Research Statement

Wen Zhang

Application development technology is constantly evolving, pushed by *new computing environments and capabilities* (e.g., serverless computing, persistent memory) and pulled by *more stringent societal requirements* (e.g., around data privacy). But today's applications are struggling to catch up: having been built using the technology of yesterday, they are unable to meet the demands of tomorrow. Even though researchers have been developing new technologies to tackle emerging demands, these solutions typically require re-implementing existing applications and thus cannot support today's installed base.[1]

My PhD research has taken an alternative approach: developing systems that can adapt existing applications or abstractions to meet emerging demands and leverage new technologies. For example, **Blockaid** [40] and **Ote** [42] enable fine-grained database access-control on existing web applications, thereby enabling the desired level of data privacy; **Persimmon** [39] adds persistence to existing in-memory storage systems using persistent memory; and **Kappa** [38] enables developers to program serverless functions using existing concurrency abstractions.

To be clear, this research direction of "enhancing existing solutions for new demands" is not one that I declared when I started graduate school, and nor am I wedded to it going forward. But it is what I observe to be a recurring theme of my work, and I believe that by adapting currently deployed systems to meet current needs, hopefully in a principled and provably correct manner, I will have a better shot at developing solutions that are adoptable by practitioners.

Database access-control for today's web applications

Many web applications serve sensitive user data, whose access is governed by *access-control policies*. To enforce these policies, today's back-end web developers must craft appropriate filter predicates for their database queries, and wrap queries within appropriate access checks in their code. But this practice is error-prone, and access-check bugs have led to sensitive data leakages in many production systems [14, 23, 44]. So why has it been so difficult to prevent unintended data disclosure in today's applications?

To begin with, most existing applications don't even have an explicit, precise policy written down. Instead, a web application's access-control policy is commonly reflected: (1) in the enforcement logic in application code, implicitly-embedded and scattered, (2) in natural-language documentation (imprecise, and rarely comprehensive), and (3) in the developer's head. As a result, nobody other than the developer is likely to comprehend the entire policy—a problem that worsens as the development team evolves. The solution cannot simply be to "write the program's policy down". I have manually written policies for three existing applications—a painstakingly tedious process that, as I would later discover, still resulted in policies containing quite a few errors. Clearly, we need a better way!

But even if we had an explicit policy, enforcing it on an existing application is still nontrivial. Many enforcement solutions in the literature turn out to be incompatible with existing applications: Some apply only to applications written in a specialized language [4, 17]; others resort to silently altering query results, which may lead to unexpected or misleading outcomes [16, 22, 24, 36]. Perhaps as a result of this, a common practice today is still to grant privileges on entire tables to an entire application—much too coarse-grained to prevent data leaks within an application.

To address these pain points, I developed two systems: Ote and Blockaid.

Ote: Policy extraction

Ote [42] is a *policy-extraction* tool that bootstraps policy creation for existing web applications. It extracts an application's implicitly-embedded policy by enumerating and summarizing its possible data accesses. The extracted policy is expressed as *view definitions*—i.e., SQL `SELECT` statements specifying the information accessible to a given user.

A human can then audit the extracted policy to ensure they are within the bounds of intended data revelations; if this is not the case, the application likely has an access-check bug. Once audited and adjusted, the extracted policy can be adopted as the application's final policy, and can optionally be enforced to (1) cover any application queries missed by Ote,[2] and (2) ensure the application's continued compliance as its code changes.

Ote works in two steps. First, it explores execution paths through the application via concolic execution, producing transcripts that record the SQL queries issued and conditions for their issuance. Second, it merges and simplifies the

[1]Case in point, here is a November, 2024, article on why a development team gave up on migrating from "legacy" Ruby on Rails to newer technology despite great determination: `https://dirkjonker.bearblog.dev/rewrite-it-in-rails/` (accessed Nov 1, 2024).

[2]Like many program-analysis tools, Ote cannot guarantee that the extracted policy covers all queries that the application can possibly issue. In contrast, access-control enforcers (like Blockaid below) typically *guarantee* that every data access is checked against the policy.

transcripts to produce a policy allowing each query to be issued under its recorded conditions. We tailor Ote's concolic execution to track only operations that commonly govern query issuance, allowing it to scale to real-world code bases.

We used Ote to extract policies for three open-source Ruby-on-Rails applications, two of which I had manually written policies for years earlier. Ote not only slashed the burden of manual policy-writing, but also produced more accurate policies. To wit, by reviewing the extracted policies I uncovered several errors in my handwritten counterparts, including some overly-permissive views that could reveal sensitive data to unauthorized users.

Without a policy, access-control is a nonstarter for complex existing applications. By streamlining policy creation, Ote can be the catalyst to get access-control adoption off the ground.

Blockaid: Policy enforcement

Given a policy—whether it be handwritten, derived from an Ote-extracted policy, or from elsewhere—Blockaid [40] enforces it on an application's database queries by acting as a SQL proxy. Blockaid has two distinguishing features: (1) it works with existing applications, and (2) it maintains *semantic transparency*—by fully answering queries that comply with the policy and outright blocking queries that do not.

Blockaid checks each query in the context of past queries. For example, it may allow a query for Calendar Event 42 to be answered, but only if a prior query has established that the current user is invited to that event. We formalize this query-compliance criterion as a generalization of *query determinacy* [28], and Blockaid checks it via SMT solving.

Because query checking lies on the application's request-handling path, it must be made very fast; Blockaid does so via *generalization-based caching*. Once a query has been deemed compliant in some context by an SMT solver, Blockaid generalizes this determination into a *decision template* by finding a small unsat core. This way, any future query (along with its context) that matches the template can be declared compliant without SMT solving. With this optimization, Blockaid achieved an overhead of 2 % to 12 % when applied to three existing Ruby-on-Rails applications.

Other work

Persimmon Persistent memory (PM) offers fast persistence, but systems that exploit it are usually implemented from scratch. To make PM more accessible, we developed Persimmon [39], a system that converts an existing in-memory distributed storage system into a fast, persistent version while requiring minimal code changes. Persimmon targets state-machine applications that process a sequence of deterministic client commands. It keeps two copies of the state, one in DRAM and one in PM. A command is executed on the DRAM state and also logged persistently; in the background, it is applied to the PM state in a crash-consistent manner. We applied Persimmon to Redis and TAPIR [37] using less than 150 lines of code each; the resulting systems are persistent and fast on Intel's PM hardware.

Kappa Serverless computing has been applied in diverse domains like video processing [7] and data analytics [12], but *general-purpose serverless programming* remained challenging for developers, who had to partition computation into small chunks and had no concurrency primitives at their disposal. The Kappa framework [38] provides a familiar programming model for general-purpose serverless programming. Developers write standard Python code using the familiar concurrency abstractions of tasks and futures; the Kappa coordinator manages execution on an unmodified serverless platform; and the runtime provides fault tolerance via language-level checkpointing. Kappa demonstrated competitive performance in serverless data analytics and enabled complex serverless applications like a web crawler.

Control replication While certainly not planned, my undergraduate research also aligns with the theme of adapting existing abstractions to new environments. I helped develop *control replication* [34], a compiler technique for transforming implicitly-parallel code into scalable SPMD [6] code. Implemented for the Regent language [33], it delivered speed on par with handwritten SPMD code, while preserving the productivity of implicitly-parallel programming.

Ongoing work I am currently collaborating on the Pringles project (led by Micah Murray) for implementing a scalable datacenter-wide distributed shared log [3] using programmable switches, and on the Fava project (led by Sam Son and Zhihong Luo) for efficiently executing existing Java applications on tiered memory—e.g., with CXL. In both projects, we're aiming to adapt applications written using existing abstractions to exploit new hardware capabilities.

Future directions

Access control

In our quest to make access-control work for existing applications, there remain many open problems that are both intellectually intriguing and practically important. Here, I will highlight a few that I plan to work on; a more detailed discussion on some of these issues can be found in our HotOS paper [41].

Policy comprehension While (a subset of) SQL as a policy language is precise and familiar, it can be verbose for complex policies. I am interested in designing a more concise policy domain-specific language (DSL) that desugars into SQL. To ensure that the DSL is comprehensible and easy to learn, I am excited to collaborate with colleagues in human-computer interaction, as well as borrow ideas from user studies on query languages [1, 31] and from the design of object-relational mappings [35]. Once a DSL is in place, a policy-extraction tool must convert extracted SQL views into this DSL; I plan to implement this conversion using techniques like verified lifting [13, 15].

Policy testing Even a meticulously written policy can contain errors: A small typo might vastly expand the information revealed by a view, and views that individually look reasonable, when put together, might lead to *partial disclosure* of sensitive information [25]. I am interested in developing automated tools that test a policy for such errors. To catch obvious common-sense errors,[3] I plan to enlist the help of large-language models (LLMs), which are known to possess SQL knowledge [27] and at least a modicum of common sense [19]. I will experiment with prompting an LLM with an application description, a database schema, and policy view definitions, and asking if anything stands out. For finer-grained detection of partial disclosure, I plan to identify disclosure criteria that are easy to understand (a good candidate is *prior-agnostic privacy criteria* [41, § 4.3]), and to develop algorithms to check these criteria.

Violation diagnosis When an application query gets blocked due to a policy violation, how might a tool provide useful feedback to help diagnose this violation? This is a tricky issue: "Explaining" a violation is difficult because, with an allow-list policy, no subset of the policy can be singled out for *causing* the violation. It might be more useful to directly propose *fixes*: policy- or application-patches that would allow the offending query to go through. I plan to explore the usefulness of different strategies for generating patches—e.g., replacing the offending query with a *contained rewriting* using the policy views [18].

Decidable compliance checking SMT solvers have proven effective for checking query compliance in real-world scenarios, but it remains theoretically unclear if compliance checking is decidable in these cases. Even for single-query checking, where Blockaid's criterion degenerates into query determinacy, our understanding still remains "at the extremes". On one side, I proved that query determinacy is decidable for project-select views and a project-select-join query with no self joins, provided the selection formulas are reasonable [43]; but this result is too restricted for practical use. On the other side, query determinacy is undecidable for conjunctive queries [9, 10], yet practical views and queries look far simpler than those constructed in the undecidability proofs. So I wonder: Is there a natural class of views and queries—reflecting those in practice—for which compliance checking is decidable? An affirmative answer could lead to explicit compliance-checking algorithms with more stable performance than SMT solving [30].

Other topics

SMT caching Many systems today rely on SMT solvers to solve computationally hard problems. Given the effectiveness of Blockaid's caching, I am hopeful that generalization-based caching can speed up other solver-based systems that repeatedly dispatch structurally-similar SMT queries. I'm interested in developing a *generic SMT caching layer* applicable to diverse problem domains. This would require designing a suitable API—ideally higher-level than SMT formulas but still versatile. Furthermore, we must address privacy concerns that arise if SMT queries come from different tenants (like for Amazon's Zelkova [2]): When a cache entry is produced via generalization, can it leak too much information about the SMT query from which it originated?

The power of synchronized clocks Recent work has shown that it is possible to synchronize clocks precisely in the datacenter using commodity hardware [8, 20, 26]. This led me to wonder: For distributed computing, what power is fundamentally gained by assuming synchronized clocks? Concretely, consider the asynchronous message-passing model where nodes can be equipped with perfectly- or loosely-synchronized clocks. Can we characterize the classes of problems (1) whose solution is not helped by having clocks at all?[4] (2) that can be solved *more efficiently* using clocks [21]? (3) that are solvable *only with* clocks? Answers to these questions may help both designers of clock synchronization and designers of distributed protocols better understand the inherent power and limitations of clocks.

Trace minimization for troubleshooting distributed systems When a bug is triggered in a distributed system after a lengthy execution, generating a *minimal event trace* that reproduces the bug can significantly aid in debugging. However, trace minimization is challenging due to the astronomical number of possible traces; one state-of-the-art solution, DEMi, relies on developer-provided heuristics to guide the search but may still take hours to complete [32]. I am interested in making trace minimization faster and more automated by leveraging two recent developments: (1) the

[3] I once made a typo in a policy view that made it read, "if the user is an instructor for this course, the user can access grades in all courses".
[4] Neiger and Toueg [29] studied this question from the solvability perspective, but did not consider performance.

existence of formal specifications for distributed systems that closely align with their implementations [5, 11], and (2) large language models, which are increasingly used to comprehend code.

Application platform for tiered memory With the rise of CXL, datacenters with tiered memory are quickly becoming a reality: each node is equipped with memory expansion modules, and nodes within a rack also share remote memory blades. I am interested in developing an *integrated application platform* tailored for tiered-memory datacenters; this platform would include a programming model, a language runtime, and an orchestrator. Breaking from my usual focus on supporting existing applications, I plan to design the platform from the ground up by addressing three key questions: (1) What is a good programming abstraction for heterogeneous memory (fast/slow, private/shared)? (2) How to efficiently implement runtime functionality like synchronization and garbage collection? (3) How to place or migrate jobs to maximize resource utilization, considering the memory available in each tier?

References

[1] A. Ahadi, J. C. Prior, V. Behbood, and R. Lister. A quantitative study of the relative difficulty for novices of writing seven different types of SQL queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, ITiCS 2015*. ACM, 2015. URL https://doi.org/10.1145/2729094.2742620.

[2] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. S. Luckow, N. Rungta, O. Tkachuk, and C. Varming. Semantic-based automated reasoning for AWS access policies using SMT. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018*. IEEE, 2018. URL https://doi.org/10.23919/FMCAD.2018.8602994.

[3] M. Balakrishnan, D. Malkhi, J. D. Davis, V. Prabhakaran, M. Wei, and T. Wobber. CORFU: A distributed shared log. *ACM Trans. Comput. Syst.*, 31(4), 2013. URL https://doi.org/10.1145/2535930.

[4] A. Chlipala. Static checking of dynamically-varying security policies in database-backed applications. In *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010*. USENIX Association, 2010. URL http://www.usenix.org/events/osdi10/tech/full_papers/Chlipala.pdf.

[5] H. Cirstea, M. A. Kuppe, B. Loillier, and S. Merz. Validating traces of distributed programs against TLA+ specifications. *CoRR*, abs/2404.16075, 2024. URL https://doi.org/10.48550/arXiv.2404.16075.

[6] F. Darema. The SPMD model : Past, present and future. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 8th European PVM/MPI Users' Group Meeting*, volume 2131 of *Lecture Notes in Computer Science*. Springer, 2001. URL https://doi.org/10.1007/3-540-45417-9_1.

[7] S. Fouladi, R. S. Wahby, B. Shacklett, K. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*. USENIX Association, 2017. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi.

[8] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018*. USENIX Association, 2018. URL https://www.usenix.org/conference/nsdi18/presentation/geng.

[9] T. Gogacz and J. Marcinkowski. The hunt for a red spider: Conjunctive query determinacy is undecidable. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015*. IEEE Computer Society, 2015. URL https://doi.org/10.1109/LICS.2015.35.

[10] T. Gogacz and J. Marcinkowski. Red spider meets a rainworm: Conjunctive query finite determinacy is undecidable. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016*. ACM, 2016. URL https://doi.org/10.1145/2902251.2902288.

[11] H. Howard, M. A. Kuppe, E. Ashton, A. Chamayou, and N. Crooks. Smart casual verification of ccf's distributed consensus and consistency protocols. *CoRR*, abs/2406.17455, 2024. URL https://doi.org/10.48550/arXiv.2406.17455.

[12] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. Occupy the cloud: distributed computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017*. ACM, 2017. URL https://doi.org/10.1145/3127479.3128601.

[13] S. Kamil, A. Cheung, S. Itzhaky, and A. Solar-Lezama. Verified lifting of stencil computations. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016*. ACM, 2016. URL https://doi.org/10.1145/2908080.2908117.

[14] B. Krebs. USPS site exposed data on 60 million users, 2018. URL https://krebsonsecurity.com/2018/11/usps-site-exposed-data-on-60-million-users/. [Online; accessed November 18, 2024].

[15] S. Laddad, C. Power, M. Milano, A. Cheung, and J. M. Hellerstein. Katara: synthesizing crdts with verified lifting. *Proc. ACM Program. Lang.*, 6(OOPSLA2), 2022. URL https://doi.org/10.1145/3563336.

[16] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. J. DeWitt. Limiting disclosure in hippocratic databases. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004*. Morgan Kaufmann, 2004. URL http://www.vldb.org/conf/2004/RS3P3.PDF.

[17] N. Lehmann, R. Kunkel, J. Brown, J. Yang, N. Vazou, N. Polikarpova, D. Stefan, and R. Jhala. STORM: refinement types for secure web applications. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*. USENIX Association, 2021. URL https://www.usenix.org/conference/osdi21/presentation/lehmann.

[18] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, 1995. URL https://doi.org/10.1145/212433.220198.

[19] X. L. Li, A. Kuncoro, J. Hoffmann, C. de Masson d'Autume, P. Blunsom, and A. Nematzadeh. A systematic investigation of commonsense knowledge in large language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022*. Association for Computational Linguistics, 2022. URL https://doi.org/10.18653/v1/2022.emnlp-main.812.

[20] Y. Li, G. Kumar, H. Hariharan, H. M. G. Wassel, P. Hochschild, D. Platt, S. L. Sabato, M. Yu, N. Dukkipati, P. Chandra, and A. Vahdat. Sundial: Fault-tolerant clock synchronization for datacenters. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020*. USENIX Association, 2020. URL https://www.usenix.org/conference/osdi20/presentation/li-yuliang.

[21] B. Liskov. Practical uses of synchronized clocks in distributed systems. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*. ACM, 1991. URL https://doi.org/10.1145/112600.112601.

[22] A. Marzoev, L. T. Araújo, M. Schwarzkopf, S. Yagati, E. Kohler, R. T. Morris, M. F. Kaashoek, and S. Madden. Towards multiverse databases. In *Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS 2019*. ACM, 2019. URL https://doi.org/10.1145/3317550.3321425.

[23] M. Maunder. Vulnerability in WordPress Core: Bypass any password protected post. CVSS score: 7.5 (High), June 2016. URL https://www.wordfence.com/blog/2016/06/wordpress-core-vulnerability-bypass-password-protected-posts/. [Online; accessed November 18, 2024].

[24] A. Mehta, E. Elnikety, K. Harvey, D. Garg, and P. Druschel. Qapla: Policy compliance for database-backed systems. In *26th USENIX Security Symposium, USENIX Security 2017*. USENIX Association, 2017. URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/mehta.

[25] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 2004. URL https://doi.org/10.1145/1007568.1007633.

[26] A. Najafi and M. Wei. Graham: Synchronizing clocks by leveraging local clock properties. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022*. USENIX Association, 2022. URL https://www.usenix.org/conference/nsdi22/presentation/najafi.

[27] L. Nan, Y. Zhao, W. Zou, N. Ri, J. Tae, E. Zhang, A. Cohan, and D. Radev. Enhancing text-to-sql capabilities of large language models: A study on prompt design strategies. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*. Association for Computational Linguistics, 2023. URL https://doi.org/10.18653/v1/2023.findings-emnlp.996.

[28] A. Nash, L. Segoufin, and V. Vianu. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.*, 35(3), 2010. URL https://doi.org/10.1145/1806907.1806913.

[29] G. Neiger and S. Toueg. Simulating synchronized clocks and common knowledge in distributed systems. *J. ACM*, 40(2), 1993. URL https://doi.org/10.1145/151261.151267.

[30] O. Padon. *Deductive Verification of Distributed Protocols in First-Order Logic*. PhD thesis, Tel Aviv University, Israel, 2018. URL https://tau.primo.exlibrisgroup.com/permalink/972TAU_INST/bai57q/alma9932991300004146.

[31] P. Reisner. Human factors studies of database query languages: A survey and assessment. *ACM Comput. Surv.*, 13(1), 1981. URL https://doi.org/10.1145/356835.356837.

[32] C. Scott, A. Panda, V. Brajkovic, G. C. Necula, A. Krishnamurthy, and S. Shenker. Minimizing faulty execu-

tions of distributed systems. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016*. USENIX Association, 2016. URL https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/scott.

[33] E. Slaughter, W. Lee, S. Treichler, M. Bauer, and A. Aiken. Regent: a high-productivity programming language for HPC with logical regions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015*. ACM, 2015. URL https://doi.org/10.1145/2807591.2807629.

[34] E. Slaughter, W. Lee, S. Treichler, W. Zhang, M. Bauer, G. M. Shipman, P. S. McCormick, and A. Aiken. Control replication: compiling implicit parallelism to efficient SPMD with logical regions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017*. ACM, 2017. URL https://doi.org/10.1145/3126908.3126949.

[35] A. Torres, R. Galante, M. S. Pimenta, and A. J. B. Martins. Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Inf. Softw. Technol.*, 82, 2017. URL https://doi.org/10.1016/j.infsof.2016.09.009.

[36] J. Yang, T. Hance, T. H. Austin, A. Solar-Lezama, C. Flanagan, and S. Chong. Precise, dynamic information flow for database-backed applications. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016*. ACM, 2016. URL https://doi.org/10.1145/2908080.2908098.

[37] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports. Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015*. ACM, 2015. URL https://doi.org/10.1145/2815400.2815404.

[38] W. Zhang, V. Fang, A. Panda, and S. Shenker. Kappa: a programming framework for serverless computing. In *SoCC '20: ACM Symposium on Cloud Computing*. ACM, 2020. URL https://doi.org/10.1145/3419111.3421277.

[39] W. Zhang, S. Shenker, and I. Zhang. Persistent state machines for recoverable in-memory storage systems with NVRam. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020*. USENIX Association, 2020. URL https://www.usenix.org/conference/osdi20/presentation/zhang-wen.

[40] W. Zhang, E. Sheng, M. A. Chang, A. Panda, M. Sagiv, and S. Shenker. Blockaid: Data access policy enforcement for web applications. In *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022*. USENIX Association, 2022. URL https://www.usenix.org/conference/osdi22/presentation/zhang.

[41] W. Zhang, A. Panda, and S. Shenker. Access control for database applications: Beyond policy enforcement. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems, HOTOS 2023*. ACM, 2023. URL https://doi.org/10.1145/3593856.3595905.

[42] W. Zhang, D. Bali, J. Kerney, A. Panda, and S. Shenker. Extracting database access-control policies from web applications. *CoRR*, abs/2411.11380, 2024. URL https://arxiv.org/abs/2411.11380.

[43] W. Zhang, A. Panda, M. Sagiv, and S. Shenker. A decidable case of query determinacy: Project-select views. *CoRR*, abs/2411.08874, 2024. URL https://arxiv.org/abs/2411.08874.

[44] Z. Zorz. OpenEMR vulnerabilities put patients' info, medical records at risk, 2018. URL https://www.helpnetsecurity.com/2018/08/08/openemr-vulnerabilities/. [Online; accessed November 18, 2024].