale Data in the Data Center". In: *USENIX Conference on File and Storage Technologies (FAST)*. 2013.

[107] D. Lu, S. Bhat, A. Kwon, and S. Devadas. "DynaFlow: An Efficient Website Fingerprinting Defense Based on Dynamically-Adjusting Flows". In: *ACM Workshop on Privacy in the Electronic Society (WPES)*. 2018.

[108] P. D. Marinescu, C. Perry, M. Pomarole, T. Yuan, P. Tague, and I. Papagiannis. "IVD: Automatic Learning and Enforcement of Authorization Rules in Online Social Networks". In: *IEEE Symposium on Security and Privacy (S&P)*. 2017.

[109] *Marriott data breach*. https://www.cnet.com/news/marriott-data-breach-impacts-500-million-starwood-hotel-guests. 2018.

[110] R. Martin, J. Demme, and S. Sethumadhavan. "TimeWarp: Rethinking Timekeeping and Performance Monitoring Mechanisms to Mitigate Side-channel Attacks". In: *International Symposium on Computer Architecture (ISCA)*. 2012.

[111] F. McSherry. "Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis". In: *ACM International Conference on Management of Data (SIGMOD)*. 2009.

[112] *MediaWiki*. https://www.mediawiki.org/wiki/MediaWiki_1.27. Accessed 31 Aug 2020.

[113] A. Mehta, E. Elnikety, K. Harvey, D. Garg, and P. Druschel. "Qapla: policy compliance in database-backed applications". In: *USENIX Security Symposium*. 2017.

[114] A. Mehta, M. Alzayat, R. D. Viti, B. Brandenburg, P. Druschel, and D. Garg. *Pacer: Network Side-Channel Mitigation in the Cloud*. http://arxiv.org/abs/1908.11568. 2019.

[115] *Microsoft accidently exposed 250 million customer service records*. https://www.engadget.com/2020-01-22-microsoft-database-exposure.html. 2020.

[116] *Millions of Facebook user phone numbers exposed online*. https://www.cnet.com/news/millions-of-facebook-user-phone-numbers-exposed-online-security-researchers-say. 2019.

[117] *Min-Max Normalization*. https://en.wikipedia.org/wiki/Feature_scaling#Rescaling_(min-max_normalization). Accessed 31 Aug 2020.

[118] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg. "Skypemorph: Protocol Obfuscation for Tor Bridges". In: *ACM Conference on Computer and Communications Security (CCS)*. 2012.

[119] *MPEG-DASH*. https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP. Accessed 31 Aug 2020.

[120] *MySQL 5.7.16*. https://dev.mysql.com/downloads/mysql. Accessed 31 Aug 2020.

[121] *MySQL Workbench*. http://mysqlworkbench.org/. Accessed 31 Aug 2020.

[122] *NapaTech SmartNIC, Feature Overview Data Sheet*. https://www.napatech.com/support/resources/data-sheets/napatech-smartnic-feature-overview/. Accessed 31 Aug 2020.

[123] R. Nithyanand, X. Cai, and R. Johnson. "Glove: A Bespoke Website Fingerprinting Defense". In: *ACM Workshop on Privacy in the Electronic Society (WPES)*. 2014.

[124] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. "Oblivious Multi-Party Machine Learning on Trusted Processors". In: *USENIX Security Symposium*. 2016.

[125] D. Ongaro, A. L. Cox, and S. Rixner. "Scheduling I/O in virtual machine monitors". In: *ACM International Conference on Virtual Execution Environments (VEE)*. 2008.

[126] *OpenCart*. https://github.com/opencart/opencart. Accessed 31 Aug 2020.

[127] *OpenSource Social Network*. https://github.com/opensource-socialnetwork/opensource-socialnetwork. Accessed 31 Aug 2020.

[128] *OpenSSL 1.1.1b*. https://github.com/openssl/openssl/releases/tag/OpenSSL_1_1_1b. Accessed 31 Aug 2020.

[129] *Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length*. https://bitmovin.com/mpeg-dash-hls-segment-length/. Accessed 31 Aug 2020.

[130] *Oracle Transparent Sensitive Data Protection*. https://docs.oracle.com/database/121/DBSEG/tsdp.htm#DBSEG855. Accessed 31 Aug 2020.

[131] D. A. Osvik, A. Shamir, and E. Tromer. "Cache Attacks and Countermeasures: The Case of AES". In: *The Cryptographers' Track at the RSA Conference on Topics in Cryptology (CT-RSA)*. 2006.

[132] D. Page. *Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel*. http://www.cs.bris.ac.uk/Publications/pub_info.jsp?id=1000625. 2002.

[133] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. "Website Fingerprinting in Onion Routing Based Anonymization Networks". In: *ACM Workshop on Privacy in the Electronic Society (WPES)*. 2011.

[134] B. Parno, J. M. McCune, D. Wendlandt, D. G. Andersen, and A. Perrig. "CLAMP: Practical Prevention of Large-Scale Data Leaks". In: *IEEE Symposium on Security and Privacy (S&P)*. 2009.

[135] C. Percival. "Cache Missing for Fun and Profit". In: *BSDCan*. 2005.

[136] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard. "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks". In: *USENIX Security Symposium*. 2016.

[137] *PHP Data Objects (PDO)*. http://php.net/manual/en/intro.pdo.php. Accessed 31 Aug 2020.

[138] *PHP MySQL Improved Extension*. http://php.net/manual/en/book.mysqli.php. Accessed 31 Aug 2020.

[139] A. Pimlott and O. Kiselyov. "Soutei, a Logic-Based Trust-Management System". In: *International Symposium on Functional and Logic Programming (FLOPS)*. 2006.

[140] *PostgreSQL 9.5.3 Documentation*. https://www.postgresql.org/docs/current/static/ddl-rowsecurity.html. Accessed 31 Aug 2020.

[141] *Protect Your Data: Row-level Security in MariaDB 10.0*. https://mariadb.com/blog/protect-your-data-row-level-security-mariadb-100. 2015.

[142] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, and Y. Cao. "Who Is Your Neighbor: Net I/O Performance Interference in Virtualized Clouds". In: *IEEE Transactions on Services Computing (TSC)* 6.3 (2013).

[143] A. Richter, C. Herber, S. Wallentowitz, T. Wild, and A. Herkersdorf. "A Hardware/Software Approach for Mitigating Performance Interference Effects in Virtualized Environments Using SR-IOV". In: *IEEE International Conference on Cloud Computing (CLOUD)*. 2015.

[144] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds". In: *ACM Conference on Computer and Communications Security (CCS)*. 2009.

[145] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. "Extending Query Rewriting Techniques for Fine-grained Access Control". In: *ACM International Conference on Management of Data (SIGMOD)*. 2004.

[146] E. T. Roei Schuster Vitaly Shmatikov. "Beauty and the Burst: Remote Identification of Encrypted Video Streams". In: *USENIX Security Symposium*. 2017.

[147] *Row and Column Access Control Support in IBM DB2 for i*. http://www.redbooks.ibm.com/redpapers/pdfs/redp5110.pdf. 2014.

[148] T. S. Saponas, J. Lester, C. Hartung, S. Agarwal, T. Kohno, et al. "Devices That Tell On You: Privacy Trends in Consumer Ubiquitous Computing". In: *USENIX Security Symposium*. 2007.

[149] D. Schoepe, D. Hedin, and A. Sabelfeld. "SeLINQ: Tracking Information Across Application-database Boundaries". In: *SIGPLAN Notes* 49.9 (2014).

[150] D. Schultz and B. Liskov. "IFDB: Decentralized Information Flow Control for Databases". In: *ACM European Conference on Computer Systems (EuroSys)*. 2013.

[151] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript". In: *International Conference on Financial Cryptography and Data Security (FC)*. 2017, pp. 247–267.

[152] M. Schwarz, M. Schwarzl, M. Lipp, J. Masters, and D. Gruss. "NetSpectre: Read Arbitrary Memory over Network". In: *European Symposium on Research in Computer Security (ESORICS)*. 2019.

[153] D. X. Song, D. Wagner, and X. Tian. "Timing Analysis of Keystrokes and Timing Attacks on SSH". In: *USENIX Security Symposium*. 2001.

[154] R. Sprabery, K. Evchenko, A. Raj, R. B. Bobba, S. Mohan, and R. Campbell. "Scheduling, Isolation, and Cache Allocation: A Side-Channel Defense". In: *IEEE International Conference on Cloud Engineering (IC2E)*. 2018.

[155] *SQL Server 2016 Technical Documentation*. https://docs.microsoft.com/en-us/sql/relational-databases/security/row-level-security?view=sql-server-ver15. Accessed 31 Aug 2020.

[156] *SQL Server Dynamic Data Masking*. https://docs.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-ver15. Accessed 31 Aug 2020.

[157] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. "Statistical Identification of Encrypted Web Browsing Traffic". In: *IEEE Symposium on Security and Privacy (S&P)*. 2002.

[158] L. Sweeney. "K-anonymity: A Model for Protecting Privacy". In: *International Journal of Uncertainty Fuzziness Knowledge-Based Systems* 10.5 (2002).

[159] J. Szefer. "Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses". In: *Journal of Hardware and Systems Security* 3.3 (2019).

[160] *The Cambridge Analytica Scandal*. https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election. 2018.

[161] *The Virtual Private Database in Oracle9iR2*. http://www.cgisecurity.com/database/oracle/pdf/VPD9ir2twp.pdf. 2002.

[162] T. S. Toland, C. Frakas, and C. M. Eastman. "The Inference Problem: Maintaining Maximal Availability in the Presence of Database Updates". In: *Computers and Security Journal* 29.1 (2010).

[163] U. Turan and I. H. Toroslu. *Privacy Preserving Secure Decomposition Algorithm for Attribute Based Access Control Mechanism*. http://arxiv.org/abs/1402.5742. 2014.

[164] L. Uhsadel, A. Georges, and I. Verbauwhede. "Exploiting Hardware Performance Counters". In: *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2008.

[165] P. Upadhyaya, M. Balazinska, and D. Suciu. "Automatic Enforcement of Data Use Policies with DataLawyer". In: *ACM International Conference on Management of Data (SIGMOD)*. 2015.

[166] A. Vahldiek-Oberwagner, E. Elnikety, A. Mehta, D. Garg, P. Druschel, A. Post, R. Rodriguez, and J. Gehrke. "Guardat: Enforcing data policies at the storage layer". In: *European Conference on Computing Systems (EuroSys)*. 2015.

[167] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. Wenisch, Y. Yarom, and R. Strackx. "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution". In: *USENIX Security Symposium*. 2018.

[168] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. "Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2015.

[169] V. Varadarajan, T. Ristenpart, and M. Swift. "Scheduler-based Defenses against Cross-VM Side-channels". In: *USENIX Security Symposium*. 2014.

[170] B. C. Vattikonda, S. Das, and H. Shacham. "Eliminating Fine Grained Timers in Xen". In: *ACM Cloud Computing Security Workshop (CCSW)*. 2011.

[171] B. C. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren. "Practical TDMA for Datacenter Ethernet". In: *ACM European Conference on Computer Systems (EuroSys)*. 2012.

[172] P. Vila and B. Köpf. "Loophole: Timing Attacks on Shared Event Loops in Chrome". In: *USENIX Security Symposium*. 2017.

[173] D. Volpano and G. Smith. "Probabilistic Noninterference in a Concurrent Language". In: *Journal of Computer Security (JCS)* 7.2-3 (1999).

[174] T. Wang and I. Goldberg. "Improved Website Fingerprinting on Tor". In: *ACM Workshop on Privacy in the Electronic Society (WPES)*. 2013.

[175] T. Wang and I. Goldberg. "Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks". In: *USENIX Security Symposium*. 2017.

[176] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. "Effective Attacks and Provable Defenses for Website Fingerprinting". In: *USENIX Security Symposium*. 2014.

[177] P. Winter, T. Pulls, and J. Fuss. "ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship". In: *ACM Workshop on Privacy in the Electronic Society (WPES)*. 2013.

[178] C. V. Wright, S. E. Coull, and F. Monrose. "Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis". In: *Network and Distributed System Security Symposium (NDSS)*. 2009.

[179] C. V. Wright, F. Monrose, and G. M. Masson. "On Inferring Application Protocol Behaviors in Encrypted Network Traffic". In: *Journal of Machine Learning Research (JMLR)* 7 (2006).

[180] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. "Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations". In: *IEEE Symposium on Security and Privacy (S&P)*. 2008.

[181] *wrk2: A constant throughput, correct latency recording variant of wrk*. https://github.com/giltene/wrk2. Accessed 31 Aug 2020.

[182] W. Wu and B. Ford. "Deterministically Deterring Timing Attacks in Deterland". In: *Conference on Timely Results in Operating Systems (TRIOS)*. 2015.

[183] *XACML Products and Deployments*. http://docs.oasis-open.org/xacml/xacmlRefs.html#Products. Accessed 31 Aug 2020.

[184] *Xen hypervisor*. https://xenproject.org/. Accessed 31 Aug 2020.

[185] *Xen Null scheduler*. https://patchwork.kernel.org/patch/9669405/. 2017.

[186] Y. Xu, W. Cui, and M. Peinado. "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems". In: *IEEE Symposium on Security and Privacy (S&P)*. 2015.

[187] J. Yang, T. Hance, T. H. Austin, A. Solar-Lezama, C. Flanagan, and S. Chong. "Precise, Dynamic Information Flow for Database-backed Applications". In: *ACM Conference on Programming Language Design and Implementation (PLDI)*. 2016.

[188] Y. Yarom and K. Falkner. "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack". In: *USENIX Security Symposium*. 2014.

[189] Y. Yarom, D. Genkin, and N. Heninger. "CacheBleed: a timing attack on OpenSSL constant-time RSA". In: *Journal of Cryptographic Engineering (JCEN)* 7.2 (2017).

[190] A. Yip, X. Wang, N. Zeldovich, and M. F. Kaashoek. "Improving Application Security with Data Flow Assertions". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2009.

[191] S. Zander, G. Armitage, and P. Branch. "A survey of covert channels and countermeasures in computer network protocols". In: *IEEE Communications Surveys & Tutorials* 9.3 (2007).

[192] D. Zhang, A. Askarov, and A. C. Myers. "Predictive Mitigation of Timing Channels in Interactive Systems". In: *ACM Conference on Computer and Communications Security (CCS)*. 2011.

[193] X. Zhang, J. Hamm, M. K. Reiter, and Y. Zhang. "Statistical Privacy for Streaming Traffic". In: *Network and Distributed Systems Security (NDSS)*. 2019.

[194] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. "Cross-VM side channels and their use to extract private keys". In: *ACM Conference on Computer and Communications Security (CCS)*. 2012.

[195] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. "Cross-VM Side Channels and Their Use to Extract Private Keys". In: *ACM Conference on Computer and Communications Security (CCS)*. 2012.

[196] Y. Zhou and D. Wentzlaff. "MITTS: Memory Inter-arrival Time Traffic Shaping". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016).

[197] Z. Zhou, M. K. Reiter, and Y. Zhang. "A Software Approach to Defeating Side Channels in Last-Level Caches". In: *ACM Conference on Computer and Communications Security (CCS)*. 2016.