# Authorization as a Service provided by a Generic Policy Engine

**Eugen Bacic, Tony White**
**Texar Corp.**
**135-1101 Prince of Wales Dr.**
**Ottawa, Ontario, Canada K2C 3W7**
**{ebacic, twhite}@texar.com**

## Abstract

*Authorization of access to resources is typically viewed as being the responsibility of the applications that access specific resources. This leads to incoherent implementation of security within an organization as the protection of a resource becomes dependent upon the path through which it is accessed. This paper proposes that authorization be provided as a service that is utilized by applications that require security. An architecture is presented that supports fine grained access control wherein policies are associated with individual resources. The requirements for, and properties of, policies are described and several examples are provided.*

## 1. Introduction

Security currently and historically [4, 5, 11, 13, 14, 15, 16, 20] has been viewed as keeping malicious users out of critical computer systems. This view of restriction rather than sharing is becoming increasingly challenged as the desire for information sharing increases. In order to create such a system it is vital to view security in a radically different way, as a means of sharing information between authorized individuals [6, 7, 8, 14].

While the requirement to share information has been realized within applications, security has generally been provided by each element within the components making up the application. For example, web applications generally consist of web, application and database servers chained together; each server providing a point solution to the authorization problem. An application should not have to secure and protect a given resource; such a service should be provided independently of the application, much like an operating system provides services to access the file system, the user interface, the mouse, or the network.

The inclusion of security within applications becomes untenable with the introduction of distributed architectures, peer-to-peer computing and other networking initiatives where there is no guarantee of consistent platforms [1, 2, 3, 9, 17, 18]. Security at the application level has resulted in a plethora of security designs and solutions, few of which are interoperable [4, 5, 6, 7, 8, 24]. Security within workflow systems, for example, requires that state be used to answer the authorization question. While state may be passed from application to application, this clearly requires a degree of interoperability that exceeds current standards. This lack of interoperability has resulted in well documented security holes that have increased the reluctance of an organization adopting widespread resource sharing as a model either within or between enterprises.

The next section introduces key concepts. The remaining sections of this paper propose a new model for policy-based sharing and software architecture to realize it. The architecture is one in which authorization is recognized as a service and fine grained policy-based security is provided. Examples of policies are briefly described. The paper concludes with a discussion of the key advantages of the policy-based security architecture introduced.

## 2. Key Concepts

In this section we introduce the essential concepts of entity and policy used in the Generic Policy Engine.

### 2.1 Entities

In order to define what transpires within a system we must utilize a generic term to define the objects upon which a security system will operate. We define an *entity* as the unified security object against which all security related functions are performed. Both the base security application and any ancillary security relevant applications utilize entities in order to manage and maintain security attributes. Table 1 summarizes the basic attributes that each entity represents with respect to the actual resource being protected.

| | |
|---|---|
| **Entity Identifier** | A reference to the entity (i.e., resource, user, group, application or policy) being protected. |

| | |
|---|---|
| **Authentication** | The means by which authentication will or has happened, with the possible inclusion of information pertaining to the login session, its state, locale, strength, and mechanism used. |
| **Security Policy (Authorization)** | The security policy or codified business rules to be applied to protect this entity. Additional information, such as access control lists, roles, etc. can be obtained from external 3rd party repositories such as LDAP, X.500, operating systems, web servers, etc. |
| **Access Control Violation Policy** | A policy that is invoked should the security policy be evaluated and deny the mediation request. This policy might notify an administrator of an unauthorized access or cause the requestor to be logged out of the realm. |
| **Audit (Accountability)** | A history of events describing what has transpired to this entity, by whom, temporal information, the policy invoked, and its logical outcome. |
| **Entity State** | Information specific to this entity that defines particularly important state information, such as who last invoked this entity and when, what the most recent and relevant condition of access has been, iterative/invocation limitations, etc. |

**Table 1.** Entity Attributes

Thus, the entity provides the basic abstraction by which we can define security for a distributed, collaborative security system. It provides the handles, to the policy and to the entity state, so that complex rules can be uniformly enforced across any distributed networked environment.

The entity is meant to represent *all* resources of interest being secured. This includes security and access violation policies. By making policies *first class entities*, security of the generic policy engine can also be effectively provided. Management activity is also mediated through policy. This is something frequently overlooked in policy languages and policy-based systems.

## 2.2 Policy Language

The requirements for a policy language are that it be capable of reacting to, controlling and monitoring the system access event stream. Clearly, this goes beyond saying "yes or no" to a request to access a file. A policy language should not merely be passive, but should be capable of interacting with the system being secured to modify it; e.g. reconfiguring a web server if the policy determines that the server is under attack. Most security-related policy languages [26] seem to restrict themselves to the control problem. We believe that this limits their utility.

Much of the work in computer security has delved into verifiable code and mathematical models of policy, often relying on variations of lambda calculus [4, 10, 11].

The primary requirements for a policy language are: verifiability, small size, efficient execution, expressive functionality, dynamic and portable. The dynamic requirement is important as it must be possible to update a policy in real time without taking the policy engine offline. The expressiveness of the language is also important – policies should be concise.

The ability of security policies being able to fetch pertinent information from external (to the policy) information stores such as LDAP, X.500, Certificate Authorities, operating systems, web servers, databases, etc. allows the policy to properly determine whether to grant access or not and what to do about the access event. Clearly, standards-based interfaces available through APIs are required. The information retrieved need not be security related; it could be information on whether or not an individual is on vacation, what their physical location is, whether another individual is also logged on, and so on. In order to access certain entities it may be necessary to provide additional authentication and this, too, can be provided for in the policy via codified rules requesting additional authentication information.

### 2.2.1 Policy Mediation

Having defined an entity and policy, mediation need be no more complicated than evaluating:

*Mediate (activeEntity action passiveEntity env)*

Where the *activeEntity* is the entity requesting access, as represented by *action* upon the *passiveEntity*. It is the job of the mediation authority to determine whether or not such a relationship is allowed within the confines of the defined environment (*env*).

The importance of the mediation request is two-fold. First, it matches existing security policy modeling nomenclature regarding subject-object interaction and policy modeling. Second, it is can easily define iterative and reflective definitions. For

example, if a user, George, wishes to access a particular application which in turn wishes to access an information resource, the agents would request two separate mediations:

*Mediate (George execute Application env)*
*Mediate (Application open Resource env)*.

If state information is required in order to efficiently or more safely process any mediation request then it can be stored in the environment for later use. Using the above example, the environment would remember that it was George that initiated the application and that it was operating on his behalf. Therefore, if any access restrictions were placed on the resource as to which application and which user could access it, the information would be contained within the environment. Since mediations can be logically chained there is no restriction in the number or complexity of the mediation requests that can be handled.

### 2.2.2 Policy Language

The policy language we have implemented is based upon Scheme (a Lisp variant); a decision based upon its denotational semantics and small virtual machine. It is also familiar, being widely taught at university. While not a theoretical concern, it is important to choose languages that people understand. We have also created an efficient interface between Scheme and Java in order to take advantage of both programming paradigms. The ability to take advantage of the large number of Java standards and components, such as JNDI, JDBC and JMAPI has facilitated the creation of sophisticated policies that automate the sending of e-mail to administrators, access data bases for account information, and create custom audit reports.

An example of a policy implementing NTFS security is shown below. The nt-policy function is the entry point to the policy which states that if you are the system user return *true*. If you are editing permissions (as indicated by the isMeta flag) and you are the owner of the file, return *true*. Otherwise, if you or a group of which you are a member are denied access, return false. If you are not denied access, test whether you or a group of which you are a member are explicitly allowed access. The policy is concise; the equivalent policy within the operating system is probably 10's of thousands of lines of code.

```
(define (nt-policy subj action obj isMeta)
  (if (isSystem? subj)
    #t
    (if isMeta
      (owner subj)
      (if (denied (list subj) action obj)
        #f
        (allowed (list subj) action obj
        )
      )
    )
  )
)
```

When using this policy and testing across of the wide range of file system scenarios, we observed a potential hole in NT security. The scenario has to do with a file with _WXD permissions for User_1 in a directory with no access specifically for User_1.

Without the above policy, User_1 could append to the file but since he did not have read permission the content of the file was overwritten. Our assumption was that append required a read to determine where to append but since it could not read the file it just appended over the whole file. With the policy User_1 was denied access to the directory and therefore denied access to the file.

## 3. Centralized Policy-based Authorization

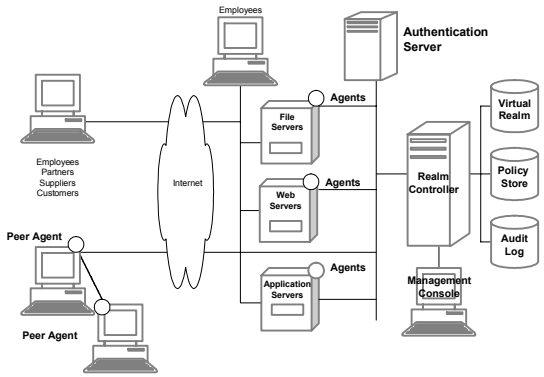


**Figure 1**. Policy Engine Architecture

Having described the abstractions necessary to implement coherent security, it is now possible to map the authentication, authorization and accountability requirements onto an implementation. Figure 1 contains elements of the disparate information systems that are deployed in enterprises today. These include file and web servers, along with peer-to-peer systems. This is clearly not an exhaustive list. Authorization is provided by the introduction of lightweight agents on the devices where information access occurs, a Realm Controller, a Virtual Realm database and Policy

persistent storage. Authentication is provided by the Authentication Server, which may be domain-based or providing capabilities such as single sign on. Accountability is provided by the Audit Log.

We will now describe essential functions of these components.

### 3.1 Agents

There are three types of agents that run on the device where information is accessed. First, lightweight agents are implemented at kernel level for file servers. These agents intercept all access to the file system at its lowest possible level. In the case of the Windows™ family of operating systems, this interception is via a file filter. In the case of UNIX operating systems, a pluggable module is implemented that wraps function calls that pertain to file system access. For web servers, a plug-in is implemented which is invoked during the servicing of an HTTP Get request. In all cases, the interception agents forward the request for access to the Realm Controller, blocking the request until the mediation result is returned. For performance reasons, results are cached, thereby avoiding potentially unnecessary network traffic and Realm Controller access. The second type of agent that runs on the device hosting the information being protected is the Service Agent. The function of the Service Agent is to act on behalf of the Realm Controller, issuing commands to control the host device; e.g. change permissions on a file or move it to a safe location if malicious attempts at access are detected. Commands are sent to the Service Agent through policy evaluation. The Service Agent is also policy-driven, with code being delivered on demand from the Realm Controller. For example, sending the command, *(change-permissions "C:/autoexec.bat" "tony" "r")*, results in the code for the policy *change-permissions* being requested from the Realm Controller. When received by the Service Agent, it is compiled and cached. Once again, control and security are centrally administered.

Finally, a Discovery Agent runs as part of the system initialization process in order to map all of the resources to be protected into the Virtual Realm.

### 3.2 Virtual Realm

The Virtual Realm is a database of all of the entities being protected. The Virtual Realm is organized around two basic structures: the entity and binary relationships between entities. Examples of binary relationships are: *parent(a,b),* where *a* represents a directory and *b* a file within it; *read(x,y),* where *x* is a user and *y* a file and *execute(u,v),* where *u* is a user and *v* an executable application. Agents interact with the Virtual Realm through the Realm Controller is order to manage the full lifecycle for an entity; i.e. creation, utilization, modification, and deletion. The audit responsibility falls with the domain of responsibility of the Realm Controller, not the agent.

### 3.3 Realm Controller

The Realm Controller provides authorization services for client agents. The Realm Controller answers the mediation question described earlier, doing so by interacting with the Virtual Realm. A typical scenario works as follows:

1. Agent A sends request, "Can Alice read Design.doc?" The agent also passes environmental information such as the permissions that Alice has to access Design.doc.
2. Lookup entity for Alice.
3. Lookup entity for Design.doc.
4. If resource-based access, retrieve policy for Design.doc.
5. If user-based access, retrieve policy for Alice.
6. Evaluate policy retrieved, store result.
7. If result is false and we are using resource-based access, retrieve access violation policy for Design.doc.
8. If result if false and we are using user-based access, retrieve access violation policy for Alice.
9. If result is false, evaluate retrieved access violation policy.
10. Store mediation request and its result in the Audit Log.
11. Return result to requesting agent.

When a policy is loaded, it is compiled and cached. Subsequently evaluations cause the retrieval and compilation step to be bypassed. The compilation step reduces the policy to Java byte code and the just-in-time compiler subsequently reduces it to machine code, where possible. When a policy is modified within the Policy Store, the Realm Controller is notified and the cached copy flushed.

The policy evaluation process is achieved using a reflective policy engine architecture, shown on the next page.

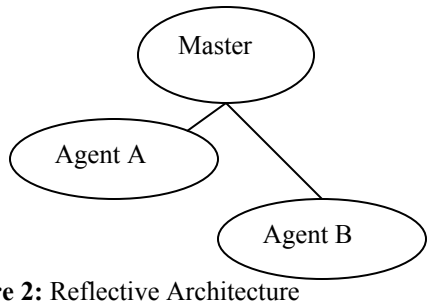


**Figure 2:** Reflective Architecture

The figure above shows two agent policy evaluators connected to a Master evaluator. A policy evaluation environment, or engine, is created for each agent connected to the Realm Controller. The Master evaluator contains policies that are shared by all agent evaluators. For example, if Agent A needs to evaluate the *subscription* policy, and it is defined in the Master, it will be evaluated there. If it is not present in the Master it will be loaded into the Agent A evaluator and evaluated locally. Policy namespaces are separated, and state is only shared where and when appropriate. Further, policy definitions cannot be overloaded. Attempts to define a policy, *p*, within, say, the Agent A evaluator, which is already defined within the Master, will be rejected. The Master evaluator is created when the Realm Controller starts up. Modifications to the policies stored within the Master evaluator can only be effected by management action. This reflective architecture ensures that policy modification and evaluation are tightly controlled.

A further refinement in the control of the policy language is that security is built into the language itself. When Agent A first authenticates itself with the Realm Controller it is given a security context, determined by the evaluation of the *authentication-context* policy associated with the agent entity. This context determines which functions can be evaluated by the agent. This is best illustrated with an example. Reducing the security context to a simple integer, Agent A might have the context 3, while Agent B has the context 2. Consider a hypothetical scenario where a policy is evaluated by both agents that would result in writing to a database; the API used being *write-record*. The *write-record* function requires a security context of 3. Hence, in this scenario, Agent A will succeed in writing a record to the database, while Agent B will fail.

This capability ensures that agents are unable to undertake actions that exceed their security context. Being implemented at function lookup time, it is very difficult to see how this would be circumvented. It is possible to modify the security context of an agent dynamically, both through policy evaluation and management activity.

The Realm Controller also deals with entity lifecycle events. For example, whenever a new file is created or destroyed, the Realm Controller is notified in order that an entity can be created or an existing entity removed from the Virtual Realm.

### 3.4 Policy Store and Language

The Policy Store is a repository for all policies that may be associated with entities. It is responsible for lifecycle maintenance of policies. The repository maintains an association between the policy name and its actual implementation. The actual policy is not stored with the entity, the name is. Both security and access violation policies are stored within the repository. The Policy Store is also responsible for notifying the Realm Controller when a policy changes in order that cached, compiled policies may be flushed.

Approximately 250 API calls have been implemented in order to hide the details of accessing the Virtual Realm and Audit Logs. Several interface functions have also been provided in order to allow access to the underlying services of the Realm Controller. A graphical user interface has been created for policy creation and testing. The details of this environment are beyond the scope of this paper.

### 3.5 Audit Log

The Audit Log is a repository for all mediation requests. Whenever a mediation request occurs, the request and its result are logged. Should an access violation policy be invoked, it too is logged. The Audit Log insulates the Realm Controller from knowing anything about the log details; i.e. whether it is a simple flat file or a relational database.

### 3.6 Management Console

The Management Console provides a management view onto the Realm Controller. Policy and Virtual Realm management functions are provided through a graphical interface. It is here that the importance of entity modeling is apparent, as modification of a policy is authorized by asking the question, “Can the Realm Administrator write to the *subscription* policy?” The mediation request, as before, reduces to a policy evaluation concerning the relationship between two entities. Similarly, whenever changes are being made to users within the Virtual Realm, the mediation question, “Can the Realm Administrator write to the user Alice?” is asked.

## 4. Conclusions

To date, many organizations have met their security concerns by implementing access prevention mechanisms such as firewalls, cryptography, and virtual private networks (VPNs). General access to host systems is provided based on the premise that once authenticated, users can be given full freedom to perform their duties. Existing security products protect only the perimeter creating islands of security and although each performs their individual tasks admirably, interoperability issues constantly arise. Solutions to the interoperability problems include special servers accessible to external partners, utilization of Web servers to store restricted views of "webified" information, and use of physical media to transmit critical information between partners.

With the emergence of the need for real-time collaboration [25], stopgap measures are no longer sufficient. Security officers and system administrators must now fulfill five security requirements:

- integration with existing products;
- uniform, ubiquitous security across the enterprise;
- controlled trust between Intranet and extranet systems;
- centralized and uniform controls; and,
- programmable policies that reflect the business rules.

With the emergence of distributed architectures for sharing the focus is now on *sharing,* not *restricting.* As advances in network computing continue the requirements to share information and other resources among physically separate locales will continue. Creating walls of preventative access due to restrictive rules will no longer suffice. Further exacerbating the problem is the fact that when two or more computer systems are linked, their security policies often clash and overall security actually diminishes. Until recently the act of restriction was sufficient. Now, with the desire to share comes the need to create a new security paradigm. The solution outlined in this paper provides a means of creating that paradigm.

This paper has proposed the need to remove authorization responsibilities from applications and provide a network-available service. The implications of this requirement have been explored, resulting in the key concepts of entity, virtual realm (or community) and policies associated with entities. An implementation has been described, which secures access with the enterprise and via the Web.

We believe that this generic policy engine has applications beyond those described here, and we are currently working on several extensions relating to the policy migration problem, and distributed policy evaluation. We look forward to communicating the results of this research in the near future.

## 5. References


[1] Abrams, Marshall D. and Michael V. Joyce. *On TCB Subsets and Trusted Object Management.* MITRE Technical Report #92W0000248, MITRE, McLean, Virginia. January 1993.

[2] Abrams, Marshall D. *Perspectives on General TCB Subsets,* Supplementary Reading Material for Tutorial #6, 9th Annual Computer Security Applications Conference, December 7, 1993.

[3] Abrams, Marshall D. et al. *A Generalized Framework for Access Control: An Informal Description.* MITRE Technical Report #MP-90W00043, MITRE, McLean, Virginia. August 1990.

[4] Amoroso, Edward G. *Fundamentals of Computer Security Technology,* Prentice-Hall, Englewood Cliffs, New Jersey, 1994.

[5] Anderson, Jim. *Computer Security Technology Planning Study,* ESD-TR-73-51, Volume I, AD-758, ESD/AFSC, Hanscom AFB, Bedford, Massachussetts. October 1972.

[6] Bacic, Eugen M., "Process Execution Controls as a Mechanism to Ensure Consistency," Fifth *Annual Computer Security Applications Conference*, December, 1989. Tucson, Arizona.

[7] Bacic, Eugen M., "Process Execution Controls: Revisited," Sixth *Annual Computer Security Applications Conference*, December, 1990. Tucson, Arizona.

[8] Bacic, Eugen M., "The Canadian Trusted Computer Product Evaluation Criteria," Sixth *Annual Computer Security Applications Conference*, December, 1990. Tucson, Arizona.

[9] Bacic, Eugen M. and Milan S. Kuchta, "Considerations in the Preparation of a Set of Availability Criteria," Third *Annual Canadian Computer Security Symposium*, May, 1991. Ottawa, Ontario.

[10] Bell, D. Elliott. "Concerning 'Modeling' of Computer Security," *Proceedings of the 1988 Symposium on Security and Privacy,* Oakland, California.

[11] Bell, David E. and L.J. LaPadula. *Secure Computer Systems: Mathematical Foundations,* ESD-TR-73-278, Volumes I, II, and III. The

MITRE Corporation, March, May, and December 1973.

[12] Biba, K.J. *Integrity Considertations for Secure Computer Systems,* ESD-TR-76-372, MTR-3153, The MITRE Corporation, Bedford Massachussetts. April 1977.

[13] Clark, David D. and D.R. Wilson. "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the 1987 Symposium on Security and Privacy.*

[14] Communications Security Establishment, *The Canadian Trusted Computer Product Evaluation Criteria.* Version 3.0e, January 1993.

[15] Gasser, Morrie. *Building a Secure Computer System,* Van Nostrand Reinhold, New York, New York, 1988.

[16] Gligor, Virgil, et al. "Design and Implementation of Secure Xenix," *IEEE Transactions on Software Engineering,* Volume 13, Number 2, February 1987.

[17] Hosmer, Hilary H. "Metapolicies I," *ACM SigSAC Special Workshop on Data Management Security and Privacy Standards,* San Antonio, Texas, December 1991.

[18] Hosmer, Hilary H., "Metapolicies II," *Proceedings of the 15th National Computer Security Conference,* October, 1992, Baltimore, Maryland.

[19] Hewlett-Packard Company. "Proposal for Discretionary Access Control," *IEEE P1003.6,* March 14, 1988, Austin, Texas.

[20] *The Common Criteria*. Harmonised Criteria of Canada, the United States, France, Germany, the Netherlands, and the United Kingdom. Version 2, 1999.

[21] Lee, E.S et al. *Composability of Trusted Systems, Reports 1 - 5*. Computer Systems Research Institute, University of Toronto.

[22] Lee, Theodore M.P. "Using Mandatory Integrity to Enforce "Commercial" Security," *Proceedings of the 1988 Symposium on Security and Privacy,* Oakland, California. pp. 140 - 146.

[23] Saltzer, J.H. "The Protection and Control of Information Sharing in Multics," *Communications of the ACM,* Volume 17, Number 7, July 1974.

[24] Katzke, Stuart W.; Ruthberg, Zella G., editors. *Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS).* Washington DC: NIST; January 1989; Special Pub 500-160. SN 003-003-02904-1.

[25] Bacic, E., Security as Collaborative Relationships, Collaborative Technologies Symposium 2002, San Antonio, Texas, January 2002.

[26] Policy 2001, Workshop on Policies for Distributed Systems and Networks, Bristol, 29-31 January 2001.