### Citation

### As Published
http://dx.doi.org/10.1109/MSP.2010.89

### Publisher
IEEE Computer and Reliability Societies

### Version
Final published version

### Accessed
Wed Mar 16 11:05:14 EDT 2016

### Citable Link
http://hdl.handle.net/1721.1/71140

### Terms of Use
Article is made available in accordance with the publisher’s policy and may be subject to US copyright law. Please refer to the publisher’s site for terms of use.

### Detailed Terms

The MIT Faculty has made this article openly available. Please share how this access benefits you. Your story matters.
Various organizations are constantly collecting, analyzing, and storing individuals’ private data to, for example, provide better service and recommendations, improve healthcare, and enable national defense and law enforcement. Such practices raise privacy concerns, however, and efforts to address those concerns have been mostly 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. However, such techniques also discourage honest consumers from accessing data they require. With the current push toward need-to-share, we suggest that alternative approaches are required that encourage honest consumers to comply with the data owners’ privacy policies.

Our group at MIT is exploring accountability mechanisms as a means of protecting central information policy values such as privacy and fair and reliable information use. We view accountability combined with transparency and appropriate redress as a complementary process to strict access control. 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.

To this end, we’re building systems that provide policy awareness, enabling honest users to comply with explicitly defined policies and data owners to verify that those systems are enforcing their policies correctly. Policy awareness is a property of information systems that gives participants machine-readable representations of policies and audit trails to facilitate compliance with stated rules and helps them understand policy decisions.

We’ve designed an architecture based on policy awareness and accountability that enables users to preserve privacy while sharing sensitive data. We’ve implemented most of its key components, including a Semantic-Web-based policy language; a reasoner that infers whether queries are compliant, generates justifications, and handles private policies; a user interface that allows data owners and consumers to explore justifications; and techniques for integrating the architecture with different databases, as well as handling intention, usage, and context. Here, we define our architecture, describe its individual components, and illustrate our solution’s effectiveness through several examples.

Accessing Data via Policy Awareness

Most access control mechanisms provide just a yes/no or permit/deny type answer, leaving users unsure of what they’re doing wrong and suspicious of possible prejudices. Users wanting to comply with security and privacy policies end up either afraid to make queries for fear of doing something wrong or angry that they can’t get a query past the security mechanism to access the data they require. Our objective is to use policy awareness to help users make the right queries by giving them enough information to do so.

Consider a simple privacy policy, “You may not query for both zip code and last name.” With normal access control mechanisms, a requester who is asking for several fields, including zip code and last name, from the...
Database are defined using Semantic-Web-Based Policy Language for noncompliant ones. Let's them identify and execute only compliant queries and return justifications for noncompliant ones.

As Figure 1 illustrates, our architecture includes inserting a privacy-enabling engine in front of the database containing private data. This engine consists of a policy reasoner, an event log, a set of applicable policies, and metadata about the database. Users input queries to the database through the engine, which uses the policy reasoner to reason over the policies’ metadata—such as the database’s fields, possible field values, and the distribution of those values—and also takes into account past queries. The reasoner then infers whether the current query is compliant and provides an appropriate justification. If the query is compliant, the privacy-enabling engine forwards it to the database for execution; otherwise, the engine returns the policy result and justification to the requester. In either case, the system logs the query and justification to use for auditing purposes. For compliant queries, the engine adds all associated usage restrictions to the results before returning those results to the requester. In an ideal situation, a similar architecture could be present within the data consumer’s context to verify that the consumer is complying with usage restrictions. Consumers can comply with such restrictions via up-front policy checking, with violations detected through after-the-fact auditing.

Motivating Example
Consider a database consisting of information about people in the US. This data contains a person’s name, home address, telephone numbers, work address, email, fax number, marital status, and other personal information. To allow a data consumer to use this data while maintaining privacy, our architecture helps enforce semantic policies and lets consumers understand what kinds of queries are allowed under those policies. Current access control policies for databases mainly consist of permitting or preventing access to rows, columns, or views in a table. However, our system can handle higher-level policies that aren’t necessarily displayed in this manner.

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 first name 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 * WHERE birth-country=Nepal and SELECT * WHERE zipcode=02139 are matched against the policies and identified as noncompliant, whereas SELECT * WHERE city=Cambridge is identified as compliant.

Architecture
As Figure 1 illustrates, our architecture includes inserting a privacy-enabling engine in front of the database...
Sharing Sensitive Data

An important advantage to multiple security domains, it usually involves prior negotiation between separate domains that prevent dynamic sharing or sharing new, unaccounted-for information. We propose using Semantic Web technologies for policy descriptions that span multiple domains because they provide common data and knowledge models such that entities (software and human) in a system can exchange information, policy, and requests with some assurance that they share a common meaning. The Semantic Web uses several languages, such as the Resource Description Framework (RDF), RDF Schema (RDFS), and the Web Ontology Language (OWL), which data owners can use to provide a description of concepts, terms, and relationships within domains, as well as technologies for retrieving, using, and integrating these descriptions. These languages and technologies enable us to annotate data, including policies and credentials, with machine-understandable metadata, allowing a system to automatically retrieve and store them in the correct contexts.

Our policy language, Accountability in RDF (AIR), is grounded in Semantic Web technologies, which lets it be domain-independent and especially useful when users share policy across domains that must adhere to that policy even though they have their own native schemas or data models. For example, we can define a policy in AIR that describes user characteristics, the kinds of data those users can exchange, and the conditions under which the exchange is possible between, for example, two government agencies, even though the data being used, the role descriptions, and the user attributes each agency uses are different. The AIR reasoner accepts AIR policies, queries defined in SPARQL, and domain knowledge such as the semantics of New England residents and ethnicity, and infers whether the queries comply 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 lets users employ our tools 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.

Several recent efforts have attempted to develop expressive policy languages using Semantic Web technologies for a variety of application domains, including network management, Web services, and privacy. These include Knowledgeable Agent-oriented System (KAoS) and Rei. KAoS policies are OWL descriptions of actions that are permitted (or not) or obligated (or not). This limits their expressive power but lets KAoS classify policy statements, enabling the system to discover conflicts from the rules themselves.

KAoS has another advantage in that if policy descriptions stay within OWL-Lite or OWL-DL (description logic), then the computation is decidable and has well-understood complexity results. On the other hand, Rei and AIR are more expressive and allow rules to be defined over attributes of classes in the domain, including users, resources, and context; however, both languages 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 focuses on generating explanations for policy decisions, which neither KAoS nor Rei are capable of, and can also handle noisy and inconsistent data from multiple sources by letting users and rules explicitly assert and manage provenance and other contextual information.

Integration with Different Databases

Because AIR itself is domain-independent, it works with any database that can provide ontological metadata. A database’s metadata 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 the properties name, social security number, email, address, and telephone number. Furthermore, the address property itself has several subproperties, including street, house number, state, city, and zip code. An important advantage to separating the domain knowledge from the policy language is the ability to define abstract concepts specific to different databases over which administrators can later define their policies. For example, for one database, contact information is an abstract concept containing a person’s email, address, telephone, fax, and office address. We can model it in several ways using Semantic Web languages. One simple way is to make email, address, telephone, fax, and office address subproperties of contact information. An example of a policy using contact information is “You may not access contact information of minors,” where we can define minors as the class of individuals whose age is less than 18. The metadata can also include the distribution of a field’s values, 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 noncompliant.

Justification—Why and How?

Most reasoners can generate proofs for why they believe a certain conclusion to be true. However, these proofs are in the form of proof trees, which are difficult for end users to understand and contain a lot of irrelevant information. While reasoning over the policy and data, our reasoner maintains a set of dependencies for each statement in the knowledge base. It
annotates all invoked rules in the dependencies with their natural language descriptions (list of strings and bound variables), generating a machine-understandable justification embedded with English descriptions that a user can understand. Users can interactively explore this information with another of our policy tools, called the 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 the AIR reasoner generated. The policy states “You may not query for New England residents,” and the query is noncompliant, and the AIR reasoner provides a justification for it.

### Private Policies and Justification

In most cases, the default justification the AIR reasoner provides is acceptable. However, policy administrators might want to modify these justifications if parts of the policy or the data are private, or if the justification contains too much information. In these cases, administrators will need to suppress some portions of the justification, and our policy language provides two mechanisms for this—hidden rules and elided 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.” The statements “GradStudent is a subclass of student” and “PhDstudent is a subclass of GradStudent” are believed to be true, so to prove that a PhD student has access to the data, a rule to deduce the subclass relationships will exist. However, it won’t be interesting for most users. Thus, we can declare the subclass rule as hidden to prevent this information from appearing in the explanation.

The elided rule occurs when we don’t want to hide all steps following the rule but want to suppress only a specific rule’s execution. For example, if we have a rule that states “Unless a check is over X dollars, it is not investigated for fraud,” it isn’t in the policy administrator’s best interests to reveal X (our thanks to KK Waterman for this example). In this case, he or she can declare the rule to be an elided 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 the elided 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 can’t easily determine which policies are applicable to a particular query or always understand them.

### Intention, Usage, Context, and More

Knowing the user’s intention or the proposed use for the data a requester is retrieving is usually extremely important for privacy policies and is difficult for machines to ascertain. Many privacy policies include usage restrictions—such as “My health information may not be used to contact me regarding potential clinical trials”—that are virtually impossible to enforce at time of access. Our approach lets users state their purpose at request time, logs this signed statement, and uses it to make the policy decision. Later, data owners can audit these logs and, in case of a violation, use this statement to hold the requester accountable. To be effective, however, appropriate regulation must be in place so the owner can take action against the requester. We use a similar mechanism 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 lets requesters who aren’t the primary healthcare provider access the records if they can prove that there is an emergency or provide a signed statement of this fact. The system also annotates the query results with usage restrictions in the spirit of making policies explicit and understandable. Data consumers can use our policy tools to ensure that
Related Work in Justification Generation

Researchers have suggested other approaches to explanation generation, such as the WhyNot¹ and KNOW² 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 that the 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 the privacy requirements of security policies. This privacy functionality is similar to what our approach, Accountability in RDF (AIR), 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 modified, preventing private information from being inadvertently leaked.

References


they’re using data in compliance with usage policies.

Another aspect of policies is context, by which we mean the requester’s attributes, environmental variables, and data attributes. We can obtain certain contextual information using techniques such as Friend-of-a-Friend + Secure Sockets Layer (FOAF+SSL), where users log in with a URI that contains relevant information about them. Rules about where to obtain some information will be present in the database’s metadata, but for other kinds of context, we must rely on requester input. Consider the policy “You may only access this database if you’ve recently obtained XYZ certification.” To know whether the requester has XYZ certification, the reasoner will look in the metadata for rules about which online resources it can trust to vouch for certified users or will expect a signed statement from the requester. To handle these cases, we suggest that data owners run regular audits over the log to verify users’ statements and identify policy violations.

Unlike most privacy approaches, we focus on helping both data consumers and data owners 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 can help data owners detect inappropriate information use through after-the-fact auditing.

Future work involves rewriting incoming queries so that they’re compliant with privacy policies, and integrating our privacy-enabling engine with the Privacy Integrated Queries (PINQ) toolkit¹⁰ to let us add a customizable amount of noise to the result of aggregate queries to prevent sensitive information leakage.

Acknowledgments

We thank our colleagues for their contributions to this project. This work was supported in part by the Air Force Office of Scientific Research (AFOSR) award FA9550-09-1-0152 and Intelligence Advanced Research Project Activity (IARPA) award number FA8750-07-2-0031.

References

Sharing Sensitive Data


Lalana Kagal is a research scientist in the Massachusetts Institute of Technology’s Computer Science and Artificial Intelligence Lab and deputy director of the Decentralized Information Group. Her research interests include Semantic Web technologies, policy management, and privacy. Kagal did her postdoctoral work at MIT with Tim Berners-Lee and her PhD at the University of Maryland, Baltimore County, with Tim Finin. Contact her at lkagal@csail.mit.edu.

Joseph Pato is a distinguished technologist at Hewlett-Packard Labs and is currently a visiting fellow in the Massachusetts Institute of Technology’s Computer Science and Artificial Intelligence Lab with the Decentralized Information Group. His research interests include cybersecurity, identity systems, and privacy. Pato has an ScB in computer science from Brown University. He’s a member of IEEE and a senior member of the ACM. Contact him at joe.pato@hp.com.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.

IEEE Software offers pioneering ideas, expert analyses, and thoughtful insights for software professionals who need to keep up with rapid technology change. It’s the authority on translating software theory into practice.

www.computer.org/software/subscribe