Policy Enforcement Framework for Cloud Data Management

Kevin W. Hamlen*, Lalana Kagal†, Murat Kantarcioglu*

*University of Texas at Dallas, †Massachusetts Institute of Technology

Abstract

Cloud computing is a major emerging technology that is significantly changing industrial computing paradigms and business practices. However, security and privacy concerns have arisen as obstacles to widespread adoption of clouds by users. While much cloud security research focuses on enforcing standard access control policies typical of centralized systems, such policies often prove inadequate for the highly distributed, heterogeneous, data-diverse, and dynamic computing environment of clouds. To adequately pave the way for robust, secure cloud computing, future cloud infrastructures must consider richer, semantics-aware policies; more flexible, distributed enforcement strategies; and feedback mechanisms that provide evidence of enforcement to the users whose data integrity and confidentiality is at stake. In this paper, we propose a framework that supports such policies, including rule- and context-based access control and privacy preservation, through the use of in-lined reference monitors and a trusted application programming interface that affords enforceable policy management over heterogeneous cloud data.

1 Introduction

Cloud computing security has rapidly emerged as a significant concern for businesses and end-users over the past few years. For example, in a 2010 survey, Fujitsu concluded that 88% of its customers have significant concerns about data integrity and privacy in the cloud [9]. While some cloud security issues are addressable via traditional techniques that have been used for decades to secure centralized, time-shared systems, others are endemic to the uniquely diverse and dynamic nature of cloud environments, and therefore demand new solutions [6].

Our prior research has identified at least three major categories of security challenges that are impeding information assurance in clouds: (1) semantic diversity of cloud data, (2) customer-cloud negotiated mobile computations, and (3) multi-party, cross-domain security, privacy and accountability policies.

Semantic diversity of data in clouds arises from the vast range of different datasets and data processing/querying tools that production-level clouds must process. These different datasets range from structured to unstructured data and data processing/querying frameworks range from MapReduce [7] (e.g., Hadoop [2]) to stream processing (e.g., [26]). This emerges as a security challenge because of the need to formulate policy languages that are sufficiently general to capture and relate permissible uses of security-relevant data with diverse semantics. For example, fine grained access control policies defined for streaming data applications could be vastly different from the applications that try to support SQL-like queries on relational data.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Mobile computations for clouds differ from traditional time-shared systems in that cloud infrastructures are seldom static or fully transparent to users. Further more, different cloud providers provide different internal network topology, storage model, job scheduling mechanisms, etc. These infrastructure choices often have security-relevant implications for users. For example, different versions of Hadoop may have different access control mechanisms that support different levels of granularity. In order to enforce their policies efficiently, users must be afforded a flexible range of enforcement options that allow them to choose the best enforcement strategy for each job given the (possibly evolving) constraints of the computing environment. Dually, the security adequacy of each option should be independently verifiable by the cloud provider in order to protect the cloud infrastructure and its other users.

Finally, the potential power of cloud computing stems in part from its ability to synergistically co-mingle large datasets from multiple organizations, and process distributed queries over the combined data for long periods of time. Policies for such an environment must be multi-party and history-based. For example, we may need to support contract-style data-sharing policies where organization X is willing to share data set D with organization Y if organization Y is willing to share data set D' with organization X.

In what follows, we describe how our prior work on a general cloud policy enforcement frameworks offers new, promising approaches for surmounting these challenges, and we recommend future work that will allow practical application of these technologies to clouds.

2 Proposed Approach

A general policy enforcement framework for cloud data management[1] must consider three important dimensions: (1) data type (e.g., relational data, RDF data, text data, etc.), (2) computation (e.g., SQL queries, SPARQL, Map/Reduce tasks, etc.), and (3) policy requirements (e.g., access control policies, data sharing policies, privacy policies, etc.) Given the wide range of available choices in each dimension, a policy enforcement framework must be highly flexible and must support different data processing requirements. To achieve these goals, we propose a policy-compliant data processing framework with the following modules:

- **Policy-reasoning module:** The main job of the policy-reasoning module is to map a policy to a specific set of tasks that will be executed on the data along with the submitted data processing task to enforce various policies. Based on the policy reasoning results, the policy reasoning module will output initial data pre-processing tasks, a modified data processing task, and the post-processing tasks for each submitted data processing task.

- **Data processing task rewriting module:** In some cases, using different pre-processing and post-processing tasks might result in different computational costs. In addition, to enable efficient computation, some of the pre-processing and post-processing tasks might need to be combined or simplified without provoking a policy violation. The data processing rewriting module can consider various rewriting strategies for enforcing policies.

- **Pre-processing module:** The pre-processing module executes the pre-processing tasks on the underlying data. For example, a pre-processing step might require the dataset to be anonymized with some privacy definition (e.g., l-diversity, t-closeness, k-anonymity, etc.) before any query is executed on it. In addition, pre-processing tasks could be used to filter sensitive data. For example, fine grained access control on relational data could be enforced by using a pre-processing task to create a view that can be given to each user-submitted SQL query.

- **Post-processing module:** In some cases, it might be more efficient to process the final results to enforce policies. For example, an accountability policy might require the creation of certain audit

[1]We here assume that the cloud system is trusted with enforcing given policies. Untrusted clouds are left to future work.

logs if the data processing task changed certain data records. Such policies could be enforced using post-processing tasks.

To efficiently implement the above modules, we need to able to specialize them for different data types, computation types and policy requirements. One way to achieve this goal is to create specialized modules for common data types, computations, and policies. For example, in our previous work [28], we built the above modules for enforcing role-based access control policies for a relational data store on the cloud that is processed using SQL queries. In essence, we have combined the Hadoop Distributed File System [27] with Hive [29] to provide a common storage area for storing large amounts of relational data and for running SQL like queries. Further, we have used an XACML [23] policy-based security mechanism to provide fine-grained access controls over the stored data. In this case, the policy reasoning module uses an XACML reasoning engine to check whether a user who submitted an SQL query has access to all the underlying table/views. The pre-processing module runs HiveQL queries to create materialized views that are accessed by user submitted queries. A query rewriting module can modify the submitted query based on the underlying view definitions for efficient query processing. In this case, post-processing may not be needed since pre-processing and/or query rewriting may be enough for enforcing basic access control policies.

Of course, such a specialized approached will not necessarily be applicable to other types of data, computations, and policies. For example, if the stored data is unstructured and computations executed on the data are arbitrary MapReduce jobs, then we need different policy enforcement techniques. In such scenarios, user submitted MapReduce jobs could be modified using in-lined reference monitoring techniques (see Section 2.2 for more details.).

2.1 Data-aware Policy Languages for Clouds

As mentioned previously, a policy enforcement framework for cloud must support a wide range of policies. In this section, we briefly summarize policies and policy languages previously proposed in the literature that are potentially enforceable by our framework.

Access controls are one of the most important policy classes that must be considered. At a general level, access control languages make assertions about users and their permissible operations [24]. Additionally, languages have been defined for making authorization decisions for policies that have been combined from various sources [3], as well as for supporting trust management (e.g., [32]). Furthermore, since the advent of XML and its acceptance as a *de facto* standard for information exchange, a number of access control languages based on XML have been proposed. An important example of XML-based access control is XACML[2], which provides an XML syntax for defining policies that constrain resources that may also be expressed in XML.

There have also been access control languages that are designed in a logic-based framework (e.g., [20]). The additional expressive power and formal, verifiable methodology offered by logic-based languages are particularly useful in the context of access control. Finally, access control languages have also been defined in the context of Semantic Web languages (e.g., Rei [16] and KAoS [30]). Semantic Web languages are based on description logics, which are a decidable subset of first order logic, and hence provide benefits that are similar to logic-based languages.

Group-Centric Secure Information Sharing (g-SIS) (e.g., [18]) is an example of a family of access control models that is tailored to suit the requirements of information sharing on the cloud. The family of models in g-SIS is based on the notion of brining users and resources together into a group for purposes of information sharing. In other words, this means that users and resources must be present in the system simultaneously for the users to be able to access the resources. In addition, the family of g-SIS models is based on the following two principles:

[2]http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-en.html

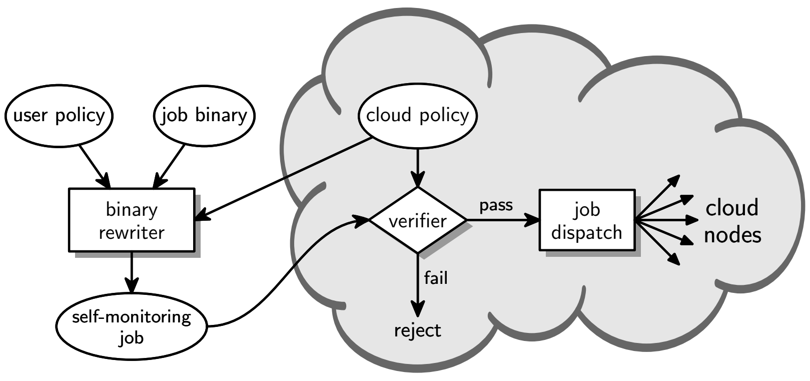

Figure 1: A cloud framework with policy-enforcement based on certifying IRMs

- **Share but differentiate:** The sharing aspect is achieved through users joining a group and then adding information to that group. However, the access that a user is granted to resources is based on several factors—for example, the time at which a user is added to the group relative to the time at which the information was posted to the group, and other user-configurable attributes.

- **Multiple groups:** This principle refers to the notion that various types of relationships can hold between different groups (e.g., hierarchical, mutual exclusion, etc.) with a possibly overlapping set of users.

Research devoted to the g-SIS family of models has included the development of a formal model for g-SIS using linear temporal logic (LTL) [21], the specification of core properties that g-SIS models must satisfy, as well as extensions that show how g-SIS models allow secure and agile information sharing when compared with traditional access control techniques, and finally, the development of a "stateful" enforceable specification of g-SIS [19].

An alternative approach to provenance-aware access control is to tailor the language to suit the requirements (e.g., policy enforcement based on an aggregation of applicable policies, redaction policies, etc.) [22]. This proposed language uses an XML syntax and specifies a variety of tags (viz., target, effect, condition, and obligations) to capture various access control use-cases that arise in the domain of provenance. However, the language is not able to capture resources with arbitrary path lengths that occur within a provenance graph. Therefore, a resource to be protected must be identified *a priori*, rather than being passed as a parameter at runtime. The task of identifying resources *a priori* might be infeasible, since there might be an exponential number of resources in a provenance graph.

Subsequent work has addresses this path-length drawback through the use of regular expressions to define resources requiring protection [4]. The use of regular expressions allows resources with arbitrary path lengths to be defined and used at runtime rather than having to create resources *a priori*. The same authors have also extended the notion of redaction to provenance graphs [5]. They use a graph grammar approach [8] that rewrites a high-level specification of a redaction policy into a graph grammar rule that transforms the original provenance graph into a redacted provenance graph.

2.2 Flexible Cloud Policy Enforcement

One way to provide cloud customers maximum flexibility with regard to policy enforcement is to permit the enforcement mechanism to reside within the jobs, with suitable checking on the cloud side to ensure that the job's self-enforcement is adequate. Such a cloud framework is illustrated in Figure 1. For example, a job expressed as a Java bytecode binary (as in typical Hadoop MapReduce clouds) can self-enforce an access control policy by voluntarily restricting its accesses to policy-permitted resources. If the full policy is not known at code-generation time, the cloud can even provide a trusted system API that job code may consult at runtime to discover policies to which it is subject and self-censor its resource accesses accordingly.

As a simple illustration of how such self-enforcement can be more efficient than cloud-implemented, system-level enforcement, consider a job J that counts database records $r \in D$ satisfying some predicate $P_J(r)$, and consider a policy C that prohibits J from accessing records that falsify a policy-prescribed predicate $P_C(r)$. To enforce this policy, a system-level implementation might intercept all attempts by J to access records r, denying those for which $P_C(r)$ is falsified. Alternatively, job J could self-enforce this policy by implementing $P_C(r)$ within its own code, but only evaluating it on records r that have already satisfied $P_J(r)$. In both cases, job J satisfies the policy and counts the set of records that satisfy conjunction $P_C(r) \wedge P_J(r)$. However, if P_C is more computationally expensive than P_J and few records satisfy P_J, the self-enforcement approach could be far more efficient.

Self-enforcement need not impose an implementation burden on clients if the inclusion of the enforcement mechanism into the job code can be automated. Our prior work has demonstrated that enforcement of safety policies—including access control policies—can be in-lined into arbitrary, untrusted, Java binaries fully automatically through the use of *aspect-oriented in-lined reference monitors* (IRMs) [10, 14, 15]. IRMs in-line the logic of traditionally system-level reference monitors into untrusted code to produce policy-compliant, self-monitoring code [25]. The in-lining process identifies potentially security-relevant instructions in the untrusted code and surrounds them with guard code that preemptively detects impending policy violations at runtime. When an impending violation is detected, the guard code intervenes by halting the job prematurely or taking some other corrective action (e.g., throwing a catchable exception or rolling back to a consistent state).

For example, a simple binary rewriter might replace all instructions $\mathit{read}()$, which read a new database record, with a wrapper function of the form

$$\big(\textbf{let}\ r = \mathit{read}()\ \textbf{in}\ (\textbf{if}\ P_C(r)\ \textbf{then}\ r\ \textbf{else}\ \mathit{error})\big) \tag{2}$$

The result of such a transformation is code that self-enforces policy C. More sophisticated rewriters can in-line such guard code more intelligently, such as by shifting the check to time-of-use sites instead of time-of-read sites when doing so improves performance, or by distributing the implementation of P_C across multiple nodes when P_C is computationally expensive (e.g., when r is large).

IRM implementations and the policies they enforce can be elegantly expressed using *aspect-oriented programming* (AOP) [31]. AOP has been used for over a decade to implement cross-cutting concerns in large source codes, and there is a rich family of production-level compilers and programming languages that support it. It extends traditional object-oriented programming with *aspects*, which consist of *pointcuts* and *advice*. Pointcuts are similar to regular expressions, but match sets of program operations instead of strings. The compiler or aspect-weaver for an AOP system in-lines the aspect-supplied advice code into the target program at every code point that matches the pointcut. In the context of IRMs, pointcuts can be leveraged to specify policy-relevant program operations (e.g., *read*), and advice can be leveraged to specify guard code for those operations (e.g., the computation in 2).

Thus, AOP and AOP-based specification languages constitute a powerful and well-developed paradigm for enforcing a broad class of security policies as IRMs. In such a framework, users and clouds specify policies using a declarative policy language, such as SPoX [10], from which an automated rewriter synthesizes appropriate aspects and weaves them into the target code at the binary level. The result is binary code that transparently self-enforces the policies without any manual intervention from the user.

The use of IRMs as a basis for enforcing policies mandated by the cloud or by other clients (e.g., owners of shared data accessed by untrusted, third-party jobs) is only feasible if the cloud can independently verify that submitted jobs satisfy the policies to which they are subject. While such code properties are in general undecidable [13], recently a series of technologies has emerged that permit a broad class of powerful IRM implementations to be verified fully automatically via type-checking [12], contract-based certification [1], or model-checking [11]. By implementing one of these algorithms, a trusted cloud can safely permit jobs to self-enforce mandatory access control policies, yet statically verify that this self-enforcement is sound prior to the job's deployment.

The cloud framework depicted in Figure 1 combines these ideas to implement a flexible policy enforcement strategy based on certifying IRMs. Jobs expressed as binary code are rewritten automatically on the client side in accordance with both user-specified (i.e., discretionary) and cloud-specified (i.e., mandatory) policies. The result of the rewriting is a new binary that self-enforces the desired policies. When the job is submitted to the cloud, the cloud first verifies that the submitted code has been instrumented with security checks sufficient to enforce the mandatory policies. Once it passes verification, the job can then be safely dispatched to the rest of the cloud without the need for additional system-level monitoring.

3 Conclusion

In this paper, we outlined a general policy enforcement framework needed for policy-compliant cloud data management. We discussed different policy types applicable for cloud data management ranging from data sharing policies to traditional access control policies, and showed how various techniques such as IRMs could be used to enforce such policies in a flexible, user-driven, but cloud-certified manner. Our proposals assumed that the cloud infrastructure is trusted to enforce or certify the enforcement of the specified policies. In our future work, we plan to explore how such policies could be enforced on semi-trusted and/or untrusted cloud infrastructures (cf., [17]).

References

[1] I. Aktug, M. Dam, and D. Gurov. Provably correct runtime monitoring. In J. Cuellar, T. Maibaum, and K. Sere, editors, *Proceedings of the 15th International Symposium on Formal Methods (FM)*, pages 262–277, 2008.

[2] Apache™ Hadoop®. http://hadoop.apache.org.

[3] P. A. Bonatti, S. D. C. di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and Systems Security (TISSEC)*, 5(1):1–35, 2002.

[4] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. M. Thuraisingham. A language for provenance access control. In *Proceedings of the 1st ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 133–144, 2011.

[5] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. M. Thuraisingham. Transforming provenance using redaction. In *Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 93–102, 2011.

[6] Y. Chen, V. Paxson, and R. H. Katz. What's new about cloud computing security? Technical Report UCB/EECS-2010-5, EE & CS Dept., U.C. Berkeley, 2010.

[7] J. Dean and S. Ghemawat. MapReduce: A flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[8] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, Berlin, 2006.

[9] Fujitsu. Personal data in the cloud: A global survey of consumer attitudes. Technical report, Fujitsu Research Institute, 2010.

[10] K. W. Hamlen and M. Jones. Aspect-oriented in-lined reference monitors. In Ú. Erlingsson and M. Pistoia, editors, *Proceedings of the 3rd ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, pages 11–20, 2008.

[11] K. W. Hamlen, M. M. Jones, and M. Sridhar. Aspect-oriented runtime monitor certification. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 126–140, 2012.

[12] K. W. Hamlen, G. Morrisett, and F. B. Schneider. Certified in-lined reference monitoring on .NET. In V. C. Sreedhar and S. Zdancewic, editors, *Proceedings of the 1st ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS)*, pages 7–16, 2006.

[13] K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(1):175–205, 2006.

[14] M. Jones and K. W. Hamlen. Enforcing IRM security policies: Two case studies. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 214–216, 2009.

[15] M. Jones and K. W. Hamlen. Disambiguating aspect-oriented security policies. In J.-M. Jézéquel and M. Südholt, editors, *Proceedings of the 9th International Conference on Aspect-Oriented Software Development (AOSD)*, pages 193–204, 2010.

[16] L. Kagal, T. W. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 63–74, 2003.

[17] S. M. Khan and K. W. Hamlen. Hatman: Intra-cloud trust management for Hadoop. In *Proceedings of the 5th IEEE International Conference on Cloud Computing (CLOUD)*, pages 494–501, June 2012.

[18] R. Krishnan and R. S. Sandhu. A hybrid enforcement model for group-centric secure information sharing. In *Proceedings of the International Conference on Computational Science and Engineering (CSE)*, volume 3, pages 189–194, 2009.

[19] R. Krishnan and R. S. Sandhu. Authorization policy specification and enforcement for group-centric secure information sharing. In *Proceedings of the 7th International Conference on Information Systems Security (ICISS)*, pages 102–115, 2011.

[20] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 114–130, 2002.

[21] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[22] Q. Ni, S. Xu, E. Bertino, R. S. Sandhu, and W. Han. An access control language for a general provenance model. In *Proceedings of the 6th VLDB Workshop on Secure Data Management (SDM)*, pages 68–88, 2009.

[23] OASIS XACML Technical Committee (B. Parducci and H. Lockhart, chairs; E. Rissanen, editor). eXtensible Access Control Markup Language (XACML) version 3.0. Oasis, 2010.

[24] P. Samarati and S. D. C. di Vimercati. Access control: Policies, models, and mechanisms. In *Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design: Tutorial Lectures*, pages 137–196, 2000.

[25] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[26] Storm: Distributed and fault-tolerant realtime computation. http://storm-project.net.

[27] K. Svachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed file system. In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, 2010.

[28] B. M. Thuraisingham, V. Khadilkar, A. Gupta, M. Kantarcioglu, and L. Khan. Secure data storage and retrieval in the cloud. In *Proceedings of the 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 1–8, 2010.

[29] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. In *Proceedings of the VLDB Endowment*, volume 2(2), pages 1626–1629, 2009.

[30] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *Proceedings of the 2nd International Semantic Web Conference*, pages 419–437, 2003.

[31] M. Wand, G. Kiczales, and C. Dutchyn. A semantics for advice and dynamic join points in aspect-oriented programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(5):890–910, 2004.

[32] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and Systems Security (TISSEC)*, 6(1):1–42, 2003.