WEB SERVER AUTHORISATION WITH THE POLICYUPDATER ACCESS CONTROL SYSTEM

Vino Fernando Crescini
Yan Zhang
University of Western Sydney
Penrith South DC, NSW 1797, Australia
{jcrescin,yan}@cit.uws.edu.au

Weiyuan Wang
Smartlink Solutions Pty Limited
1 Raymond Place Epping, NSW 2121
ww@eg- 1.com

ABSTRACT

The PolicyUpdater system is a generic access control system that provides policy evaluations and dynamic policy updates. These functions are achieved by the use of a logic- based language to represent access control policies. In this paper, we discuss the underlying details of the PolicyUpdater system as well as the issues arising from its application to a web server access control system. Integrating the PolicyUpdater system with a web server provides a more flexible and expressive means of representing authorisation policies.

KEYWORDS

security issues, web software, logic programming, authentication

1. INTRODUCTION

The simplest method of web access control is to grant resource access only to registered users that have been authenticated by the system using a username and password pair. While this method is very effective and easy to implement, its simplistic nature and inflexibility prevents it from being used in applications where access control is needed between authenticated users or even non- registered (unauthenticated) users.

Another common form of access control allows conditional policies to be defined. Such conditions include user authentication, client hostname, time of day, etc. Access control lists may be formed by grouping together rules, which are grant or deny actions associated with one or more conditions. While this method is provided as a standard core feature by most web servers, it is only capable of modelling simple access control policies.

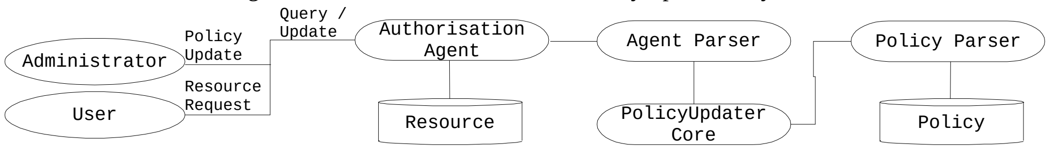
Recent advances in the broader access control field have produced a number of different approaches to logic- based access control systems. Bertino, et. al. [4] proposed such a system based on ordered logic with ordered domains. Jajodia, et. al. [7] on the other hand, proposed a general access control framework that features handling of multiple policies. Another important work is the system proposed by Bai and Varadharajan [2, 3]. Their system's key characteristic is the ability to dynamically update the otherwise static policy base. These systems, effective as they are, lack the details necessary to address the issues involved in the implementation of such a system to be used in a web server access control application. The *Policy Description Language*, or *PDL*, developed by Lobo, et. al. [9], is a language for representing event and action oriented generic policies. *PDL* is later extended by Chomicki, et. al. [6] to include *policy monitors* which, in effect, are policy constraints. Bertino, et. al. [5], again took *PDL* a step further by extending *policy monitors* to allow users to express preferred constraints. While these generic languages are expressive enough to be used for web server access control systems, systems built for such languages will not have the ability to dynamically update the policies.

To overcome these limitations, we propose the general purpose PolicyUpdater access control system, which, with its own authorisation language, provides a formal logic-based representation of policies with default propositions, a mechanism to conditionally and dynamically perform a sequence of policy updates, and a means of evaluating queries against the policy base.

2. POLICYUPDATER SYSTEM

The PolicyUpdater system is a generic access control system designed to be used in a variety of applications. The key feature of the PolicyUpdater system is that policies are stored and evaluated as logic programs. By using this approach, the PolicyUpdater system provides a means to allow policies to be dynamically and conditionally updated. Another feature of the PolicyUpdater system is that it allows policies to be defined with conditional constraints and default rules.

Figure 1. Overview of the core PolicyUpdater System



As shown in Figure 1, the core PolicyUpdater system works with an external authorisation agent, which functions as an access control enforcer and at the same time provides an interface for administrators to allow policy updates to be performed. The policy parser is used by the system to read the policy definition file, while the agent parser is responsible for all interactions with an authorisation agent. At this stage, it is important to note that the enforcer agent and the two parsers, although part of the entire access control system, are separate from the core PolicyUpdater system.

2.1 Language L

Language L is a first-order logic language that represents a policy base for the authorisation system. Two key features of the language are: (1) provides a means to conditionally and dynamically update the existing policy base and (2) provides a mechanism by which queries may be evaluated from the updated policy base. The language is composed of the following elements:

- **Entities.** The language uses three entity types: `subjects` (e.g. user, administrator), `access rights` (e.g. read, write), and `objects` (file, directory), with each type being a singular or group entity.
- **Atoms, Facts & Expressions.** Atoms provide the language a means to bind subjects, access rights and objects together. For example, `holds(s1, a1, o1)` states that subject `s1` holds access right `a1` for object `o1`. Because the language supports single and group entities, the following atoms are also defined: `memb(ss, sg)` which states that singular subject `ss` is a member of the subject group `sg` and `subst(og1, og2)` which means object group `og1` is a subset of the object group `og2`. Facts in the language are atoms or its negation. Expressions are facts or a conjunction of facts, e.g. `holds(s1, a1, o1) && !memb(s1, sg)`.

- **Initial Facts Rules.** Initial facts rules are expressions that hold before any policy updates are performed in the policy base.
- **Constraint Rules.** These rules are conditional expressions that must hold even after policy updates are performed. Constraint rules are in the form: `always exp1 implied by exp2 with absence exp3`, which asserts that expression `exp1` must hold if expression `exp2` is true and there is no evidence that expression `exp3` is true.
- **Policy Update Rules.** Policy Update rules are used to dynamically update the policy base. Syntactically, `update1() causes exp1 if exp2`, which asserts that if expression `exp2` holds, and update rule `update1` is applied to the current policy base state, then the expression `exp1` holds in the newly generated policy base state. Intuitively, facts that hold in the current state are carried over to the next state after an update applied, except those that are explicitly overridden by the policy update rule.
- **Policy Update Sequence Directives.** The language allows the policy base to be updated by a sequence of policy updates. These policy updates, which are defined by the policy update rules above, are applied to the policy base sequentially as defined by a policy update sequence list. The language provides a mechanism to manipulate this sequence list by the following directives: `seq add update1` appends the policy update rule `update1` to the list; `seq list` displays the update rules in the list; and `seq del n` removes the `n`'th update rule from the list.
- **Query Directives.** The language allows the input of authorisation queries to be evaluated with the following syntax: `query exp`, which causes the system to evaluate whether the expression `exp` is `true`, `false` or `unknown` when evaluated against the current state of the policy base.

2.2 Query Evaluation and Policy Updates

The PolicyUpdater system evaluates queries by translating language L policy base into a *normal logic program*, which can be evaluated using the *stable model semantics* [12]. As language L programs are already *extended logic programs*, the key step in the translation is to flatten the policy base states, i.e. Treat the policy base states as an additional atom property or parameter, then treat all policy updates as regular logical rules.

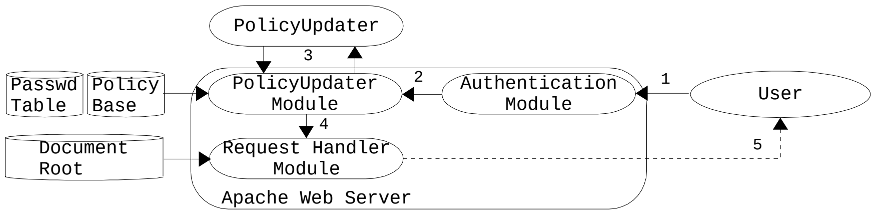
Once a language L program is translated into a normal logic program, the *smodels* library is used to generate a set of all the stable models *S* of the policy. PolicyUpdater can then perform query evaluations by checking to see if query expressions hold in set *S*.

3. WEB SERVER ACCESS CONTROL

A popular web server, Apache, has a simple built- in access control system, which is provided by its *mod_auth and mod_access* modules [1,8]. With this built- in access control system, Apache provides the standard HTTP *Basic* and *Digest* authentication schemes [11], as well as an authorisation system to enforce access control policies. Such policies may be defined as per- request rules with HTTP request methods (*GET*, *POST*, etc.) [10] as access rights; users and hosts as subjects and the resources in the server's document root as objects.

3.1 Integration of PolicyUpdater

Figure2. Apache Access Control Mechanism with PolicyUpdater



As shown in Figure 2, Apache's access control module is replaced by the PolicyUpdater module and its own policy base. The sole purpose of the PolicyUpdater module is to act as an interface between the web server and the core PolicyUpdater system. The system works as follows: as the server is started, the PolicyUpdater module initialises the core PolicyUpdater system by sending the policy base. When a client makes an arbitrary HTTP request for a resource from the server (1), the client (user) is authenticated against the password table by the built-in authentication module; once the client is properly authenticated (2) the request is transferred to the PolicyUpdater module, which in turn generates a language L query (3) from the request details, then sends the query to the core PolicyUpdater system for evaluation; if the query is successful and access control is granted, the original request is sent to the other request handlers of the web server (4) where the request is eventually honoured; then finally (5), the resource (or acknowledgment for HTTP requests other than GET) is sent back to the client. Optionally, the client can be an administrator who, after being authenticated, is presented with a special administrator interface by the module to allow the policy base to be updated.

The policy description in the policy base is written in another language, language L', which is syntactically and semantically similar to language L except for the lack of entity identifier definitions. Entity identifiers need not be explicitly defined in the policy definition. Subjects of the access control policies are the users. Since all users must first be authenticated, the password table used in authentication are be used to extract the list of subjects. Access rights of the policies are built in: they are the HTTP request methods as defined by the HTTP 1.1 standard [10] (i.e. OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE and CONNECT). Objects are the resources (e.g. files) available in the server. Assuming that the document root is a hierarchy of directories files, each of these are mapped as a unique object of language L'.

4. CONCLUSION

In this paper, we have presented a logic based approach for access control in web servers. As we have shown, the core PolicyUpdater used in conjunction with an interface module that plugs into a web server provides a flexible policy base framework that provides dynamic and conditional updates and access control query request evaluations.

One possible future extension to this work is the integration of temporal logic to language L (and therefore language L') to allow time properties to be expressed in access control policies. Such extension will be useful in access control systems such as e-commerce applications where authorisations are granted or denied based on policies that are time dependent.

REFERENCES

[1] Apache Software Foundation, 2004. Authentication, Authorization and Access Control. *Apache HTTP Server 2.1 Documentation*, http://httpd.apache.org/docs-2.1/.

[2] Bai, Y., Varadharajan, V., 1999. On Formal Languages for Sequences of Authorization Transformations. In *Proceedings of Safety, Reliability and Security of Computer Systems*. Also in *Lecture Notes in Computer Science*, Vol. 1698, pp. 375-384. Springer-Verlag.

[3] Bai, Y., Varadharajan, V., 2003. On Transformation of Authorization Policies. In *Data and Knowledge Engineering*, Vol. 45, No. 3, pp. 333-357.

[4] Bertino, E., Buccafurri, F., et. al., 2000. A Logic-based Approach for Enforcing Access Control. In *Journal of Computer Security*, Vol. 8, No. 2-3, pp. 109-140, IOS Press.

[5] Bertino, E., Mileo A., et. al., 2003. Policy Monitoring with User-Preferences in PDL. In *Proceedings of IJCAI-03 Workshop for Nonmonotonic Reasoning, Action and Change*, pp. 37-44.

[6] Chomicki, J., Lobo, J., et. al., 2000. A Logic Programming Approach to Conflict Resolution in Policy Management. In *Proceedings of KR2000, 7th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 121-132, Kaufmann.

[7] Jajodia, S., Samarati, P., et. al., 2001. Flexible Support for Multiple Access Control Policies. In *ACM Transactions on Database Systems*, Vol. 29, No. 2, pp. 214-260, ACM.

[8] Laurie, B., Laurie, P., 2003. *Apache: The Definitive Guide* (3rd Edition). O'Reilly & Associates Inc.

[9] Lobo J., Bhatia R., et. al., 1999. A Policy Description Language. In *Proceedings of AAAI 16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence*, pp. 291-298, AAAI Press.

[10] Network Working Group, 1999. *Hypertext Transfer Protocol -- HTTP/1.1(RFC 2616).* The Internet Society, ftp://ftp.isi.edu/in-notes/rfc2616.txt.

[11] Network Working Group, 1999. *HTTP Authentication (RFC 2617).* The Internet Society, ftp://ftp.isi.edu/in-notes/rfc2617.txt.

[12] Simons, P., 1995. Efficient Implementation of the Stable Model Semantics for Normal Logic Programs. In *Research Report A35*, Helsinki University of Technology, Digital Systems Laboratory, Finland.