Firewall Policies Definition Tools: An Implementation Idea

Patrizia Asirelli and Fabrizio Fabbrini
Istituto di Elaborazione della Informazione - CNR
Via S. Maria 46, I-56126 Pisa, Italy
Tel: +39-050-593477, Fax: +39-050-554342, email: asirelli@iei.pi.cnr.it

Key words: logic databases, integrity constraints, security policies, firewalls.

Abstract:
We present some ideas for a declarative approach to the implementation of a tool to define firewall policies. Our aim is to show how a deductive system, such as a deductive database management system, can be used to build a tool that a firewall administrator can use to define its policy. We present a firewall example only to highlight the advantages of such type of approach as a policy definition tool. The deductive database system we have used, besides the obvious deductive capabilities, has the ability of structuring the necessary knowledge into parts, the capability of composing the parts together by means of importing mechanisms and the ability to define and prove properties of the policy.

1. INTRODUCTION

The Internet and the World Wide Web capability are producing an explosion of information available on line. There is an ever-growing wish and need to grasp information wherever it is located and this is unfortunately true also for malicious users.

Security and privacy are becoming more crucial [JA96a], [JA96b], [BSJ97]. The need for an organisation to protect itself both from outsider and insider malicious users is growing so much that it makes even the existence of an organisation to depend on the safety and security of its internal information bases.

Every time an organisation connects its local trusted network - the organisation's "intranet" - to the Internet (for sending e-mail, downloading files, exchanging documents, browsing the web, and so on), there is the chance that protocols and

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35501-6_14

M. E. van Biene-Hershey et al. (eds.), *Integrity and Internal Control in Information Systems*

services can be abused by malicious outsiders, endangering the privacy/integrity/availability of the organisation's data.

For many sites, the most practical way to make the internal network much more resistant to external attacks [Ches94] is to use a Firewall: this is a system (router, personal computer, host) specifically set up to shield a site or a subnet from inherently vulnerable protocols and services, such as TCP/IP, NFS or NIS. Even though a firewall system is usually located at a higher-level gateway, such as a site's connection to the Internet, firewalls can be located at lower-level gateways to provide protection for some smaller subset of hosts or subnets. Firewalls are thus instruments that often organisations adopt to defend themselves from the outside world but they can also be used as protection mechanisms from some internal possible attack. Thus, a firewall can also be considered a tool for implementing an internal security policy. In fact, it can be used:

- to define network use and responsibilities,
- to identify who is allowed to use which network resources,
- to establish who is authorised to grant/deny access,
- to define auditing requirements,
- to prepare recovery plans.

The aim of this paper is to show that a deductive tool, such a deductive database management system, can be used as an implementation instrument to help the administrator of a firewall to define its security policy.

We have used a firewall example just only because it is an example of a system where the security policy is crucial for the system to meet its objectives. In fact, a firewall definition [Bar99] can be divided into two parts, a policy definition part and a topology definition part. We have been able to face both definitions within the same approach. We claim that the same approach could be applied to other systems or organisations, (e.g. a bank or a multi-branch company), where the administrator has to define a **policy**: in a structured way, with the ability of performing some reasoning on it (such as proving some properties), with the ability of being certain that some properties hold and, furthermore, with some ability of trying the policy out on examples, that is, the possibility of performing some prototyping of the policy.

In other words, we claim that the great advantages of the system we have used are the following:

- its ease of use: the language consists of simple definite clauses and the system is provided with a very simple interface;
- the possibility for the administrator of structuring the body of knowledge into different theories;
- the ability of proving properties of each body of knowledge by being able to define and prove integrity constraints on the theory;

- the ability to put together all bodies of knowledge by means of importing mechanisms and thus being able to prove further properties on the resulting composed knowledge,
- the ability to define and build output body of knowledge that can be passed onto other different tools. For example, this means the ability of producing a final body of rules to pass onto a generic firewall system, once the semantic consistency of the firewall policy has been checked. Finally,
- to run the defined policy on examples, thus being able of prototyping the policy itself.

In this paper we have based our example on the firewall definition tool presented in [Bar99]. As a matter of fact we were able to spot some mistake in the Entity Relationship Model they gave in their paper. May be they were typos, but anyway, when we gave the same policy definition with the same examples we found out that some constraints were violated: we can then formally conclude that either the model they give is wrong or the examples are.

The paper is intended to be a first attempt to try out this idea of constructing a policy definition tool within a deductive database system. It is preliminary in at least two ways: one concerns this particular firewall example, the other concerns our long-term objective. With respect to this example, we show the theories implementing the Entity Relationship Model of [Bar99], while we have not implemented yet the operational part for producing the final rule-base that has to be given to a firewall. This is going to be our next issue although we strongly believe that it should be quite straightforward.

Concerning our long-term aim instead, we want this to be the first step towards the study and construction of a tool for validating systems with respect to their security policy aspects.

In the next section we briefly introduce JEDBLOG, the deductive system we use. In section 3 we discuss the possible implementation, within JEDBLOG, of the firewall definition tool of [Bar99]. We finally conclude in section 4.

2. THE JEDBLOG SYSTEM

JEDBLOG is the new version of GEDBLOG [Asi94], a prototype deductive database management system [AHV95], [Ull88], where the main difference is the Java interface while both systems are implemented in SICStus Prolog [SICS95].
JEDBLOG supports fast prototyping of applications that can take benefit from a declarative specification style. It is based on a logic language extended with the capabilities of:

- handling separate theories, i.e. separate pieces of knowledge;

- defining and executing transactions, i.e. compound updates to the theory;
- defining and verifying integrity constraints.

More precisely, JEDBLOG is a deductive (logic) database that can deal with basic knowledge management functionality (storing, retrieving, and querying), and besides it is enriched with several additional features like:

- Integrity Constraints and Checks, to define the data model entities must fit in;
- Transactions, to enter the operational framework.

The system window manager looks as shown in Fig. 1 and 2 of the Appendix.

JEDBLOG can manage logical theories that consist of different kinds of clauses: Facts, Rules, Integrity Constraints, Checks and Transactions. The syntax of the different pieces of knowledge is the following:

- Facts :< *Fact* >.
- Rules :< *Head* > <-- $< B_1 >$ &...& $< B_n >$.
- IC :< *Condition* > >> $<B_1 >$ &...& $< B_n >$.
- Check :< C_1> &...& <C_n> ==> <B_1> &...& <B_n>.
- Transactions :< *Transaction* > := <*Pre*> # <*Body*> # <*Post*>.

Example:

- **Facts**:
  age(mary,20).
  age(fred,30).
- **Rules**:
  accommodation(Dh,Ah,1,1,1) <•• (Ah-Dh) > 1.
  departure(Company,Flight,From,To,DH,DM,AH,AM) <•• airport(X,From) & flight(Company,Flight,X,Y,DH,DM,AH,AM) & airport(Y,To).
- **Integrity Constraints**:
  flight(meridiana,X,Y,Z,W,E,R,T) >> sardegna(Y).
- **Checks**:
  age(gianni,X) & X<40 ==> false.
  true ==> age(rosa,X) & X<30.
- **Transactions**:
  new(X,Y) := true # in(age(X,Y)) # true.

There are two primitive pre-defined transactions:

1. in (<Theory>, <Fact>).
   to insert the fact <*Fact*> into the theory <*Theory*>
2. out (<Theory>, <Fact>).
   to delete the fact <*Fact*> from theory <*Theory*>, if there is one such fact in it.

Importing mechanisms:

a) By means of the system pre-defined predicate *theory*, it is possible to perform inclusion among theories. In this way, given a *starting* theory *Th*, the associated

JEDBLOG theory can be defined as the set-theoretic *union* of all the theories in the inclusion tree rooted in *Th*. The theories are included at run-time each time the starting theory is opened.

b) The system also permits to import theories (see the *import* command in the file menu of the manager window in Fig.1), where the imported theory is copied into the importing one giving rise to a bigger theory that remains as such from one session to another. This realises a sort of "materialisation" mechanism [LU95]. Imported theories can always be "deleted".

3. A FIREWALL IMPLEMENTATION

According to the implementation, firewall systems can be referred to as packet filters or application gateways, or a combination of both [Ha96]. Regardless of the kind of implementation, using a firewall makes it possible, at the organisation level:

- to concentrate security handling,
- to control access to private network systems,
- to notify security-related events to the security manager,
- to insert strong authentication mechanisms,
- to enhance privacy,
- to create log for security audits,
- to enforce the security policy itself.

A packet filter simply filters packets that pass to and from the Internet and the protected subnet. Examining the contents of the fields (usually the source address and destination address) of every packet going in or out of the organisation's network, the filter can prevent packets from certain addresses from entering the network and other packets from leaving:

- limiting or disabling services such as telnet,
- restricting access to and from specific domains,
- hiding information about the subnet.

Application gateways, also referred to as "proxy servers", provide a higher level of security through the scanning of the contents of the Internet traffic and are typically located so that all application traffic (generally telnet, ftp and e-mail traffic) addressing hosts within the protected subnet must first be sent to the gateway: the application gateway is usually slower than a packet filter, but it allows for a higher level of security to be gained.

The results of using a firewall depend on the decision to filter certain protocols and fields, which in turn depends on the organisation's security policy. Thus, as we claimed before, a firewall can also be considered as a tool to handle some aspects of internal security.

We have considered the Entity Relationship model defined in [Bar99] and from there we have derived an equivalent model that in our case is only defined by means of constraints. Our implementation example is also taken from [Bar99]. Thus our approach has been as follows.

For every relation *r* in their model
a) we define a new JEDBLOG theory *r_base*;
b) we build constraints formulas that define the properties of *r* (i.e. its type, attributes, etc.);
c) we import all theories defining relations r_1, ... , r_n if r_1, ..., r_n are used in *r_base* but not defined in it.

Thus we have theories *protocol_base*, *service_base*, *service_group_base*, etc. Furthermore, we have defined other theories such as *port*, *service_names*, etc., as separated bodies of knowledge, that we considered more handy for the administrator to decide what ports and services to allow. Of course this is an example and each administrator can decide on his/her own requirements.

In the following we list all theories we have actually implemented. For each one of them we give the constraints, the other theories they use and the rules, if any. More explanations are given below.

- **Theory:** *port_base*:
  IC&Checks=none;
  Import=none;

- **Theory** *service_names*:
  IC&Checks =none;
  Import=none;

- **Theory** *protocol*:
  IC&Checks =none;
  Import=*port_base*;

- **Theory** *service_base*:
  IC&Checks =service(X,Y)>>service_name(X)&protocol(Y).;
  Import=*service_names*, *protocol_base*;

- **Theory** *service_group*:
  IC&Checks=service_group(X,Y)>>service_group_name(X) & service(Y).;
  Import=*allowed_ser_gr*;

- **Theory** *basic_definition*:
  IC&Checks =none;
  Import=*role_names*, *role_group_names*;

- **Theory** *role_names*:
  IC&Checks =none;
  Import=none;

- **Theory** *role_goup_names*:
  IC&Checks =none;
  Import=none;

- **Theory** *role_base*:
  IC&Checks=role(X,Y,Z,W)>>role_name(X) & in_out(Y) & peer(Z) &
  service_id (W).;
  Rules: service_id(X) <-- service(X).
  service_id(X) <-- service_group(X).;
  Import=*service_group*, *basic_definitions*;

- **Theory** *role_group_base*:
  IC&Checks=role_group(X,Y)>>role_group_name(X)& peer(Y).;
  Import=*service_group_names*, *basic_definitions*;

- **Theory** *ip_addresses*:
  IC&Checks =none;
  Import=none;

- **Theory** *host_names*:
  IC&Checks =none;
  Import=none;

- **Theory** *host_base*:
  IC&Checks =host(X,Y,Z) >> host_name(X) & ip_address(Y) & peer(Z).;
  Import=*ip_addresses*,*host_names*, *basic_definitions*;

- **Theory** *host_group_names*:
  IC&Checks =none;
  Import=none;

- **Theory** *host_group_base*:
  IC&Checks =host_group(X,Y,Z,W) >> host_group_name(X) & ip_address(Y)
  & ip_address(Y) & peer(Z).;
  Import= *ip_addresses*, *host_group_names*, *basic_definitions*;

- **Theory** *gateway_names*:
  IC&Checks =none;
  Import=none;

- **Theory** *gateways_base*:
  IC&Checks =gateway(X,Y)>>gateway_name(X) & host(Y,Z,W).;
  true ==> gateway(X,Y) & gateway(X,Z)&Y=/=Z.
  Import= *gateway_names*, *host_base*;

- **Theory** *zone_base*:
  IC&Checks =zone(X,Y)>>zone_name(X) & gateway(Y,Z);
  Import=none;

The above theories, besides defining the equivalent of the Entity Relationship Model of [Bar99], provide a framework to define the "actual" model of the firewall. In [Bar99] it is claimed that they are able to handle both the definition of the firewall policy and the definition of the firewall topology in a distinct way. The same is possible in our approach that also provide a way of prototyping the system and proving properties of both the policy and the topology. In fact, our constraint mechanisms allow to perform controls on the relations that are somehow stronger than just a type checking.

To be more precise, let us observe that, defining the actualisation of the model [Bar99] within our approach means **to insert into the theories the actual information in terms of facts and rules, if necessary.** For example, consider the theories *port_base, protocol_base, service_names* and *service_base* as defined above. To give the actual model means to make the following insertions, respectively:

- *port_base*
  Facts: **port(22).**
  port(25).
  port(8).
  port(0).
  ...

- *protocol_base*
  Facts: **protocol(tcp(nil)).**
  Rules: **protocol(tcp(X))←port(X).**
  protocol(icmp(X,Y)←port(X)&port(Y).
  ...

- *service_names*
  Facts: **service_name(smtp).**
  service_name (ssh).
  service_name (ping).
  service_name (https).
  service_name (all_tcp).

- *service_base*

  Facts: **service(smtp,tcp(25)).**
  service(ssh,tcp(22)).
  service(ping,icmp(8,0)).
  service(https,tcp(443)).
  service(all_tcp,tcp(nil)).

While the administrator implements the actual model, i.e. he inserts the actual information about the firewall as above, the system checks that the inserted knowledge is correct wrt the constraints. This, in general, means that the knowledge is checked against the types. Since there is no distinction, in our system, between types and other properties, constraints are, in general, used to define and prove properties of relations.

In the above case, for example, the system checks, for all facts in the *service_base*, their correctness wrt the constraint:

service(X,Y)>>service_name(X)&protocol(Y).

and since the protocol "tcp(443)" is not a valid one (it cannot be deduced from facts in the *propocol_base* since in the *port_base* port(443) was not defined) then "service(https, tc(443))" is not valid either.

As an other example, consider the constraint defined in the theory *service_group*:

service_group(X,Y) >> service_group_name(X) & service(Y).

The second argument of service_group needs to be a valid service, not just a service name as it is defined in the [Bar99] paper. This means that:

a) One can give theories that define the universe of names for ports, protocols, services, etc., then define constraints that use those theories to define the type of relations.
b) We can also put stronger conditions, such that not only there has to be a name for that service, but it also has to be a valid name, i.e. we can change the name universe when we want or put other conditions on the names and still ask for a service_group to be defined on service names which are legal. Thus we have given a different definition which is stronger than just a type definition.

The information that is not valid in the cases we have discussed above cannot be deduced. That is, there is no way that anyone can use that service(https,tcp(443)) and a service_group on it. In this sense, constraint checking works as a strange kind of type checking: instead of alerting for a type mistake it simply does not allow the use of that information. In other words, the incorrect information is ignored as if it was never given to the system.

This example also allows us to better explain the difference among the two forms of constraints. If the constraint on the service_base was given in the following "check" form:

service(X,Y)==>service_name(X)&protocol(Y).

Then the information service(https, tc(443)) could be deduced and thus used in other relations, but then running a constraints checking command in the Query window would give "NO" as answer, while it would have given: "Check OK" in the previous version.

These two different possible constraint definitions permit different development strategies:

- in the first case the constraints are always guaranteed to be satisfied (*IC* form). This means that the pieces of knowledge that violate the constraints cannot be deduced, i.e. that knowledge is not available;
- the other case (*check* form) allows developing pieces of knowledge that violate the constraints but still is available, although a check for consistency would fail.

When developing an application, the second form of constraints may be useful, allowing for freedom of definition and allowing for consistency checking every now and then. Once the application is developed, the first form of constraint may be useful to "consolidate" the application and make sure that every future use of it will not cause inconsistency.

Note that the importing mechanism by means of the predicate *theory*, permits to decide at run-time which service is allowed and thus forbid the others.

Once the theory that constitutes the model has been defined, we need to concentrate on the next problem that is faced in [Bar99]. This is concerned with the rule-base generation for a "generic firewall" that uses an ordered rule list as most firewalls do and that disallows whatever is not explicitly allowed.

We have not yet defined the transaction to produce the rule-base, although we believe it should be quite straightforward. The approach we are going to undertake is to use JEDBLOG transactions to generate the rules in the rule-base for the firewall.

4. CONCLUSIONS

We have presented some ideas on the implementation of a tool to define firewalls policies by means of a deductive database management system. The work has recently started and thus much more work needs to be done. It is the authors opinion that the firewall example, even if restricted, is meaningful considering that it is very easy to use, that it gives back a very concise definition of the firewall policy and, furthermore, it permits to prove properties of the definitions, at each step. This means to be able to make some reasoning on the policy and produce it with a higher level of confidence. Moreover, it is possible to prototype the policy and run it on examples.

Future work on the application we have presented will concern, besides defining the operational behaviour of this sketched implementation, to compare it with different firewall definitions and vendors requirements. In other words, we are going to study different security models and enforcing policies existing in the literature to evaluate the feasibility of more complex examples and the effectiveness of our approach in real systems.

5. BIBLIOGRAPHY

[AHV95] Abitebul S., Hull R., and Vianu V. (1995) *Foundations of Databases*, Addison Wesley.

[Asi94] Asirelli P., Di Grande D., Inverardi P. and Nicodemi F. (1994) Graphics by a logical Database Management System, *Journal of Visual Languages and Computing*. 5, 365-388.

[Bar99] Bartal Y., Mayer A., Nissim K. and Woll A. (1999) *Firmato*: A Novel Firewall Management Toolkit, in *Proc. of 1999 IEEE Symposium on Security and Privacy,* Oakland, California, May 9-12, 1999.

[BSJ97] Bertino E., Samarati P. and Jajodia S. (1997) An Extended Authorization Model for Relational Databases, *IEEE Transaction on Knowledge and Data Engineering*, 9(1), 85-100.

[Ches94] Cheswick W. R., Bellovin S. M. (1994) *Firewalls and Internet Security: repelling the wily hacker*, Addison Wesley.

[Ha96] Hare C., Siyan K. (1996) *Internet Security and Firewalls*, New Readers Publishing.

[JA96a] Jajodia S. (1996a) Database security and privacy, *ACM Computing Surveys*, 50th anniversary commemorative issue, 28(1).

[JA96b] Jajodia S. (1996b) Managing Security and Privacy of Information, *ACM Computing Surveys*, 28(4es), 129-131.

[LU95] Lu J. J., Moerkotte G., Schue J. and Subrahmanian V. S. (1995) Efficient Maintenance of Materialized Mediated Views, *SIGMOD Conference 1995*, 340-351.

[SICS95] Sicstus Prolog User's Guide, SICS, 1995.

[Ull88] Ullman J., (1988) *Principles of Database and Knowledge-Base Systems*, Computer Science Press.

APPENDIX

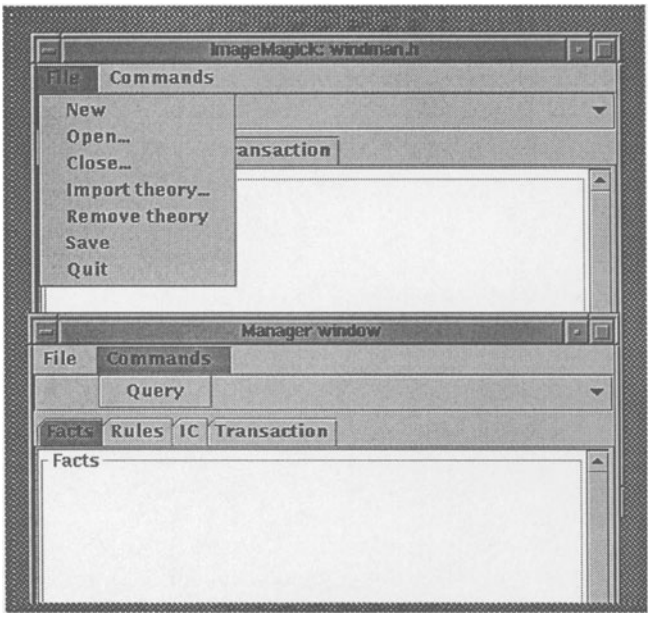

Figure 1. System Window Manager

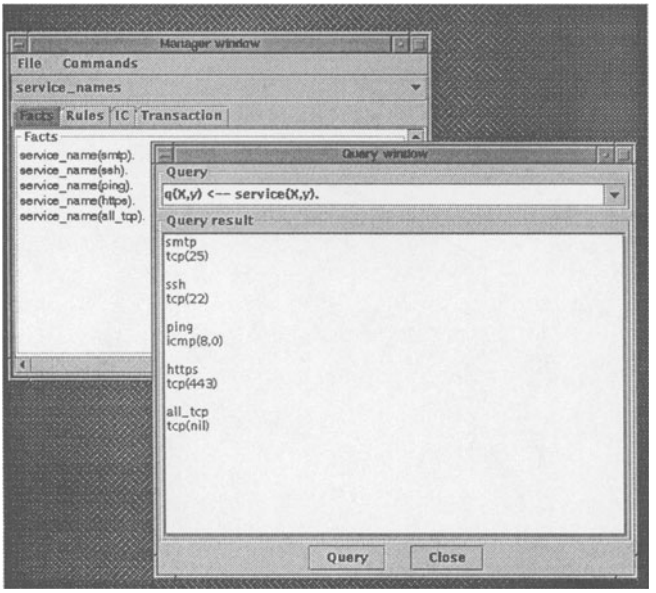

Figure 2. The Query Window