1
On Access Control, Data Integration, and Their Languages

Martín Abadi

This paper considers the goals and features of recent languages for access control in distributed systems. In particular, it relates those languages to data integration.

Languages for access control

Access control is central to security, and in computer systems it appears in many guises and in many places. Applications, virtual machines, operating systems, and firewalls often have their own access-control machinery, with their own idiosyncrasies, bugs, and loopholes. Physical protection, at the level of doors or wires, is another form of access control.

Over the years, there have been many small and large efforts to unify models and mechanisms for access control. Beyond any tiny intellectual pleasure that such unifications might induce, these may conceivably contribute to actual security. For example, when there is a good match between the permissions in applications and those in the underlying platforms, access control mechanisms may have clearer designs, simpler implementations, and easier configurations. The benefits are however far from automatic—the result is sometimes more problematic than the sum of the parts—and there probably will always be cases in which access control resorts to *ad hoc* programs and scripts.

Those efforts have sometimes produced general languages for access control (e.g., [2–5,7,10,11]). The languages are flexible enough for programming a wide variety of access control policies (for example, in file systems and for digital rights management). They are targeted at distributed systems in which cryptography figures prominently. They serve for expressing the assertions contained in cryptographic credentials, such as the association of a principal with a public key, the membership of a principal in a group, or the right of a principal to perform a certain operation at a specified time. They also serve for combining credentials from many sources with policies, and thus for making authorization decisions. More broadly, the languages sometimes aim to support trust management tasks.

Several of the most recent language designs rely on concepts and techniques from logic, specifically from logic programming: Li et al.'s D1LP and RT [10, 11], Jim's SD3 [7], and DeTreville's Binder [4]. These are explicitly research projects. Languages with practical aims such as XrML 2.0 include some closely related ideas, though typically with less generality and simpler logic. This note will focus on Binder.

One might question whether the use of these sophisticated languages would reduce the number of ways in which access control can be broken or circumvented. Policies in these languages might be difficult to write and to understand—but perhaps no worse than policies embodied in Perl scripts and configuration files. There seem to be no hard data on this topic.

A look at Binder

Binder is a good representative of this line of work. It shares many of the goals of other languages and several of their features. It has a clean design, based directly on that of logic-programming languages.

Basically, a Binder program is a set of Prolog-style logical rules. Unlike Prolog, Binder does not include function symbols; in this respect, Binder is close to the Prolog fragment Datalog. Also, unlike Prolog, Binder has a notion of context and a distinguished relation `says`.

For instance, in Binder we can write:

```
may-access(p,o,Rd) :- Bob says may-access(p,o,Rd)
may-access(p,o,Rd) :- good(p)
```

These rules can be read as expressing that any principal `p` may access any object `o` in read mode `(Rd)` if `Bob` says that `p` may do so or if `p` is good.

Here only `:-` and `says` have built-in meanings. The other constructs have to be defined or axiomatized. As in Prolog, `:-` stands for reverse implication ("if"). As in previous logical treatments of access control, `says` serves to represent the statements of principals and their consequences [1]. Thus,

```
Bob says may-access(Alice,Foo.txt,Rd)
```

holds if there is a statement from `Bob` that contains a representation of the formula

```
may-access(Alice,Foo.txt,Rd)
```

More delicately,

```
Bob says may-access(Alice,Foo.txt,Rd)
```

also holds if there is a statement from `Bob` that contains a representation of the formula

```
may-access(Alice,Foo.txt,RdWr)
```

and another one that contains a representation of the rule

```
may-access(p,o,Rd) :- may-access(p,o,RdWr)
```

The author of an access control policy need not be concerned with the details of how formulas are associated with piles of bits and network protocols. In particular, `says` abstracts from the details of authentication. When `C says S`, `C` may send `S` on a local channel via a trusted operating system within a computer, on a physically secure channel in a machine room, on a channel secured with shared-key cryptography, or in a certificate with a public-key digital signature.

Each formula is relative to a context. In our example, `Bob` is a context (a source of statements). Another context is implicit: the local context in which the formula applies. For example,

```
may-access(p,o,Rd) :- Bob says may-access(p,o,Rd)
```

is to be interpreted in the implicit local context, and `Bob` is the name for another context from which the local context imports statements. This import relation might be construed as a form of trust.

There is no requirement that predicates mean the same in all contexts. For example, `Bob` might not even know about the predicate `may-access`, and might assert

```
peut-lire(Alice,Foo.txt)
```

instead of

```
may-access(Alice,Foo.txt,Rd)
```

In that situation, one may adopt the rule:

```
may-access(p,o,Rd) :- Bob says peut-lire(p,o)
```

On the other hand, Binder does not provide much built-in support for local name spaces. A closer look reveals that the names of contexts have global meanings. In particular, if `Bob` exports the rule

```
may-access(p,o,Rd) :-
      Charlie says may-access(p,o,RdWr)
```

the local context will obtain

```
Bob says may-access(p,o,Rd) :-
      Charlie says may-access(p,o,RdWr)
```

without any provision for the possibility that `Charlie` might not be the same locally and for `Bob`. Other systems, such as SDSI/SPKI [5], include more elaborate naming mechanisms.

Distributed access control as data integration

In the database field, a classic problem is how to integrate multiple sources of data. The basic problem set-up is that there is a collection of databases, each defining some relations, and one wants to do operations (in particular queries) on all of them. The query language may be some variant of Prolog, or of its fragment Datalog. Modern versions of the problem address the case where some or all of the sources of data provide semi-structured objects—on the Web in XML, for instance. The languages vary accordingly.

Each database may expose a different interface and export its data in a different format. In systems such as Tsimmis [6,12], wrappers translate data from each source into a common model. Mediators then give integrated views of data from multiple (wrapped) sources. For instance, the following is a mediator, written in the language MSL (Mediator Specification Language) of Tsimmis:

```
<cs_person {<name N> <relation R> Rest1 Rest2}>@med :-
      <person {<name N> <dept 'CS'> <relation R> |
              Rest1}>@whois
      AND decompose_name(N, LN, FN)
      AND <R {<first_name FN> <last_name LN> | Rest2}>@cs
```

This mediator defines an information source `med` in terms of two others, `whois` and `cs`. A query to `med` on `cs_persons` results in two queries, one on `whois` and one on `cs`, plus a call on the external predicate `decompose_name`. In expressions of the form `<...>@s`, `s` is a site: a constant or a variable that represents an information source. The details, which are unimportant for present purposes, can be found in Papakonstantinou's dissertation [12].

MSL and Binder have more in common than their proximity to Datalog. Both deal with multiple sources of data (sites or contexts). In Binder, access control policies may be regarded as mediators that integrate data from multiple contexts. Each context may define some relations (`good`, `may-access`, etc.), so we may as well regard contexts as databases. However, the databases may be implemented by certificates, rather than with big tables (so revocation and negation can be difficult). There is even a remarkable syntactic similarity between MSL and Binder, at least at the level of abstract syntax: `@` in MSL is analogous to `says` in Binder, and we may read `P@s` as `s says P`.

These similarities suggest the possibility of exploiting ideas and methods from databases in security. For instance, we may borrow implementation techniques and some theory. We may also borrow some language design. The thought of basing access control on semi-structured data is inevitable but somewhat frightening. More conservatively, languages for access control may incorporate important query-language constructs that go beyond first-order logic and Datalog, for example for aggregating data.

While MSL and Binder have similarities in syntax and semantics, their pragmatics are quite different. In short, the two languages are used in different environments, for different purposes, and under different constraints.

- Work on data integration seems to assume a messy but benign world. This attitude may sometimes motivate pragmatic shortcuts, for example the plausible assumption that two relations with the same name in different sites might be intended to mean the same unless stated otherwise.
- In security, on the other hand, we tend to regard data from foreign contexts with a healthy dose of distrust. While users may work around mistakes in data integration, and tolerate them as ordinary bugs, mistakes in access control are vulnerabilities, often with serious consequences.

The term “views,” so often used in data integration, suggests that each source of data provides part of the truth on a whole. The literature on data integration explores two possible approaches [9]:

- global-as-view (GAV): each relation in the mediator schema is defined by a query over the data sources;
- local-as-view (LAV): the data sources are defined by queries over the mediator schema.

Both approaches have benefits in data integration. On the other hand, Binder seems to fit only the GAV model; it is not clear how the LAV model might apply in distributed access control.

Security is primarily a property of systems, not a property of languages. The observation that some “security languages” resemble some “data integration languages” seems intriguing, and perhaps useful, but it mostly ignores the systems for which the languages were invented.

Nevertheless, distributed access control is at least partly about data integration. We may therefore hope that advances in data integration, and more broadly in databases, would eventually be of some benefit in security. We may even imagine that we will be able to dispense with much of the special machinery for access control, relying instead on systems for data integration and the like (e.g., [8]), by subsumption. Whether that outcome would be good, rather than merely interesting, remains open to debate.

Acknowledgments

I am grateful to John DeTreville, Phokion Kolaitis, Butler Lampson, Roger Needham, Dan Suciu, and Wang-Chiew Tan for discussions that contributed to this note and to Mike Burrows for comments on the presentation of a draft. This work was partly supported by the National Science Foundation under Grants CCR-0204162 and CCR-0208800.

References

1. ABADI, M., BURROWS, M., LAMPSON, B. AND PLOTKIN, G., 'A calculus for access control in distributed systems,' *ACM Trans. on Programming Languages and Systems,* vol. 15, no. 4, September 1993, pp. 706-734.
2. BLAZE, M., FEIGENBAUM, J., IOANNIDIS, J. AND. KEROMYTIS, A.D., The KeyNote trust-management system, version 2. IETF RFC 2704, September 1999.
3. BLAZE, M., FEIGENBAUM, J. AND LACY, J., 'Decentralized trust management,' *Proc. 1996 IEEE Symposium on Security and Privacy*, pp. 164-173.
4. DETREVILLE, J., 'Binder, a logic-based security language,' *Proc. 2002 IEEE Symposium on Security and Privacy*, pp. 105-113.
5. ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B. AND YLÖNEN, T,. SPKI certificate theory. IETF RFC 2693, September 1999.
6. GARCIA-MOLINA, H., PAPAKONSTANTINOU, Y., QUASS, D., RAJARAMAN, A., SAGIV, Y., ULLMAN, J.D., VASSALOS, V. AND WIDOM, J., 'The TSIMMIS approach to mediation: data models and language,' *Journal of Intelligent Information Systems,* vol. 8, no. 2, 1997, pp. 117-132.
7. JIM, T., 'SD3: A trust management system with certified evaluation,' *Proc. 2001 IEEE Symposium on Security and Privacy*, pp. 106-115.
8. JIM, T. AND SUCIU, D., 'Dynamically distributed query evaluation,' *Proc. 2001 ACM Symposium on Principles of Database Systems*, pp. 28-39.
9. LENZERINI, M., Slides of the invited tutorial 'Data integration: a theoretical perspective,' given at the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2002, available at: http://www.dis.uniroma1.it/~lenzerin/homepagine/publifile.html.
10. LI, N., GROSOF, B.N. AND FEIGENBAUM, J, 'Delegation logic: A logic-based approach to distributed authorization,' *ACM Trans. on Information and System Security*, vol. 6, no. 1, February 2003, pp. 128-171.
11. LI, N., MITCHELL, J.C. AND WINSBOROUGH, W.H., 'Design of a role-based trust-management framework,' *Proc. 2002 IEEE Symposium on Security and Privacy*, pp. 114-130.
12. PAPAKONSTANTINOU, I.G., 'Query processing in heterogeneous information systems'. Doctoral Dissertation, Stanford University, 1997, available at: http://www.db.ucsd.edu/people/yannis.htm.