Security Policy Refinement using Data Integration: A Position Paper*

Robert Craven
Department of Computing
Imperial College London
rac101@doc.ic.ac.uk

Jorge Lobo
IBM T.J. Watson
Research Center
jlobo@us.ibm.com

Emil Lupu
Department of Computing
Imperial College London
e.c.lupu@imperial.ac.uk

Alessandra Russo
Department of Computing
Imperial College London
ar3@doc.ic.ac.uk

Morris Sloman
Department of Computing
Imperial College London
m.sloman@doc.ic.ac.uk

ABSTRACT

In spite of the wide adoption of policy-based approaches for security management, and many existing treatments of policy verification and analysis, relatively little attention has been paid to *policy refinement*: the problem of deriving lower-level, runnable policies from higher-level policies, policy goals, and specifications. In this paper we present our initial ideas on this task, using and adapting concepts from *data integration*. We take a view of policies as governing the performance of an *action* on a *target* by a *subject*, possibly with certain *conditions*. Transformation rules are applied to these components of a policy in a structured way, in order to translate the policy into more refined terms; the transformation rules we use are similar to those of 'global-as-view' database schema mappings, or to extensions thereof. We illustrate our ideas with an example.

Categories and Subject Descriptors

K.6.4 [**Computing Milieux**]: Management of Computing and Information Systems—*System Management*; K.6.1 [**Computing Milieux**]: Management of Computing and Information Systems—*Project and People Management*

*Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

SafeConfig'09, November 9, 2009, Chicago, Illinois, USA.

Keywords

Policies, Refinement, Security, Authorization

General Terms

Management, Security, Design

1. INTRODUCTION

In this position paper we present an approach to the problem of *policy refinement*. This is the task of automating, so far as possible, the derivation of lower-level and eventually implementable policies from higher-level policies, specifications, and goals—and doing so in an efficient, sound and complete way. The need for automating refinement is of particular interest for the rapid and secure configuration of devices in military and non-military coalition operations. We generically speak of high-level goals, requirements and policies and of implementable configurations but in practice there are multiple levels of abstraction between the overall goal of an operation or mission and the concrete security configuration of the devices. There are goals or requirements, policies and a system on which policies operate, specified at each level of an abstraction hierarchy. The abstractions used to describe the desired system behaviour at one level are usually different from those used at a more concrete level below. For example, at one level a commander may specify policies in terms of missions, objectives and capabilities of units whilst at a more concrete level this would typically break down into individual tasks with their own objectives to be undertaken by specific devices or army personnel. Similarly, at one level a commander may specify requirements on the confidentiality and integrity of sensor data, this breaking down into individual permissions to access either sensor data or the sensors themselves.

Our proposal is to apply syntactic and algorithmic ideas from *data integration* [6] as a way of formalizing the transformation of the various components of a security policy, and allowing multiple different configurations of a policy to flow from a single higher-level specification.

2. BACKGROUND: DATA INTEGRATION

Data integration is the provision of a seamless interface to a number of heterogenous databases. The databases, associated for an application or the solution of a knowledge

problem, may have different schemata; these may be different from the schema in terms of which queries are posed to the application. All must be systematically related; data integration, studied in database theory, examines the sound and efficient generation, composition, and use of such relationships. We here describe the basics of several approaches to data integration, using an example about publishing.

Consider three databases s_1, s_2 and s_3, with the schemata:

$$s_1(\mathit{BookTitle}, \mathit{ISBN}, \mathit{Publisher})$$
$$s_2(\mathit{BookTitle}, \mathit{BookAuthor}, \mathit{Field})$$
$$s_3(\mathit{ISBN}, \mathit{Year/Month}, \mathit{IsHardback})$$

Most of these fields are self-explanatory; *Field* gives the Dewey code for subject-matter, and *IsHardback* is 1 where the book is hardback, 0 otherwise.

Let us suppose that the relations of interest in the application that has access to the databases—known as the *mediated schema*—are:

$$\mathit{book}(\mathit{ISBN}, \mathit{BookTitle}, \mathit{BookAuthor})$$
$$\mathit{subject}(\mathit{Field}, \mathit{BookTitle}, \mathit{Year/Month})$$
$$\mathit{publisher}(\mathit{ISBN}, \mathit{Publisher}, \mathit{IsHardback})$$

Queries to access the databases, formed within the application, are expressed in terms of the mediated schema. For example, if one wanted to know the titles and ISBNs of all hardback books on religions of Indic origin:

$$q(\mathit{Ttl}, \mathit{ISBN}) \leftarrow$$
$$\mathit{publisher}(\mathit{ISBN}, \_, 1), \mathit{subject}(294, \mathit{Ttl}, \_), \mathit{book}(\mathit{ISBN}, \mathit{Ttl}).$$

A central task for data integration here is to write rules that let the query be translated into queries to the component databases' schemata, s_1, s_2 and s_3. The *Global as View* or GAV approach allows Horn rules, with relations of the mediated schema in the head, and a conjunction of source relations as *positive* literals in the body. For example:

$$\mathit{book}(\mathit{ISBN}, \mathit{BookTitle}, \mathit{BookAuthor}) \leftarrow$$
$$s_1(\mathit{BookTitle}, \mathit{ISBN}, \_), s_2(\mathit{BookTitle}, \mathit{BookAuthor}, \_).$$

Query answering, for this type of rule, is a matter of *unfolding* the relations in the query. A different approach to the translation of queries into the terms of the component databases is to use *Local as View* or LAV, effectively the inverse of GAV. Here, relations of the source database are defined in terms of the mediated schema, for example:

$$s_1(\mathit{BookTitle}, \mathit{ISBN}, \mathit{Publisher}) \leftarrow$$
$$\mathit{book}(\mathit{ISBN}, \mathit{BookTitle}, \_), \mathit{publisher}(\mathit{ISBN}, \mathit{Publisher}\_).$$

Again, Horn clauses are used, but in this case query answering is not simply unfolding the relations; there are algorithms [9, 5] to transform, at a preliminary stage, LAV rules into a form with mediated schema relations in the head and source relations in the body—only then may unfolding proceed. Why then use LAV rules in the first place? A prominent advantage is the ease of update when databases are added to, or removed from, the pool. With GAV rules, the new source schema may appear in any rule defining a relation of the mediated schema, where the schemata for all other source databases may occur: so the relationships between the new database's schema and *all* other schemata must be systematically understood. In contrast, if LAV rules are used and a new database is added, the only mappings that need to be understood are between the new base's schema and the mediated schema.

3. POLICY REFINEMENT RULES

We conceive of policies as governing the performance of an *action* by a *subject* on a *target* under certain conditions. The conditions may refer to previous system history, policy logic, and actions. The domains where policies apply are described using *fluents* (for changing facts about system state), *static facts* (unchanging properties of systems) and *events*; the latter are akin to actions, but are not governed by policies. For example, an authorization policy that

> UK signal corps personnel are permitted to view low-resolution camera data from sensors in quadrant four if they have previously registered with central information services

could be expressed in our policy language as:

$$\begin{aligned}&\mathit{permitted}(\mathit{Sub}, \mathit{Tar}, \mathit{camera}(L), T) \leftarrow\\ &\quad \mathit{holdsAt}(\mathit{personnel}(\mathit{Sub}, \mathit{uk.signals}), T),\\ &\quad \mathit{object\_type}(\mathit{Tar}, \mathit{stillCam}),\\ &\quad \mathit{object\_property}(\mathit{stillCam}, \mathit{resolution}, \mathit{low}, L),\\ &\quad \mathit{do}(\mathit{Sub}, \mathit{centralInformation}, \mathit{register}, T'),\\ &\quad T' < T.\end{aligned}$$

The predicate *permitted* marks a positive authorization policy (*denied* would be used for negatives), with four arguments for the subject, target, action, and time at which the permission is given; this time is typically a variable which enables the time of the permission to be related by constraints to the times at which historical conditions must be true. In the policy above, it must *hold* that the subject of the policy is a member of the UK signal corps at any time T permission is to be given. At some previous time $T' < T$, the same subject must have 'done' a *register* action with *centralInformation* (this may also have been subject to a further policy). Only permission for low-quality still camera data is covered by the policy above; the levels that count as low are determined by the relevant *object_property* condition. For full details on the policy language we use, see [3] or [1]. We stress that although it is necessary to use a given formalism for representing policies, and describing the domains in which they operate (for the latter, we use the Event Calculus [8] or EC), common formalisms and languages can be translated into our language. In the case of domain descriptions, state charts and action languages, and commonly-used ontologies, can be translated into the EC. We have ourselves formulated translation schemes between our policy language and Ponder2 [4]; similar schemes can be framed for XACML2 [10], for instance.

There are several reasons why data integration and its methods could be used for policy refinement in general, and specifically for our approach to security policy analysis and refinement. Part of the process of refinement is to move from policies expressed in higher-level, abstract vocabularies to lower level vocabularies; the degree of abstraction concerns the detail with which subjects, targets, actions, fluents, static facts and events are represented. Therefore, we reason that vocabulary mapping rules, for each component of a policy, will be a key part of a refinement framework. We

believe these mapping rules would contain just the type information that the GAV or LAV clauses express, and that aspects of policy refinement that concerns vocabulary refinement is analagous to query reformulation within the context of data integration. Algorithms and theorems developed for data integration can be used or extended to apply to policy refinement. It is not hard to show that query reformulation using GAV rules is polynomial in the size of the query and rules. A transformed version of LAV rules and queries can be constructed using the *inverse rules* algorithm [9, 5] in time polynomial in the size of a conjunctive query and set of conjunctive source descriptions (i.e. LAV rules). The presence of PTIME complexity theorems in both cases is encouraging for the feasibility of their use for policy refinement.

We now describe and illustrate by example one category of these vocabulary mappings, or *refinement rules*: that concerning action refinement. We concentrate on an authorization policy, and focus on the policy without giving many details about the underlying domain. A preliminary step in policy refinement, before transformation rules can be applied, may be to formalize a natural language policy in our logical policy language (*may*, because policies may also be automatically derived, for example from security goals). The highest level of user-entered policy would thus be represented in a carefully circumscribed slice of natural language; elementary parsing transforms policies in such structured natural language into an equivalent form in our formal policy language (see [2] for first steps towards this). The example used is one fragment of a larger scenario we have developed to develop and test the application of ideas from data integration to many different aspects of refinement.

Thus, consider the following:

> High quality sensor information is reserved for US personnel only

Initial formalization, breaks this apart into one positive and one negative authorization policy:

$$\begin{aligned} &permitted(Sub, Tar, sensor\_data(high), T) \leftarrow \\ &\quad holdsAt(personnel(Sub, us), T), \\ &\quad holdsAt(sensor(Tar), T). \end{aligned} \tag{1}$$

$$\begin{aligned} &denied(Sub, Tar, sensor\_data(high), T) \leftarrow \\ &\quad holdsAt(personnel(Sub, us), T), \\ &\quad holdsAt(sensor(Tar), T). \end{aligned}$$

At the most abstract view of the system, as can be seen, there is a single action $sensor\_data(+Quality)$, where $Quality$ must take one of the ground values $\{low, medium, high\}$. The effects of the performance of such an action, if permitted, could be to set up a streaming channel or data relay between the requester and the data source. These effects would be modelled by the domain description; we omit the details.

Let us suppose that there are three types of sensor:

1. A still camera that can take pictures at three resolutions: 5, 3 and 1 Mega-pixels (Mp).
2. A video camera that can take videos with two resolutions: 2CIF (704×240) and 4CIF (704×480).
3. A global positioning system (GPS) with precisions 1 metre, 10 metres, 100 metres and 1000 metres.

Each of these types of sensor has an action that is the sensor-specific instance of the *sensor_data* action. Thus, pictures from the still camera are obtained using

$$camera(+Resolution)$$

Streamed videos from the video camera are obtained using

$$videoCam(+Resolution)$$

The locations of devices with GPS receivers are acquired with

$$gpss(+Device, +Precision)$$

Each of these actions may only be executed on the type of device mentioned: a subject may not, for instance, run a *videoCam* action on a still camera. This shows what ought to be intuitively obvious: there is a close logical link between targets and actions, forced because the nature of the target on which an action is performed determines the kinds of action that can be performed on it. The structure of action refinement rules should reflect this.

We give three examples of action refinement rules that transform the higher-level action *sensor_data* into device-specific forms. The first two relate to still cameras:

$$\begin{aligned} &Tar{:}sensor\_data(high) \leftarrow sensor(Tar). \\ &\qquad\Downarrow \\ &Tar{:}camera(5Mp) \leftarrow still\_camera(Tar) \end{aligned} \tag{2}$$

$$\begin{aligned} &Tar{:}sensor\_data(medium) \leftarrow sensor(Tar). \\ &\qquad\Downarrow \\ &Tar{:}camera(3Mp) \leftarrow still\_camera(Tar). \end{aligned} \tag{3}$$

The meaning of the first rule is that a $sensor\_data(high)$ action on a sensor can be understood, at a lower level of abstraction, as a *camera* action, with parameter *5Mp*, on a still camera. (It may be easier to put this the other way round: performing a *camera* action on a *still_camera* will *count as* [7], *implement*, or *be an instance of*, the *sensor_data* action on a sensor.) The second rule refines a $sensor\_data(medium)$ action into another *camera* action, but with a different configuration. Other rules apply to other types of device, for example:

$$\begin{aligned} &Tar{:}sensor\_data(high) \leftarrow sensor(Tar). \\ &\qquad\Downarrow \\ &Tar{:}videoCam(4CIF) \leftarrow video\_camera(Tar). \end{aligned} \tag{4}$$

Here, the high-level *sensor_data* action, with the *high* parameter, is refined into a different low-level action, on a different kind of target: a video camera.

In general, the form of an action refinement rule is:

$$\begin{aligned} &Tar{:}a(\vec{x}) \leftarrow l_1(\vec{x}_1), \ldots, l_m(\vec{x}_m). \\ &\qquad\Downarrow \\ &Tar'{:}a'(\vec{x}') \leftarrow l'_1(\vec{x}'_1), \ldots, l'_n(\vec{x}'_n). \end{aligned}$$

Everything above the vertical arrow is the *top*; everything below, the *bottom*. Tops and bottoms have a clause-like structure, whose heads contain a target and action, and whose bodies are conjunctions of fluents or fluents preceded by negation-as-failure **not**, or static facts possibly preceded by **not**.

Application of the action refinement rules above to the positive authorization policy (1) is straightforward. The head of the top of an action refinement rule is matched against the head of the policy rule: targets to targets, actions to actions. Clearly the heads of policy (1) and rule (2) match. Then the bodies are checked: all literals in the body of the top of (2) must match literals in the body of the policy (in the case of fluents, after being wrapped in the *holdsAt* predicate). In our case, they do. The actions and literals are then replaced, in the policy, with the bottom of the refinement rule, to produce:

$$\begin{aligned} &permitted(Sub,\, Tar,\, camera(5Mp), T) \leftarrow \\ &\quad holdsAt(personnel(Sub, us), T), \\ &\quad holdsAt(still\_camera(Tar), T). \end{aligned} \tag{5}$$

Rule (3) does not match, in this way, the original policy (1). But the third action refinement rule (4) does, to produce:

$$\begin{aligned} &permitted(Sub,\, Tar,\, videoCam(4CIF), T) \leftarrow \\ &\quad holdsAt(personnel(Sub, us), T), \\ &\quad holdsAt(video\_camera(Tar), T). \end{aligned}$$

At the next level of concreteness, permission to view images produced from by still cameras may be refined into a form close to SQL queries to access a picture database where such images are stored. Consider the following rule:

$$\begin{aligned} &Tar{:}camera(5Mp) \leftarrow still\_camera(Tar). \\ &\qquad\qquad \Downarrow \\ &picDB{:}select(res5Mp). \end{aligned}$$

Application of this refinement-rule to the policy (5) gives the following authorization policy:

$$\begin{aligned} &permitted(Sub,\, picDB,\, select(res5Mp), T) \leftarrow \\ &\quad holdsAt(personnel(Sub, us), T), \end{aligned}$$

At this point, the security policy is close to the content of a SQL query such as

GRANT SELECT ON picDB.res5CIF TO USid@ahost

Subject refinement (which we do not illustrate here) supplies the remaining phases of refinement, and the policy can then be straightforwardly translated from our logical language into SQL. (Of course, we use SQL as an example: refinement rules and translation schemes could be produced for many different implementable languages.)

In this way multiple action refinement rules can control the setting of different configurations for security policies: the same high-level policy can be refined in different ways, producing different settings appropriate for each of the devices to which the policy applies, and the policies that result from refinement can be expressed in different languages. The action-refinement rules we have shown here do not correspond exactly to the syntax of GAV or LAV rules for data integration. We have found that in the case of fluent and static fact refinement, either GAV or LAV rules are suitable, but that action refinement requires an extension of the syntax, allowing reference to conditions qualifying the target and action. Such an extension of the syntax does not, we believe, increase the computational complexity of using the rules for refinement. Subject refinement, and a more general form of target refinement than that shown above, involves different considerations again, and is current work.

4. CONCLUSION

We described the essence of one thread of our approach to policy refinement: the use of data integration as starting point for refinement specification. We believe that the existence of algorithms for query reformulation, and their low computational complexity, is very promising. But data integration needs to be complemented to cover many other issues of refinement. For example, a high-level goal may be achieved by several combinations of concrete actions i.e., the action decomposition process typically has several solutions. It is therefore necessary to choose amongst them through some algorithm that selects between alternative refinement solutions. Also refinement must be interleaved with analysis. This will ensure that refined policies are implementable on the target systems and do not violate predefined system constraints. Parallel, distributed refinement could lead to policies which conflict with each other as the refinement context will not be the same across all entities. We have an approach for centralized policy analysis [3] that can be used for policies during, or at the end of, the refinement process; in a related strand of current work we are generalizing this to distributed policy analysis.

5. REFERENCES

[1] A. Bandara, S. Calo, R. Craven, J. Lobo, E. Lupu, J. Ma, A. Russo, and M. Sloman. An expressive policy analysis framework with enhanced system dynamicity. TR, Department of Computing, Imperial College London, 2008.

[2] C. Brodie, C.-M. Karat, and J. Karat. An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench. In L. F. Cranor, editor, *SOUPS*, volume 149 of *ACM International Conference Proceeding Series*, pages 8–19. ACM, 2006.

[3] R. Craven, J. Lobo, J. Ma, A. Russo, E. Lupu, A. Bandara, S. Calo, and M. Sloman. Expressive policy analysis with enhanced system dynamicity. In *ASIACCS*, pages 239–250. ACM, 2009.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In M. Sloman, J. Lobo, and E. Lupu, editors, *POLICY*, volume 1995 of *LNCS*, pages 18–38. Springer, 2001.

[5] O. M. Duschka and M. R. Genesereth. Query planning in infomaster. In *SAC*, pages 109–111, 1997.

[6] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *J. Log. Program.*, 43(1):49–73, 2000.

[7] A. J. I. Jones and M. J. Sergot. A formal characterisation of institutionalised power. *Logic Journal of the IGPL*, 4(3):427–443, 1996.

[8] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[9] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB*, pages 251–262. Morgan Kaufmann, 1996.

[10] OASIS XACML TC. extensible access control markup language (XACML) v2.0, 2005.