Preserving Privacy Based on Semantic Policy Tools

Lalana Kagal* and Joe Pato+
*MIT Computer Science and Artificial Intelligence Lab
+HP Labs

Abstract—Private data of individuals is constantly being collected, analyzed, and stored by different kinds of organizations: shopping sites to provide better service and recommendations, hospitals for improved healthcare, and government agencies to enable national defense and law enforcement. Sharing data between these organizations makes it possible to discover important knowledge and draw useful conclusions but raises concerns about information privacy and trust. Until recently the focus was on restricting access to data on a "need to know" basis, but since the 9/11 Commission there has been a paradigm shift to "need to share". Our work explores the use of semantic privacy policies, justifications for data requests, and automated auditing to encourage sharing of sensitive data between organizations. We describe an architecture based on our policy tools that evaluates incoming queries against semantic policies and domain knowledge and provides a justification for each query - why they are permitted, denied, or inapplicable. Using a semantic policy language gives policies explicit semantics that allows all participants to unambiguously understand their meaning. The justifications generated by checking incoming requests against these policies help requesters formulate privacy-aware queries. Lastly, reasoning over event logs and justifications allows data owners to verify that their privacy policies are being correctly enforced.

Index Terms—Privacy policy, justifications, proofs, sharing, trust

1 Introduction

Efforts to address privacy have been dominated by strict access restriction and privacy-preserving algorithms such as anonymization, generalization, and perturbation. These techniques assume that most data consumers are malicious and focus on helping data owners restrict access to data. However, these techniques also discourage honest consumers from accessing data they require. With the current push towards need-to-share, we suggest that alternative approaches are required that help honest consumers be honest as well as enforce privacy policies of data owners.

Our group is exploring accountability mechanisms as a means of protecting central information policy values such as privacy and fair and reliable use of information. We view accountability combined with transparency and appropriate redress as a complementary process to strict access control [1]. Fundamental to this approach is the use of formalisms that can express realistic data-use policies, and automated reasoning engines that can interpret those policies and automatically determine whether particular uses of data are policy-compliant.

We are buildings systems that provide policy-awareness enabling honest users to comply with explicitly defined policies and data owners to verify that their policies are being enforced correctly. *Policy-awareness* is a property of information systems that provides participants with machine-readable representations of policies and audit trails in order to facilitate compliance with stated rules, as well as understandable views of policy decisions made.

Most access control mechanisms provide just a yes/no or permit/deny kind of answer leaving users unsure of what they are doing wrong and suspicious of possible prejudices. Users wanting to comply with security and privacy policies end up either being afraid to make queries for fear of doing something wrong or being angry that they are never able to get a query past the security mechanism to the data they require. Our objective is to use policy-awareness to enable users to make the right queries by giving them enough information to do so. Consider a simple privacy policy "You may not query for both zipcode and last name". With normal access control mechanisms, a requester who is asking for several fields including zipcode and

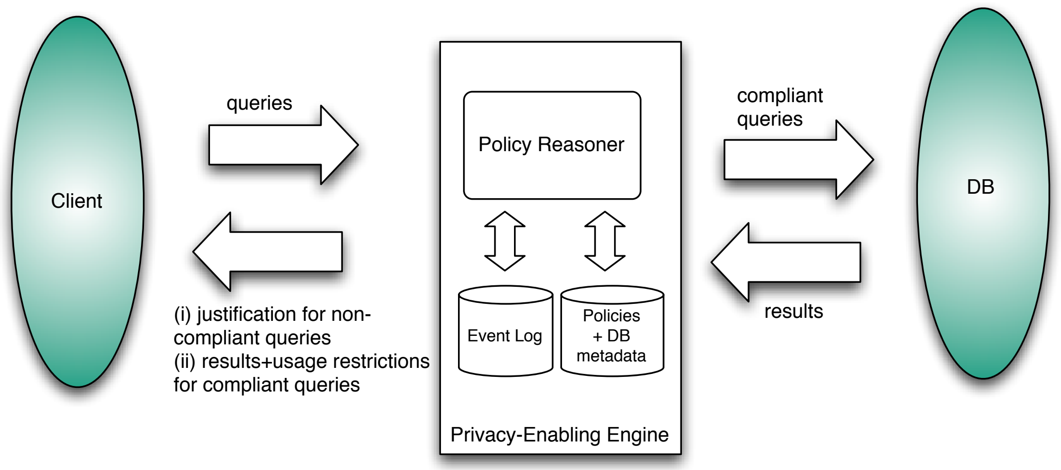


Fig. 1. Architecture: Databases have a policy layer that allows them to identify and execute only compliant queries and return justifications for non-compliant ones

last name from the database will be constantly rejected without knowing why. Similarly a requester who initially asked for zipcodes and is now asking for first and last names will be rejected. Our policy tools will provide a reason for this rejection such as "You've previously requested zipcode. Once one of zipcode and last name has been requested, the other cannot be retrieved". This justification enables the user to reformulate his query so that it does respect the privacy policies. Further, policies derived from regulatory controls governing information sharing are often complicated and may need to draw on contradictory sources from different jurisdictions [2], [3]. Examination of explicit justifications can expose these contradictions and lead to redefinition of the appropriate policy.

Policy-awareness supports extensive and machine-readable logs of events that have occurred in the system including requests for data and the results of those requests. We provide tools to allow data owners to run audits over these logs to check whether the enforcement mechanisms are fulfilling the intent of the policy. In case of non-conformance, the justifications associated with the decision provide guidance on how the policy should be adjusted to meet their intended result. For example, if the log shows that Alex accessed certain records the data owner or an authorized user can ask why this was permitted. Our tools provide a detailed justification saying that Alex is from the FBI, and his purpose was law enforcement, so it was permitted. If the data owner realizes that he does want FBI agents to access all data for law enforcement but only wants them to retrieve data about a "person of interest" he can add this rule to the policy and continue monitoring the logs.

We have designed an architecture based on policy-awareness and accountability that enables privacy-preserving sharing of sensitive data. We have implemented most of the key components including the Semantic Web based policy language, the reasoner that infers whether queries are compliant, generates justifications and handles private policies, the user interface that allows justifications to be explored, and techniques for integrating the architecture with different databases, as well as for the handling of intention, usage and context. In this article, we define our architecture, describe the individual components, and illustrate the effectiveness of our solution through several examples.

2 Architecture

As illustrated in Figure 1, our architecture includes inserting a privacy-enabling engine in front of the database containing private data. This engine consists of (i) policy reasoner, (ii) event log, (iii) set of applicable policies, and (iv) meta-data about the database. Queries to the database are input to the policy reasoner. The policy reasoner reasons over the policies, meta-data about the database such as the fields in the database, possible values of the fields, and the distribution of values in the fields, and also takes into account past queries. The reasoner then infers whether the current query is compliant or not and provides an appropriate justification. If the query is compliant, it is forwarded to the database for execution otherwise the policy result and justification are returned to the requester. In either case, the query and justification are logged so they can be used for auditing purposes. For compliant queries, all associated usage restrictions are added to the results before being returned to the requester. In an ideal situation, a similar architecture could be present within the data consumer's context to verify that usage restrictions are being complied with. Usage restrictions can be complied with by up-front policy checking and violations detected through after-the-fact auditing.

3 Motivating Example

Consider a database consisting of information about US Persons. This data contains names, home address, telephone numbers, work address, email, fax number, marital status and other personal information. In order to allow this data to be used while maintaining privacy, our architecture is used to enforce semantic policies and help consumers understand what kinds of queries are allowed under them. Current access control policies for databases mainly consist of permitting/preventing access to rows, columns or views. However, our system is able to handle higher level policies that are not necessarily in terms of rows and columns of the database. Some examples of semantic policies include "You may not filter based on ethnicity", "You may not retrieve contact information of foreign nationals", and "You may not retrieve both Firstname and age for residents of New England" where the semantics of *ethnicity*, *contact information*, being a *foreign national* and being a *resident of New England* are defined using Semantic Web languages. Queries of the form "SELECT age, firstname WHERE birth-country=Nepal" and "SELECT * WHERE zipcode=02139" are matched against the policies and are identified as non-compliant whereas "SELECT lastname WHERE city=Cambridge" is identified as compliant.

4 Semantic Web-based Policy Language

Several policy languages in the literature could be used for policies about data sharing within a single security domain because a single domain typically uses uniform domain-specific semantics, is uniformly managed, and is less open compared to the cross-domain sharing scenarios. Though there has been some research on the problem of multiple security domains, it usually involves prior negotiation between separate domains in advance of the sharing that prevents dynamic sharing or sharing of new unaccounted-for information. We propose the use of Semantic Web technologies for policy descriptions that span muliple domains as they provide common models of data and knowledge such that entities (software and human) in the systems can exchange information, queries and requests with some assurance that they share a common meaning. The Semantic Web consists of several languages such as Resource Description Framework (RDF), RDF Schema (RDFS), and the Web Ontology Language (OWL), which can be used to provide a description of concepts, terms, and relationships within the domain(s), as well as technologies for the retrieval, use, and integration of these descriptions. These languages and technologies enable data, including policies and credentials, to be annotated with machine-understandable meta-data, allowing them to be automatically retrieved and used in correct contexts. Our policy language, AIR [4] is grounded in Semantic Web technologies [5], which allows it be domain independent and especially useful when policy is shared across domains that must adhere to the policy even though they have their own native schemas or data models. For example, it is possible to define a policy in AIR that describes the characteristics of users, the kinds of data that can be exchanged, and the conditions under which the exchange is possible between two government agencies, even though the data being used, the role descriptions, the user attributes, etc. used by each agency is different. The AIR reasoner accepts AIR policies, queries defined in the SPARQL Query Language for RDF [6], and domain knowledge such as the semantics of *New England residents* and *ethnicity* and infers whether the queries are compliant with the policy.

We use Semantic Web technologies not only to model policies but also for our justifications and logs. Having common semantics for this security information enables our tools to be used in any system that understands basic Semantic Web technologies and allows participants from different domains to effectively use and reason over each other's policies, logs and justifications.

In recent years there have been several efforts to develop expressive policy languages using Semantic Web technologies for a variety of application domains including network management, Web services, and privacy. These include languages such as KAoS [7] and Rei [8]. KAoS policies are OWL descriptions of actions that are permitted (or not) or obligated (or not). This limits the expressive power, but allows the classification of policy statements, enabling conflicts to be discovered from the rules themselves. Another advantage that KAoS has is that if policy descriptions stay within OWL-Lite or OWL-DL, then the computation is decidable and has well understood complexity results. On the other hand, Rei and AIR are more expressive and allow for rules to be defined over attributes of classes in the domain including users, resources, and the context, however, they lack well defined semantics. Rei also includes tools for policy analysis and speech acts for dynamic policy modification, which both AIR and KAoS lack. AIR is focused on generating explanations for policy decisions, which neither KAoS nor Rei are capable of, and is also able to handle noisy and inconsistent data from multiple sources by allowing users and rules to explicitly assert and manage provenance and other contextual information.

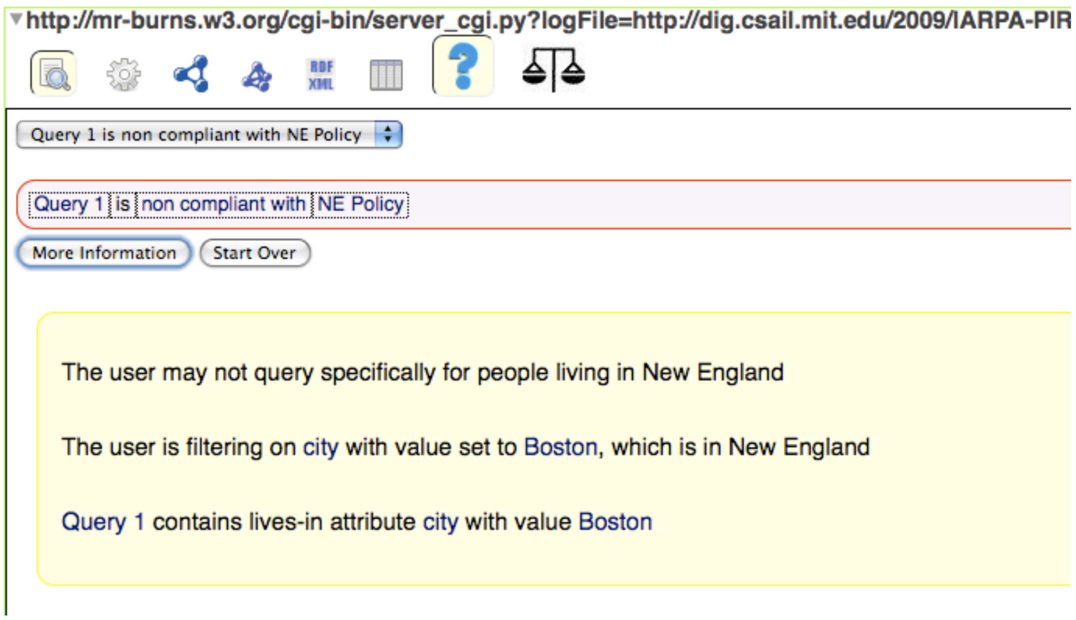
5 Integration with Different Databases

As the policy language itself is domain independent, it works with any database that can provide ontological meta-data. The meta-data of a database usually includes a description of the fields in the tables. For example, to model personal data you might say that Person is a class and has name, social security number, email, address and telephone number as properties. Further, address itself has several properties including street, house number, state, city, and zipcode. An important advantage of separating the domain knowledge from the policy language is the ability to define abstract concepts specific to different databases over which policies can be later defined. For example, for one database *Contact-Information* is an abstract concept containing email, address, telephone, fax, and office address of Person. It can be modeled in several ways using Semantic Web languages. One simple way is to make email, address, telephone, fax and office address sub-properties of *Contact-Information*. An example of a policy using *Contact-Information* is "You may not access Contact Information of minors" where *minors* can be defined as the class of those individuals whose age is less than 18. The meta-data can also include distribution of the values of a field such as "70 percent of the field gender is male". This means policies such as "You may not retrieve more than 50 percent of the database" will find queries such as "SELECT * WHERE gender=male" to be non-compliant.

6 Justification - Why and How ?

Most reasoners are capable of generating proofs for why they believe a certain conclusion to be true, however, these proofs are in the form of proof trees. These trees are difficult for end users to understand and contain a lot of irrelevant information. While reasoning over the policy and data, our reasoner maintains the set of dependencies for each statement in the knowledge base. It annotates all invoked rules in the dependencies with their natural language descriptions generating a machine-understandable justification embedded with English descriptions that a user can understand [4]. This information can be interactively explored with another of our policy tools called Justification UI (for more information, please see http://dig.csail.mit.edu/TAMI/2008/JustificationUI/howto.html). Figure 2 shows an example of a justification generated by the AIR reasoner. The policy states "You may not query for New England residents" and the query is "SELECT NAME, ID WHERE AGE>18 and CITY=BOSTON". The reasoner finds this query to be non-compliant and provides a justification for it.

Researchers have suggested other approaches to explanation generation such as the WhyNot [11] and KNOW [12] systems. Both these systems focus explicitly on failed queries and try to suggest changes to the input or knowledge base that would cause the queries to succeed. WhyNot uses abductive reasoning to compute partial proofs for how to satisfy the query. This implies requester has to sort through potentially irrelevant information to find the reason for failure. These proofs could also reveal private data and rules. The KNOW system searches all applicable policies using ordered binary decision diagrams to suggest modifications to the input that would cause the query to be satisfied. It also uses meta-policies to maintain privacy requirements of security policies. This privacy functionality is similar to what our approach provides. However, along with being domain independent and applicable to policies of all kinds, our approach also allows the system to capture and provide justifications for both successful and failed queries. Instead of using meta-policies, AIR supports syntax and semantics for adding natural-language descriptions to justifications and allows justifications to be declaratively


http://mr-burns.w3.org/cgi-bin/server_cgi.py?logFile=http://dig.csail.mit.edu/2009/IARPA-PIR
RDF XML
Query 1 is non compliant with NE Policy
More Information
Start Over

Query 1 is non compliant with NE Policy

The user may not query specifically for people living in New England

The user is filtering on city with value set to Boston, which is in New England

Query 1 contains lives-in attribute city with value Boston

Fig. 2. Justification Example: Under a policy that states “You may not query for New England residents”, the query “SELECT NAME, ID WHERE AGE>18 and CITY=BOSTON” is found to be non-compliant and this justification is provided.

modified preventing private information from being inadvertently leaked.

7 Private Policies and Justification

In most cases the default justification provided by the AIR reasoner is acceptable. However, policy administrators might want to modify these justifications if (i) parts of the policy or the data are private, or (ii) if the justification contains too much information. In these cases, some portions of the justification need to be suppressed and our policy language provides two mechanisms - *Hidden* rules and *Ellipsed* rules. Hiding rules is a mechanism for suppressing unimportant or private information from explanations. Declaring a rule to be hidden prevents all its deduction steps from appearing in the explanation. Consider a rule that says “Students have access to the database” and the statements “GradStudent is a subclass of student” and “PhDstudent is a subclass of GradStudent” are believed to be true. Then while proving that a PhD student has access to the data, there will be a rule to deduce the sub-class relationships, but it won’t be interesting for most end users. So we can declare the subclass rule as a *Hidden* rule to prevent this information from appearing in the explanation. Another mechanism is *Ellipsed* rule where we do not want to hide all steps following the rule but only want the execution of a specific rule to be suppressed. For example, if we have a rule that states “Unless a check is over X dollars it is not investigated for fraud” [1], it is not in the best interests of the policy administrator to reveal this “X”. In this case, he can declare the rule to be an ellipsed rule. Another example is if the data owner maintains a list of untrusted requesters or wants to limit the number of requests any single requester makes. Using *Ellipsed* rule will prevent only the rule that checks the list or the rule that checks the number of requests from appearing in the overall justification.

Even when policies are public, justifications are important because most policies especially those that relate to data sharing and usage are complex. A data consumer is not able to easily figure out which policies are applicable to a particular query and is not always capable of understanding these policies.

8 Intention, Usage, Context, and More

Knowing the intention of the user or the proposed use of the data being retrieved is usually extremely important for privacy policies and is very difficult for machines to ascertain. Many privacy policies include usage restrictions such as “My health information may not be used to contact me regarding

1. Thanks to KK Waterman for this example

potential clinical trials" that are virtually impossible to enforce at time of access. Our approach is to allow the user to state his or her purpose at request time, log this signed statement, and use it to make the policy decision. Later these logs can be audited and in case of a violation, this statement can be used to hold the requester accountable [1]. In order to be effective, however, appropriate regulation needs to be in place so action could be taken against the requester. A similar mechanism is used to model unexpected circumstances such as an emergency situation. If a privacy policy states "Only my primary healthcare provider may access my records unless there is a health emergency" then the system allows a requester who is not the primary healthcare provider to access the records if he or she can prove that there is an emergency or is willing to provide a signed statement of the fact. The system also annotates the query results with usage restrictions in the spirit of making policies explicit and understandable. The data consumer can use our policy tools to ensure that they are using data in compliance with their usage policies.

Another aspect of policies is context. By context we mean attributes of the requester, environmental variables, and attributes of the data itself. We can obtain certain contextual information using techniques such as FOAF+SSL [13], where users log in with a Uniform Resource Identifier (URI) that contains relevant information about them, rules about where to obtain some information will be present in the meta-data of the database but for other kinds of context we need to rely on input from the requester. Consider the following policy "You may only access this database if you've recently obtained XYZ certification". In order to know whether the requester has XYZ certification the reasoner will look in the meta-data for rules about which online resources can be trusted to vouch for certified users or will expect a signed statement from the requester. In order to handle these cases, we suggest that regular audits be run over the log so statements made by users can be verified and that violations of the policy can be identified.

9 Summary

Unlike most privacy approaches, we focus on helping both data consumers as well as data owners to share sensitive data according to privacy policies. We provide consumers with sufficient information to make correct queries enabling them to comply with complicated policies associated with accessing and using sensitive data. Along with this, by maintaining machine-processable logs the system is able to detect inappropriate information use through after-the-fact auditing.

Possible future work includes rewriting incoming queries so that they are compliant with the privacy policies and integration with the Privacy Integrated Queries (PINQ) toolkit [14] to allow the addition of a customizable amount of noise to the result of aggregate queries in order to prevent the leakage of sensitive information.

Acknowledgments

The authors wish to thank their colleagues for their contributions to this project. This work was supported in part by AFOSR YIP award FA9550-09-1-0152 and IARPA award number FA8750-07-2-0031.

References

[1] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman, "Information accountability," *Communications of the ACM*, June 2008.

[2] K. Krasnow Waterman, "Pre-processing legal text: Policy parsing and isomorphic intermediate representation," in *Intelligent Information Privacy Management Symposium at the AAAI Spring Symposium*, 2010.

[3] T. Breaux and A. Anton, "Analyzing regulatory rules for privacy and security requirements," *IEEE Transactions on Software Engineering*, vol. 34, pp. 5–20, 2007.

[4] L. Kagal, C. Hanson, and D. Weitzner, "Using dependency tracking to provide explanations for policy management," in *IEEE Policy 2008*, 2008.

[5] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web: Scientific american," *Scientific American*, May 2001. [Online]. Available: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&#38;pageNumber=1&#38;catID=2

[6] W3C, "Sparql rdf query language (sparql)," http://www.w3.org/TR/rdf-sparql-query/, 2008.

[7] J. Bradshaw, A.Uszok, R. Jeffers, N. Suri, P. Hayes, M. Burstein, A. Acquisti, B. Benyo, M. Breedy, M. Carvalho, D. Diller, M. Johnson, S. Kulkarni, J. Lott, M. Sierhuis, and R. V. Hoof, "Representation and reasoning about DAML-based policy and domain services in KAoS," in *2nd International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS2003)*, 2003.

[8] Lalana Kagal and Tim Finin and Anupam Joshi, "A Policy Based Approach to Security for the Semantic Web," in *2nd International Semantic Web Conference (ISWC2003)*, September 2003.

[9] P. A. Bonatti, D. Olmedilla, and J. Peer, "Advanced Policy Explanations on the Web," in *European Conference on Artificial Intelligence (ECAI)*, 2006.

[10] F. H.R., "Abductive reasoning as a way of worldmaking," *Foundations of Science*, vol. 6, pp. 361–383(23), 2001.

[11] H. Chalupsky and T. Russ, “Whynot: Debugging failed queries in large knowledge bases,” in *Fourteenth Innovative Applications of Artificial Intelligence Conference (IAAI-02)*, 2002, pp. 870–877.

[12] A. Kapadia and G. Sampemane and R. H. Campbell, “Know why your access was denied: regulating feedback for usable security,” in *11th ACM conference on Computer and Communications Security*, 2004, pp. 52–61.

[13] H. Story, B. Harbulot, I. Jacobi, and M. Jones, “FOAF+SSL: RESTful Authentication for the Social Web,” in *European Semantic Web Conference, Workshop: SPOT2009. Heraklion, Greece*, 2009.

[14] F. McSherry, “Privacy Integrated Queries,” in *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, June 2009.