An Authorization Framework for Web-based Applications

David Jacobs
MITRE Corporation
USCINCPAC/J6
Bldg. 9, Room 235
Camp H. M. Smith, HI 96861-4029
djacobs@mitre.org

Abstract

As the web becomes the application platform of choice, supporting complex authorization policies is becoming increasingly difficult. This paper discusses the security challenges that web applications must face. It then proposes an authorization framework to help address those challenges. It provides some examples of how various policies can be implemented using the framework and how responsibility for managing them is divided up. Finally it discusses which issues require further study.

Keywords: authorization, access control, web application, security

---

1. Introduction

The web is increasingly becoming an application platform rather than a platform for simply delivering documents. In the realm of electronic commerce alone, IDC predicts that revenues will rocket past one trillion dollars by 2003 [1].

Since web servers were originally designed for document delivery, their authorization systems follow this history. This has left web application developers with the burden of developing their own authorization systems. Because of the huge time pressures on developers, these authorization schemes are usually minimally designed, inflexible and unreliable. Higher maintenance costs, difficulty in adapting new policies and security breaches are the results.

Authentication, the problem of identifying the user, has received the bulk of attention in web security research. In the commercial world this research has led to the development of public key infrastructures, the secure socket layer protocol (SSL), the secure electronic transaction protocol [2], smart cards, biometrics based schemes [3] and numerous other authentication technologies.

Authorization is the problem of identifying a user's privileges once their identity has been established. Extensive research has gone into examining the different kinds of policies that an authorization system should enforce [4] [5].

- Subject/object/action access control (e.g. traditional file systems)
- History based access control (e.g. separation of duty, Chinese wall, workflow)
- Time based access control (e.g. during working hours, until June 5th)

- Location based access control (e.g. from certain terminals)
- Trust based access control (e.g. how much do you trust the credentials)
- Input/output based access control (e.g. number of allowed requests, limits on response size)

There is a large body of access control work that has focused on relational databases and multilevel secure systems [6]. There have been attempts at developing generalized authorization languages from early work like Sandhu's Schematic Protection Model, to later works like the Authorization Specification Language by Jajodia, Samarati and Subrahmanian [7] and the generalized access control language (GACL) by Woo and Lam [8]. Additionally, efforts have been made to extend authorizations into time dimensions [9] and the workflow arena [10].

There have been a few attempts to address the needs of authorization services for distributed application environments. This includes Adage [11] which focused heavily on user interface issues. It also includes HP's Vault authorization server [12]. None of these projects have taken the task of solving the web application's security problems head on. Instead by staying more generic, they leave many of the web's problems untended.

Several open problems remain in the design and implementation of authorization services for web based applications. Especially in how to integrate authorization services into the web environment and designing a framework that is both flexible enough to enforce the policies of most applications and simple enough for day to day administration.

This paper is organized as follows. The second section gives an overview of the authorization challenges a web application faces. The third section describes the different parties that have a stake in creating and maintaining security policies. The fourth section details the authorization framework. The fifth section discusses how to meld this framework with web applications. The final section discusses remaining issues and future work. An example is included in the appendix that the paper will draw on from time to time.

2. Problems with the Web Application Environment

The web application environment is not particularly friendly to security issues. The following is a list of issues that cause the majority of web specific security difficulties from the server's perspective.

- Stateless HTTP protocol makes it difficult to maintain authentication and session state in a secure fashion [13] [14].
- User intolerance for additional complexity.
- Lack of controls on parameters and their contents in a web request.
- Code written elsewhere is often "reused'.
- CGI scripts do not usually map to business objects easily.
- Non standard application log formats make monitoring them burdensome.
- Building and maintenance of custom security frameworks is burdensome.
- Management of most web authorization frameworks is burdensome.

The first of these, maintaining authentication and session state are being aggressively pursued by

many others and will not be discussed in this paper. Users have time and again shown they will not tolerate additional inconvenience. So authorization schemes that require more work from the user (e.g. user certificates, plug-ins) are often non-starters in the Internet environment and hard sells in the Intranet environment. This issue then acts as a constraint on how to solve the remaining ones.

The Internet is full of security bulletins warning about security problems with this or that web program because the program forgot to check its parameters for malicious inputs which cause buffer overflows, access to unauthorized files and other nefarious problems [15] [16]. They make the faulty assumption that because the HTML control on the web page is limited to 20 characters they don't have to worry about receiving more. Unfortunately a malicious hacker can submit information to a web site completely bypassing the form and substitute in any values they desire.

The Perl language has a taint mode that prohibits system calls from using variables that have been passed in from the web without first being run through a regular expression checker. This helps protect Perl CGI scripts from system call hacks but not from hackers causing havoc in a database or other data repositories. Because so many Perl scripts written elsewhere are used in constructing web sites, retrofitting security into them is even more problematic.

When a developer is defining authorizations, they typically think in terms of business objects and transactions. Because a large number of business objects and/or methods can be contained in a single script, URLs do not map directly to business objects. Since most web servers currently require authorizations specified by URL, this too becomes problematic.

Making the problem even worse, many authorization decisions, especially workflow related ones, are very reliant on the state of a business object. Since most web servers typically don't have access to this information they have to leave the authorization decision in the hands of the application.

Many networks, in addition to firewalls have started adding intrusion detection systems (IDS) to improve security. Because of the complete lack of standards for application logging, IDSs typically have restricted themselves to monitoring only the basic services and those applications in wide spread use. This has led to misuse detection at the application level being virtually unknown and makes forensics very difficult when trying to piece together what happened to a compromised system.

All these problems have forced developers to create their own security frameworks with all the maintenance headaches that entail. The management of access controls is also not well defined in these home grown environments. This means the developer also has to create the environment to manage it as well. This can add up to a lot of work both in terms of development and managing the access control system for each application.

Lastly, there are a number of constraints that an authorization system for web applications must abide by. Web applications often employ a number of different web technologies. They may use Java Server Pages, Perl CGI scripts, Active Server Pages or dozen other technologies to build their applications. These applications may be hosted on Windows NT or Unix and a number of different web servers. Any authorization system needs to be able to work across all these heterogeneous environments.

3. Roles for Managing Authorizations

The authorization framework presented in this paper divides the responsibility for maintaining security into four roles: developers, policy administrators, system administrators and end users. Many design decisions were made to enhance the support for these roles.

3.1 Developers

As noted earlier, developing the security framework currently falls squarely on the shoulders of the web application developer. The goal of this framework is to standardize the representation of security policies and to move their enforcement out of the application as much as possible. Because developers are the most intimately familiar with how information is stored and the different means of accessing it, they still have a substantial security role however. They need to encode the entry points that an application supports and the general rules for permission to those entry points. This allows the programmer to reinstitute programming by contract, where they assert what conditions must hold before the performance of an action. As an example, an expense report process would normally require validations of dates, money fields and enforcement of the signer being different from the recipient.

3.2 Policy Administrators

Most large corporations have employees whose focus is to develop checks and balances in corporate processes. They decide how many signatures are required for authorization and how much money the process can authorize. These are the policy administrators. During application development, they often work closely with developers to ensure that their policies can be enforced by the application. In this new framework they will be responsible for grouping the permissions defined by the developer into roles. It also allows them to take the generalized constraints in permissions and specialize them further within a Role. In this way they could create a manager role that has a signing permission with a $2,500 limit and a vice president role that has a signing permission with a $50,000 limit.

Having a consistent framework across applications reduces the learning curve the policy administrator must go through for each application and ensures greater flexibility in altering roles in the future. Current development methods often leave policy administrators at the mercy of developers to implement the changes they desire and since developers are pushed to focus on end user requirements first, they often have to wait a long time.

A second role of policy administrators is the audit function. Most are involved in auditing various practices to see how well their controls are working. Where they find problems they try to come up with corrective policies. By standardizing the policy enforcement mechanism auditing becomes more uniform across applications. This provides policy administrators more time for careful analysis instead of simple collection and processing of data.

3.3 System Administrators

Workplaces are very dynamic environments and it falls to system administrators to ensure that each user has access to the applications they need to perform their duties. With reorganizations at all levels of the corporation having become commonplace, being able to quickly change role groupings

is crucial. They are typically overworked and in desperate need of any simplification. This framework allows the system administrator to focus on how users are grouped and what roles those groups and individuals should have. This alleviates the need for them to have deep understandings of the underlying applications and analyze the permissions for subtle logic bugs.

3.4 End Users

Most users see applications as tools to accomplish their tasks. Anytime the tool gets in the way of accomplishing their task they are annoyed. Because checks and balances implemented by the policy administrators often entail extra work for a user, they are highly sensitized to additional work in the name of the corporate good. They want a minimum of fuss and a security system that gets out of their way. The current situation of multiple logins and varying security frameworks is extremely frustrating. This framework creates a more uniform interface across web applications to the end user.

4. Constructs for a Security Framework

To address the concerns of each of these groups the authorization framework was divided up into four constructs: permissions, roles, groups and users. Each construct has a set of attributes that are stored in separately managed tables that the authorization server accesses.

4.1 Permissions

A permission is the authority needed to perform an action. It houses the logic to check the validity of a request and whether to log a successful or unsuccessful authorization request. Because of the intimate knowledge required of the application, the developer is normally responsible for developing the permissions for each of their web accessible methods. Permissions as described here have many parallels to Varadharajan, Crall and Pato's entitlements [12]. Permissions have the following attributes:

- Name
- List of transactional parameters (i.e. those which come in from the web request)
- List of business object attributes (i.e. those which come from the internal state of a business object)
- Parameter Validation rules (type, range, mask, regex, enumerated list)
- Business object rules (can include user information but not role)
- Log on failure flag
- Log on success flag

Permissions are addressed by name. They have two lists of attributes that should be submitted as part of the request. The transactional parameters come from the web request and the business object attributes come from the application. Parameter validation rules reintroduce programming by contract to the web application environment. By specifying the type and range, enumeration, mask or regular expression for each transactional parameter, parameter validity can be formally checked before further processing. Examples of validation rules include, social security number must match the mask of "999-99-9999" and age must be an integer in the range from 0 to 150.

Business object rules are where more complex authorization concepts can be implemented. These rules are a collection of implicitly ANDed logical expressions which use comparative operators (=, <>, <, <=, >, >=), logical operators (and, or, not) and the transactional and business attributes along with any desired constants for their formation. Normal precedence rules and grouping using parenthesis apply. In the case of an expense report you might use a business object rule to enforce the separation of duty between the creator of the expense report and the signer. The signer identity would be one of the transactional parameters and the creator's identity would be one of the business object's attributes.

By keeping relevant state information in the object, the authorization system is relieved from having to determine what history to maintain and for how long to support workflow and separation of duty types of authorization issues. This has the side benefit of providing nice encapsulation. The largest disadvantage is that a large change in workflow could result in having to change what information is stored in the business object. Also rules that require cross object information are hard to implement in this model. For example, to prevent mutual signing of expense reports requires knowledge from other expense reports when determining who can sign this one.

From an expressiveness standpoint, the alternative of maintaining a history outside of the object is much more powerful. Unfortunately its implementation quickly becomes problematic trying to match history to object instances and determining which and how much history to keep. For example, if Mary creates an expense report and then wants to sign an expense report the system needs to know the object identifiers (OID) of the expense reports to determine if they are the same. To make it even harder, this OID needs to be long lasting.

The log on failure and log on success flags determine which events are to be logged. A logging event causes the logging of all the business object and transactional attributes, along with other context information such as the username, datetime, source IP address and URL. This enforces a standardization of log files, which will simplify enhancing intrusion detection systems and audit analysis systems for misuse detection.

4.2 Roles

Attributes:

- Name
- Inherit from Roles
- List of permissions with associated rules.

Roles are collections of permissions. They have a name, a list of associated permissions and can inherit from multiple other roles. Lastly, each associated permission a role contains can be further restricted by associating additional business object rules to it. If a parent role already has the permission, these additional rules will replace any additional rules specified in the parent for that role. These additional rules are added to the list of business object rules the permission already contains.

Returning to our expense report example, let us assume we have the roles of signor, manager, sales manager, each of which inherits from the former and the role of vice president which inherits from the manager role. We wish for managers to have signing authority for up to $2,500 and VPs up to

$50,000. We could accomplish this by giving the signor or manager role the signing permission with the additional business rule of `amount <= 2500' and giving the VP role the signing permission with the additional business rule of `amount <= 50000. Notice that since the VP role has the signing permission the inherited signing permission with the lower limit is not used.

Roles would typically be managed by a policy administrator, developer or both depending on the size of the system. It is of interest that the policy administrator by controlling the granularity of roles can control how much flexibility system administrators have in reorganizing peoples permissions. If the policy administrator gives the permission to the role manager, then a system administrator cannot give the signing permission to someone without giving that person all the other permissions a manager gets. If however the policy administrator put the permission in a signor role that the manager role inherits from, then the system administrator has the flexibility to give a user that capability by itself.

4.3 Groups

Attributes:

- Name
- Inherit from Groups
- List of Roles

Groups are collections of roles. Like roles, they have a name and can inherit from multiple other groups. They also have a list of associated roles. A child group inherits all the roles associated with its parents. There is one special anonymous group that all users automatically belong to. System administrators and/or help desk personnel normally maintain groups. Groups provide administrators the flexibility on how roles should be grouped and assigned to people. As long as roles are fine grained enough, duties can be easily juggled around allowing system administrators to quickly support reorganizations.

Without groups, when a new system came on line, its permissions would either have to be added to a new role or an existing one. If it were added to a new role then the system administrator would have to manually add the role to each user that required it. If it were added to an existing role, difficulties occur when a department wants to split the new responsibility from the old. They have to go back to the policy administrators to have them create a new role, which will then make more work for everyone in the company along with probably taking a while to implement since the policy administrator probably doesn't report directly to the department.

With groups, the policy administrators would always create new roles when new systems come on line (unless they specifically want a tight coupling). This then gives each department the flexibility on how they want to divvy up the responsibilities to their personnel. They can either add it to an existing group or create a new one.

In our expense report example, we might have a traditional department where the managers group is given the manager's role where they automatically inherit the signing permission. An innovative department might create a new group called fiscal aides which consists of people who have had certain fiscal training and give these folks the signor role. While the same thing could be accomplished purely with roles, because of the locus of control for roles, they cannot be as

responsive to department needs as groups can be.

4.4 Users

Attributes:

- User Identifier
- List of Groups with date/time limitations
- List of granted roles with date/time limitations
- List of denied roles with date/time limitations

Users are what map directly to the authentication system. Each user can be assigned to groups. These assignments can be time limited (i.e. from/to datetimes). Users can also be granted and denied roles directly. Like group associations these can be time limited as well. Time limitations allow administrators to assign a user to a group for a specified period.

For example, an employee might be made a member of the manager's group for a week while the normal manager is out. The manager may not want the employee to have access to the evaluation system however, and so has the system administrator add a deny to the Evaluator role to the user's record.

4.5 How evaluation occurs

First the system checks if it was given a user identity. If no user was given then the anonymous group is used to determine available roles. If the user is given then the user's roles are computed as follows. The roles from all of the currently active groups are gathered. Then this list of roles is added to and subtracted from by the user's active explicit grants and denies of roles. If a user has both an explicit grant and explicit deny for the same role then the one with the shorter expiration time left will override the other. An explicit deny of a role will always override a role granted by a group association. This results in a final list of roles the user currently has available.

Each role is then checked to see if it has the required permission. If it does, the permission is evaluated with the additional business rules in the role. If the permission evaluates positively then authorization succeeds. If the permission evaluation fails, then the next role is checked. If all roles have been checked without a positive permission evaluation then authorization fails.

5. Mapping to the Web

Up to this point, the security framework has focused on generic object oriented representations of actions. This paper will now address how to integrate the security framework into the web environment of URLs and CGI scripts.

Here is a cradle to grave description of a web transaction using this security framework. The web server receives a request. It consists of a URL and a set of name/value pairs and possibly a user name. The URL and parameter values are used to perform a lookup in a permission-mapping table. If the lookup fails, authorization is denied and the web server returns the default error page. If the lookup succeeds then the web server calls the authorization server with the permission name, the transactional parameters and the authenticated username.

The authorization server then collects the active roles the user has. Each role the user has is checked for the required permission. If no roles have the required permission then failure is returned. Each role containing the needed permission is then evaluated in turn. Permission evaluation will return success, failure or a transaction identifier if additional business object information is needed. If any roles evaluate to success, success is returned. If all the roles evaluate to failure then failure is returned. Otherwise a transaction identifier is returned to indicate further information is needed.

Permission evaluation performs the following. First it performs the parameter checks. If they fail, it returns failure. If they succeed then it checks to see if it also requires business object attributes to complete its authorization. If it doesn't, it returns success, otherwise it returns a transaction identifier.

If the web server receives failure, it redirects to the failure URL specified in the mapping. If the web server receives a success it calls the CGI script normally. If it receives a transaction identifier it calls the CGI script with the addition of the transaction identifier. The CGI script is then responsible for calling the authorization server with the business object attributes and the given transaction identifier. The authorization server then completes the processing and returns a success or failure. The application is responsible for deciding how to deal with authorization failures in this case.

For the above to work there needs to be a mapping table containing the name of the permission, the URL, the key parameters (name/value pairs - values can be don't care), and the failure URL. Matching occurs first on the URL and then on the most specific set of matching parameters. If there are two matching entries with equal specificity then an error is logged and access is denied.

Perm	URL	Parameters	Failure URL
ListObj	http://some.server/cgi.bin/obj.cgi		http://some.server/unauthorized.html
EditObj	http://some.server/cgi.bin/obj.cgi	Oid	http://some.server/cgi-bin/HandleError.cgi
SaveObj	http://some.server/cgi.bin/obj.cgi	Oid, action=save	http://some.server/cgi-bin/HandleError.cgi

Using this technique we can easily integrated into most web application platforms. It also keeps the implementation of security policies in a standardized framework. It allows simple permissions to be handled by the web server and minimizes the work needed to implement complex permissions in the application logic.

Of course if the web application framework has access to the business object attributes it can eliminate the need for the developer to insert calls to the authorization server. In addition if the application framework has a built in mapping of URLs to business objects this too can then be automated. One such environment is Zope [17].

6. Discussion

There are still a number of issues that need to be addresses. First and foremost is how the permissions should be maintained. Currently they are maintained by hand by people with special system access. To make this system complete it needs to have a GUI for at least the system administrator's job (i.e. group management. One solution would be to write a group management web application and use its own system to manage access to itself.

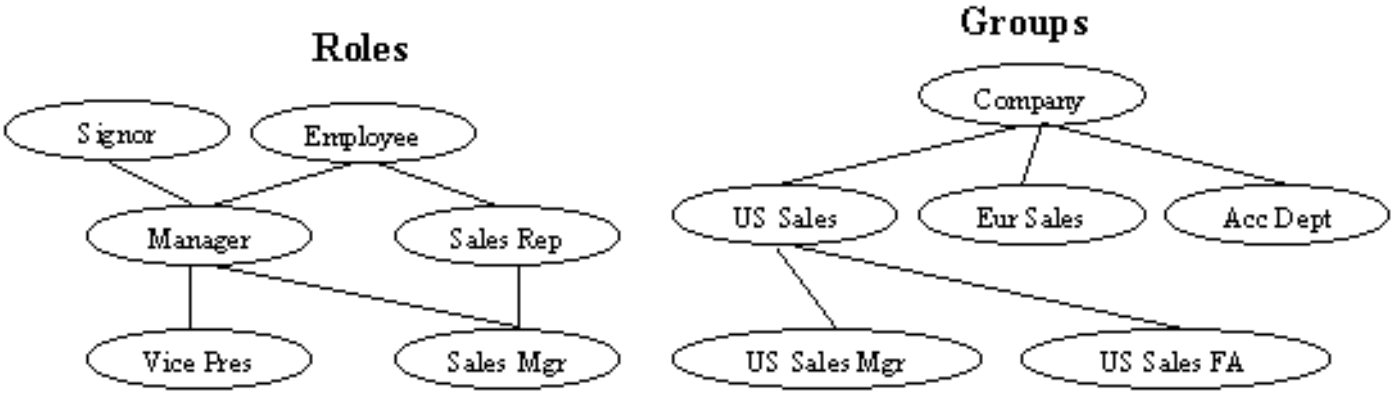
Currently the authorization service is simply added to the application in the form of a library. While this is fine for a proof of concept, it needs to be its own service to achieve greater cross platform applicability. This introduces the problem of needing a communication channel that must be secure.

Lastly, one area that has not been delved into deeply is the effect of transactions on Authorization systems. For example, in the typical bank situation, when making a withdrawal, the object must be locked, so that if a second withdrawal is attempted at the same time it will be blocked until the first one is complete. Otherwise an authorization decision could be made on object information that was in the process of changing. This was one of the main motivators for leaving state information in the object and not using a history mechanism. The object is in the best situation to address its transactional needs.

Appendix - Expense Report Example

Throughout this paper examples are given relating to an expense report application. Here, that example is presented is greater depth. The idea is that the application should enforce the following characteristics. Employees should be able to create and edit their own expense reports. Managers should be able to review and sign expense reports up to $2,500 that they didn't create. Vice Presidents should be able to review and sign expense reports up to $50000 that they didn't create. Expense reports can't be created for periods longer than a year ago. Expense reports can't be signed if submitted more than 3 months ago. Log all payments made by the system. Log all fail authorization attempts.

- Permissions - Create, Edit, Sign, Pay
- Roles - Employee, Signor, Manager, Vice President, Sales Rep, Sales Mgr
- Groups - Employees, US Sales, US Sales Mgr, US Sales Fiscal Aide, Europe Sales, Accounting
- Department



PERMISSIONS

Create

- Transaction Attributes
  - CreatorId - valid user
  - PeriodFrom - date
  - PeriodTo - date
  - Amount - range 1..50000
- Log on fail: True
- Log on success: False
- Rules
  - PeriodTo <= today
  - PeriodFrom >= Today - year
  - PeriodFrom <= PeriodTo

Edit

- Transaction Attributes
  - EditorId - valid user
- Business Attributes
  - CreatorId
- Log on fail: True
- Log on success: False
- Rules
  - CreatorId = EditorId

Sign

- Transaction Attributes
  - SignorId - valid user
  - DateSigned - date
- Business Attributes
  - CreatorId
  - PeriodFrom
  - PeriodTo
- Log on fail: True
- Log on success: False
- Rules
  - SignorId <> CreatorId
  - DateSigned <= Today
  - DateSigned >= PeriodTo
  - DateSigned - PeriodTo < 3months

Pay

- Transaction Attributes
  - PaymentDate - date
- Business Attributes
  - SignorId
  - DateSigned
- Log on fail: True
- Log on success: True
- Rules
  - PayorId <> SignorId
  - PayorId <> CreatorId
  - PaymentDate >= DateSigned
  - PaymentDate - DateSigned < 3months

ROLES

Employee

- Parents: None
- Permissions
  - Create
  - Edit

Signor

- Parents: None
- Permissions
  - Sign
  - Amount <= $2500

Manager

- Parents
  - Employees
  - Signor
- Permissions: None

Vice President

- Parents
  - Manager
- Permissions
  - Sign
  - Amount <= $50000

Accounting

- Parents: None
- Permissions
  - Pay

GROUPS

Employees

- Parents: None
- Roles
  - Employee

US Sales

- Parents
  - Employees
- Roles: None

US Sales Managers

- Parents:
  - US Sales
- Roles
  - Managers

US Sales Vice Presidents

- Parents
  - US Sales Managers
- Roles
  - Vice President

US Sales Fiscal Aid

- Parents
  - US Sales
- Roles
  - Signor

Accounting

- Parents
  - Employees
- Roles
  - Pay

USERS

Mary

- Groups
  - Employees
  - US Sales Mgr (from June 15, 1999 to June 30, 1999)
- Roles
  - Deny: evaluation
  - Grant: new-system

References

[1]

C. Glasheen and J. Gantz, "The Global Market Forecast for Internet Usage and Commerce: Based on Internet Commerce Market Model, Version 5," IDC W19262, June 1999 1999.

[2]

SET, "SET Secure Electronic Transaction LLC,". http://www.setco.org/. SET Secure Electronic Transaction LLC, 1999.

[3]

T. J. Alexandre, "Biometrics on smart cards: an approach to keyboard behavioral signature," presented at The 1996 2nd International Conference on Smart Card Research and Advanced Applications, Amsterdam, Neth, 1997.

[4]

D. D. Clark and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," presented at 1987 IEEE Symposium on Security and Privacy, 1987.

[5]

R. Simon and M. Zurko, "Separation of Duty in Role-Based Environments," presented at 10th Computer Security Foundations Workshop, Rockport, MA, 1997.

[6]

D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model," MITRE Corp, Bedford, MA M74-244, 1975 1975.

[7]

S. Jajodia, P. Samarati, and V. S. Subrahmanian, "A logical language for expressing authorizations," presented at 1997 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 1997.

[8]

T. Y. C. Woo and S. S. Lam, "Designing a distributed authorization service," presented at INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, 1998.

[9]

I. Merlo, E. Bertino, E. Ferrari, and G. Guerrini, "A temporal object-oriented data model with multiple granularities," presented at Sixth International Workshop on Temporal Representation and Reasoning, 1999.

[10]

R. K. Thomas and R. S. Sandhu, "Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management," presented at 11th IFIP WG11.3 Working Conference on Database Security, 1997.

[11]

M. E. Zurko, R. Simon, and T. Sanfilippo, "A user-centered, modular authorization service built on an RBAC foundation," presented at 1999 IEEE Symposium on Security and Privacy,

1999.

[12]

V. Varadharajan, C. Crall, and J. Pato, "Authorization in enterprise-wide distributed system: a practical design and application," presented at 14th Annual Computer Security Applications Conference, 1998.

[13]

A. Iyengar, "Dynamic argument embedding: preserving state on the world wide Web," IEEE Internet Computing, vol. 1, pp. 50-56, 1997.

[14]

J. S. Park and R. Sandhu, "RBAC on the Web by Secure Cookies," presented at IFIP WG 11.3, 1999.

[15]

J. McCarthy and E. Mills, "E-commerce sites exposing customer information," . http://www.infoworld.com/cgi-bin/displayStory.pl?990423.icinfo.htm. InfoWorld Electric, 1999.

[16]

C. Mellon, "CERT* Advisory CA-97.25.CGI_metachar," CERT Coordination Center, 1998.

[17]

B. Lloyd, "An Introduction to Zope,". http://www.devshed.com/Server_Side/Zope/Intro/. Developer Shed, 1999.

Vitae

David Jacobs is a senior technical staff member of the MITRE corporation. He received a M.S. in Computer Science from Rensselaer Polytechnic Institute in 1989 and is currently a Ph.D. candidate at Nova Southeastern University. He currently supports the DOD in their efforts to migrate Intranet applications to the web and to improve knowledge management. His research interests include web application infrastructure, security and knowledge management.