Don't Even Ask: Database Access Control through Query Control

Richard Shay Uri Blumenthal Vijay Gadepally
Ariel Hamlin John Darby Mitchell Robert K. Cunningham
MIT Lincoln Laboratory, Lexington, MA USA
{richard.shay, uri, vijayg, ariel.hamlin, mitchelljd, rkc} @ ll.mit.edu

ABSTRACT

This paper presents a vision and description for query control, which is a paradigm for database access control. In this model, individual queries are examined before being executed and are either allowed or denied by a pre-defined policy. Traditional view-based database access control requires the enforcer to view the query, the records, or both. That may present difficulty when the enforcer is not allowed to view database contents or the query itself. This discussion of query control arises from our experience with privacy-preserving encrypted databases, in which no single entity learns both the query and the database contents. Query control is also a good fit for enforcing rules and regulations that are not well-addressed by view-based access control. With the rise of federated database management systems, we believe that new approaches to access control will be increasingly important.

1. INTRODUCTION

There are great opportunities associated with large-scale data collection, but also associated risks. Increasing privacy concerns about data collection are demonstrated by the recent European Union General Data Protection Regulation (GDPR) [6]. There is a need for greater privacy protections in securing, storing, and transmitting big data [22]. In this paper, we advance the concept of *query control*, an expressive database access control strategy.

Commonly used view-based access control restricts a user's *view* of the database. Query control is an alternative, complementary database access control strategy based on examining what queries a user is submitting. Each querier is assigned a *query control policy*; that querier's queries can only execute if they conform to the policy. Query control limits the questions being asked, rather than directly limiting the data being returned. Query control may be especially useful in enforcing policies that limit the questions that are allowed to be asked of a database, or limiting how data sets are utilized [32].

The use case that first motivated the development of query control is access control for privacy-preserving databases, in which encrypted queries are executed against encrypted data and encrypted results are returned to the querier [9, 14]. In addition to protecting data privacy, such databases protect the privacy of queries submitted and results returned from both the data owner and the entity processing the queries. As we will elaborate upon in Section 3, query control allows a third party to enforce access control on encrypted queries for an encrypted database, while preserving data privacy for both. Another promising use case of query control is in management systems supporting multiple heterogenous, federated databases. Such systems may need the ability to execute a single query across multiple individual database engines, each with its own access control and data storage approach [31]. Query control will enable database-agnostic access control policies to effect centralized, unified access control across diverse, composite database systems.

We present a brief overview of database access control, highlighting where query control fits, in Section 2. We present the background and history of query control, with a focus on privacy-preserving databases, in Section 3. Section 4 discusses salient use cases for query control. In Section 5, we present a high-level overview of our reference implementation for query control; while the focus of this paper is highlighting the case for using it, this reference implementation demonstrates its feasibility. Finally, we present promising future applications of query control in Section 6 and conclude in Section 7.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Air Force.

2. DATABASE ACCESS CONTROL

In order to understand the function of query control in database access control, it is useful to begin with a larger picture of database access control and then zoom in. Access control in databases has played an integral role in their development and popularity [7, 12]. Access control can be used at granularities such as the level of a table, individual rows, or even individual cells. Depending on the access control implementation, controlling which users can access what data entries can be a challenging task [3], sometimes amplified by application-specific requirements [4].

In general, *database access control* limits the access of a principal, a user or users, to the contents of a database. We separate the concepts of access control *strategies* and access control *mechanisms*. An access control *strategy* refers to how access control policies are assigned to principals, whereas an access control *mechanism* determines how access to the database is restricted. View-based access control, the most common access control mechanism found in production database systems, is data-dependent and is often implemented through metadata. *Query control* places a restriction on the queries that a principal can issue, and is therefore not data-dependent.

Access control strategies are orthogonal to access control mechanisms. For example, role-based strategies can be applied to view-based or query-based mechanisms. For the most part, relational database management systems have concentrated on view-based mechanisms with varying access control strategies. Therefore, these terms are often conflated and role-based access control in many contexts implies a view-based access control mechanism with a role-based strategy. It should be noted that multiple access control mechanisms need not be mutually exclusive. On a single database, query control can be used to limit the queries that are actually executed, and view-based access control can be used to limit the results returned.

2.1 Traditional Database Access Control

In a view-based database access control model, a principal requests access to database contents. The system evaluates whether the principal is authorized to access the database contents by examining the access control policy. Often, an access control policy depends on the contents being accessed. The system issues a decision that either allows or denies access. View-based access control uses a database view as an abstraction mechanism for the data available to a particular principal [12].

There are a number of historical models for access control strategies applied to the view-based access control mechanism [1, 2]. Some early strategies, such as Discretionary Access Control and Mandatory Access Control [25], were often implemented via individual or group level access control. Role-based access control is a popular way to implement access control policies [24].

Query re-writing was initially explored for optimization [13]. Rizvi et al. discuss using it for access control, such as ensuring a given column has a specific value [23]. This changes a query to match a policy, rather than rejecting a non-conforming query like query control. Re-writing requires the enforcement mechanism to have direct query access, which may not work in the privacy preserving use case for which query control was created.

2.2 Query Control Policy Definition

We formally define *Query Control* as a protocol between four entities: a *data owner* who provides the database and policies; a *data host* who hosts the database; a *policy enforcer* who determines if a query is valid according to the given policy; and, finally, a *querier* who is the principal issuing queries. Often the data host is the same entity as the data owner or policy enforcer.

The querier has been assigned a policy to restrict the queries that it can issue. There may be multiple queriers, each with separate query control policies. Each query issued by the querier is evaluated by the policy and met with either *accept* or *reject*. The decision is *accept* if and only if the query is properly formed and is acceptable based on the applicable policy. Otherwise, the decision is *reject*.

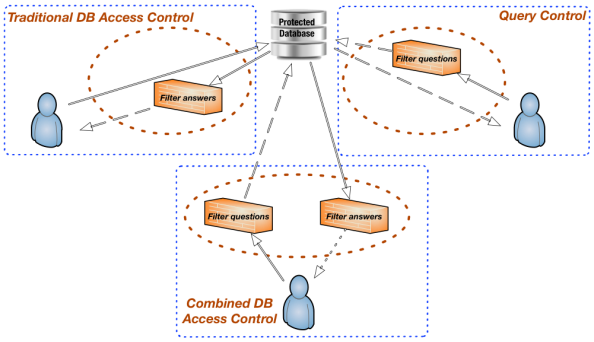

Figure 1: Placement of access control mechanisms on the database interaction path.

Figure 1 shows where the access control mechanisms can be applied. The traditional approach applies access control to the query result before being returned to the querier, filtering out what the

querier is not allowed to receive. This contrasts with query control, where the received query is evaluated against policy *before* it is executed by the database.

3. PRIVACY-PRESERVING DATABASES

The concept of query control came from IARPA research into privacy-preserving databases, which allow organizations to share data in a precisely controlled way [14]. Here, a privacy-preserving database means the data owner is the source of database records and wants assurance that any query adheres to a given policy. A querier submitting a query learns the details of any records that satisfy that query, but the data owner does not learn the contents or results of the query. Selecting query control as a model to preserve privacy enables a third party to enforce access control without learning about database contents and without the data owner learning about the contents of queries. Fuller et al. contains a more detailed explanation of the different design approaches and leakage tradeoffs of privacy-preserving databases [9].

This approach to a privacy-preserving database, with encrypted database and encrypted queries, constrains how access control can be implemented. Enabling a privacy budget requires that a single entity calculate on the distribution of data and also view queries. In this model, there is no such entity, and therefore a traditional privacy budget is not feasible. Further, this does not lend itself to changing permissions based on system load, as access control decisions may be made by an entity without any insight into the state of the database system.

The predecessor to SPAR, the Automatic Privacy Protection (APP) program, initially developed technology that included coarse query control [14]. Kagal used the AIR language for creating and enforcing permissions for semantic web technology [18], with language features such as restricting database columns [15]. Further work demonstrated that policy compliance can be enforced without being able to view database contents [27].

Following the success of APP, the SPAR program substantially increased the scope of research into privacy preserving databases. Performers designed their own mechanisms to express and enforce policies and integrated query control securely into query processing [14]. These query control mechanisms demonstrated a diverse range of capabilities but were found to be insufficient to express and enforce the variety of policy rules the government wanted. As a result, the query control policy language in Section 5 was developed under the subsequent Security and Privacy Assurance Research Software Evaluation (SPARSE) program. This program illustrates a real-world situation that called for query control, and provided an opportunity to create a reference implementation for a query control policy language.

4. QUERY CONTROL USE CASES

Query control can complement traditional view-based access control by filling in gaps in that access control strategy. Query control can replicate some types of control found in the view-based access control strategy – specifically column-based portions of view-based access control. Further, there are a number of restrictions that can be placed on queries issued by the querier that would not be easily imposed by view-based access control. These mechanisms are not mutually exclusive and can be used in tandem.

Query control policies can utilize any number of conjunctions (AND, OR, NOT) and can therefore express and enforce rules that contain conditionals. This means query control is well-suited for some types of natural-language database access control policies. Consider a policy that requires a query to ask only about a particular doctor's patients, unless that query also restricts itself to patients with a particular medical condition. This is easy to express in English, but not easily expressed using a database view. Because there is a conditional in this policy, no single static view of the database suffices to represent it. However, this policy can easily be expressed as a set of atomic rules combined with conjunctions (using the language to be presented in Section 5): (doctor_last_name == "Tyre") or (med_condition is included).

Query control can be enforced without needing access to the underlying database contents. Both approaches require access to the database schema, but view-based access control also requires access to the database contents. With query control, the results of the evaluation do not change if the database contents change. Further, defects in the data do not impact the query control decision.

Query control also lends itself more naturally to some types of time-based policies that might mirror natural language policy text. Kagal points out that a policy permitting access only to records on individuals who are at least 18 years of age can be difficult to implement using view-based access control [16]. Using query control, this becomes trivial: birthday at least 18 years ago. More complex policy examples will be presented in Section 5.1.

5. REFERENCE IMPLEMENTATION

In this section, we briefly describe a language we developed for query control, called Query Control Policy Language (QCPL). This is a preliminary sketch to demonstrate the feasibility of a query control language, and not a complete language specification. QCPL facilitates the specification of *query control policies*, which are comprised of one or more query control rules. A query control rule is made up of any number of *atomic* query control rules, combined with conjunctions. These conjunctions (AND, OR, NOT) combining atomic query control rules enable expressive and complex query control policies.

A query control rule specifies a requirement for a query. For example, the rule Count = 3 specifies that queries may only be accepted if they require that database column *Count* have the value *3*. Query `SELECT * FROM table WHERE ((Count = 3))` satisfies this rule, but `SELECT * FROM table WHERE ((Count > 1))` does not. This rule is satisfied by the following query statement because both clauses satisfy the rule: `(Count = 3 AND A = 1) OR (Count = 3 AND A = 2)` However, this rule is not satisfied by the following because its second clause does not require that `Count` be 3, and therefore it does not ensure that *Count* have a value of *3*: `(Count = 3 AND A = 1) OR (Count = 2)`.

In order to demonstrate its feasibility, we implemented the language in 1,486 lines of Ruby.[1] We evaluated a set of 1203 queries, with a mean of 7.3 operations per query, across ten policies. In the interest of space, we describe only a few of these policies. P1 limits searching on low-cardinality fields; P3 ensures queries are within a particular timeframe; and P10 ensures that searches are limited to one event within a particular timeframe. It took only 18 seconds to evaluate all 1203 queries against all ten policies, of which 16.1 seconds were used to parse the queries. Figure 2 depicts the evaluation time of all queries by each policy.

5.1 Query Control Policy Example

Consider query control on a hypothetical database of hospital patients. A policy that only allows queries on patients who are at least 18 years old unless they were admitted in the past week would be non-trivial to implement via view-based access control. It is simple to express using QCPL, combining atomic rules via conjunctions: (birthday at least 18 years ago) OR (admit_date at most 7 days ago). Consider a a policy that requires that any query with a patient name must also include a patient birthday. This can be created by combining not and or operators: (not (patient_name is included)) OR (doctor_name is included) Further conjunctions are likewise simple to add to this policy. This illustrates how some access control policies that would be complex to implement in view-based access control can be easily expressed in QCPL.

[1] `ruby 2.0.0p648`

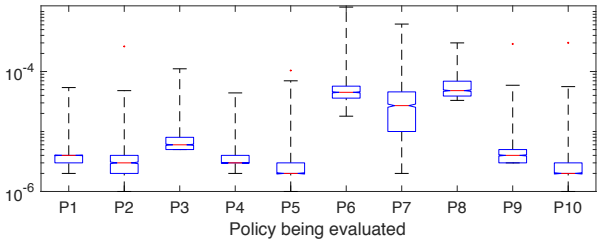

Figure 2: Evaluation time of 10 different query policies on 1203 SQL queries. Mean time is indicated by the red line. Box edges denote 25th and 75th percentiles. The Y-axis depicts time in seconds in log scale.

6. PROMISING FUTURE APPLICATIONS

6.1 Securing Consumer Data

Query control has promising applications in developing secure data-sharing solutions. One such use-case is securing consumer data. The big data phenomenon has resulted in numerous organizations collecting, storing, and processing large quantities of sensitive data. Once collected, data can be shared between organizations and within an organization – such as WhatsApp sharing user data with Facebook for advertisements [20]. In a context such as mobile devices, privacy concerns arise from users being unable to know who has their data and how their data are being used [19, 28]. If an organization is planning to share its user data with another organization, query control might be used to restrict how the second organization can access customer data.

6.2 Federated Database Systems

Developing heterogenous database management systems [31] may also benefit from an access control mechanism based on query control. These systems are built with the notion that future database systems may need to support multiple database "sizes" that are tuned to the underlying data they are storing or processing [29]. These systems are often characterized by support for heterogenous database management systems and multiple query and/or processing engines [26].

An example of such a system is the BigDAWG polystore system [10, 21]. Its current version [11] supports data querying of data stored in Apache Ac-

cumulo [17], a distributed key-value store database; PostGRES [30], a relational database; and SciDB [5], an array database. Each of these composite systems has its own access control mechanisms and strategies. Currently, developing access control for such systems may require using the "greatest common factor" of access-control-mechanism granularity across the disparate systems. As new systems are added, this challenge is compounded. Further, view-based access control heavily depends on the data being stored. Modern systems, such as BigDAWG, routinely copy data from one system to another in order to execute a query. Such challenges call for a new way of thinking about access control in database systems.

We believe that a *query-based* access control strategy can be readily applied to such systems, creating a centralized and abstract access control mechanism. Its evaluation need not depend on the database engine of any one system. Query control can be applied to any system with data stored in a predefined schema, which almost all database systems support. Thus, in the example system presented above, access control could be evaluated on the query directly and would not rely on the access control of underlying systems.

6.3 Usability Research

Query control has been initially studied through pilot testing under SPAR [8]. Further work is needed for validation of both query control as a concept and the QCPL language in particular. User studies can continue examination of how a user's experience is impacted if his or her queries are rejected. Further studies can examine how well data owners are able to express their ideal access control policies via QCPL, leading to improvements in the language.

Future work may also focus on how much a querier learns about a query control policy in a protected database. Attempting to obscure the query control policy from the querier leads to a number of interesting questions. If a querier is allowed an unlimited number of queries, he or she may discover that executing similar queries with minor changes allows discernment of part or all of the query control policy. Likewise, the querier may be able to compare the timing of queries that do and do not return results to learn about the query control policy.

7. CONCLUSION

In this paper, we have discussed query control and how it fits into the larger context of database access control. We have highlighted the past, present, and future of query control – how it was created to meet the needs of privacy-preserving databases, how organizations might benefit from it today, and how future use-cases might benefit from it even more. We advocate for further research into the space of query control, and for further usability studies. We believe that query control will become increasingly important as more data is accumulated, databases are increasingly federated, and searchable encryption becomes increasingly popular.

8. ACKNOWLEDGMENTS

The authors thank the many participants and program managers on IARPA-funded projects for interesting discussions and helpful feedback that improved this work.

9. REFERENCES

[1] E. Bertino and J. Crampton. Security for distributed systems: Foundations of access control. *Information Assurance: Survivability and Security in Networked Systems*, 2008.

[2] E. Bertino, G. Ghinita, and A. Kamra. Access control for databases: Concepts and systems. In *Foundations and Trends in Databases*, 2010.

[3] E. Bertino and R. Sandhu. Database security-concepts, approaches, and challenges. *IEEE Transactions on Dependable and secure computing*, 2005.

[4] M. A. Brookhart, T. Stürmer, R. J. Glynn, J. Rassen, and S. Schneeweiss. Confounding control in healthcare database research: challenges and potential approaches. *Medical care*, 2010.

[5] P. G. Brown. Overview of SciDB: large scale array storage, processing and analysis. In *SIGMOD International Conference on Management of data*, 2010.

[6] European Commission. 2018 reform of EU data protection rules. `https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en`.

[7] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *TISSEC*, 2001.

[8] B. Fuller, D. Mitchell, R. Cunningham, U. Blumenthal, P. Cable, A. Hamlin, L. Milechin, M. Rabe, N. Schear, R. Shay, M. Varia, S. Yakoubov, and A. Yerukhimovich. SPAR pilot evaluation.

Technical report, MIT Lincoln Laboratory, 2015.

[9] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. Gadepally, R. Shay, J. D. Mitchell, and R. K. Cunningham. SoK: Cryptographically protected database search. *Oakland*, 2017.

[10] V. Gadepally, P. Chen, J. Duggan, A. Elmore, B. Haynes, J. Kepner, S. Madden, T. Mattson, and M. Stonebraker. The bigdawg polystore system and architecture. In *HPEC*, 2016.

[11] V. Gadepally, K. O'Brien, A. Dziedzic, A. Elmore, J. Kepner, S. Madden, T. Mattson, J. Rogers, Z. She, and M. Stonebraker. Bigdawg version 0.1. In *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*, pages 1–7. IEEE, 2017.

[12] P. P. Griffiths and B. W. Wade. An authorization mechanism for a relational database system. *Transactions on Database Systems (TODS)*, 1976.

[13] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 2001.

[14] Intelligence Advanced Research Projects Activity. Security and privacy assurance research (SPAR) program broad agency announcement, 2011.

[15] L. Kagal. Policy compliance of queries for private information retrieval. IARPA BAA Appendix E, August 2010.

[16] L. Kagal. Policy compliance of queries for private information retrieval. `http://dig.csail.mit.edu/2009/IARPA-PIR/`, August 2011.

[17] J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, P. Michaleas, J. Mullen, A. Prout, et al. Achieving 100,000,000 database inserts per second using accumulo and d4m. In *HPEC*, 2014.

[18] A. Khandelwal, J. Bao, L. Kagal, I. Jacobi, L. Ding, and J. Hendler. Analyzing the air language: A semantic web (production) rule language. In *International Conference on Web Reasoning and Rule Systems*, 2010.

[19] P. G. Leon, B. Ur, Y. Wang, M. Sleeper, R. Balebako, R. Shay, L. Bauer, M. Christodorescu, and L. F. Cranor. What matters to users? Factors that affect users' willingness to share information with online advertisers. In *SOUPS*, 2013.

[20] N. Lomas. WhatsApp to share user data with facebook for ad targeting – here's how to opt out. `https://techcrunch.com/2016/08/25/whatsapp-to-share`, August 2016.

[21] T. Mattson, V. Gadepally, Z. She, A. Dziedzic, and J. Parkhurst. Demonstrating the bigdawg polystore system for ocean metagenomics analysis. In *CIDR*, 2017.

[22] President's Council of Advisors on Science and Technology. Big data and privacy: A technical perspective. Technical report, Executive Office of the President, `https://bigdatawg.nist.gov/pdf/pcast_big_data_and_privacy_-_may_2014.pdf`, May 2014.

[23] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD International Conference on Management of data*, 2004.

[24] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[25] R. S. Sandhu and P. Samarati. Access control: principle and practice. *IEEE Communications*, 1994.

[26] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys (CSUR)*, 1990.

[27] J. H. Soltren. Query-based database policy assurance using semantic web technologies. Master's thesis, MIT, September 2009.

[28] C. Spensky, J. Stewart, A. Yerukhimovich, R. Shay, A. Trachtenberg, R. Housley, and R. K. Cunningham. SoK: Privacy on mobile devices – it's complicated. In *Proceedings on Privacy Enhancing Technologies*, 2016.

[29] M. Stonebraker and U. Cetintemel. One size fits all: An idea whose time has come and gone. In *ICDE*, 2005.

[30] M. Stonebraker and L. A. Rowe. *The design of Postgres*, volume 15. ACM, 1986.

[31] R. Tan, R. Chirkova, V. Gadepally, and T. G. Mattson. Enabling query processing across heterogeneous data models: A survey. In *2017 IEEE International Conference on Big Data*, 2017.

[32] P. Upadhyaya, N. R. Anderson, M. Balazinska, B. Howe, R. Kaushik, R. Ramamurthy, and D. Suciu. Stop that query! The need for managing data use. In *CIDR*, 2013.