Using the Resource Access Decision Service to Improve the Application Security Layer

Cevat Göğebakan¹, Yenal Şener²

¹ Department of Computer Engineering, Middle East Technical University, Ankara, Turkey
² Cybersoft, Ankara, Turkey

Different solutions have been implemented for access control, and application security of enterprise applications. In this paper, we add a new layer to n-tier web application architectures, which use RAD service implementations to execute enterprise and application security policies. Proposed architecture enables applications to eliminate web application attacks including but not limited to those based on cross-site scripting, SQL injection, cookie poisoning, and most importantly session stealing.

1. Introduction

As more business requirements are embedded in web applications, the requirement of complex enterprise-level security policies for authorization, and handling these policies in application logic of web application reduce reusability and manageability of whole system [9]. Enterprise-level security policies have dynamic manner and can not be easily handled by existing general purpose security mechanisms. Implementations of Resource Access Decision (RAD) specification can also be used for authorization problem of complex web applications.

On the other hand, web applications suffer from broad kind of security attacks and the most dangerous of these attacks are the ones that target application-level vulnerabilities. Application-level web security refers to vulnerabilities inherent in the code of a web-application itself. Traditional firewall based security techniques fail to prevent these kinds of attacks and it is almost impossible to investigate whole source code of web application to find the vulnerable part. So it is much harder to detect the source of vulnerability and defend the system.

Executing enterprise-level security policies for requests that suffer from application vulnerability can probably result in error-prone access decisions. In order to decide on enterprise-level security policies, web requests must be free of
application-level security vulnerabilities. A correct access decision can be granted only if a request satisfies both “enterprise-level” and “application-level” security policies. Besides, combining “enterprise-level” and “application-level” security aspects in one layer in a transparent manner can result in great benefits such as reusability, manageability, scalability.

2. Access Control Problems in Enterprise Web Applications

The Internet is forcing enterprises to implement collaborative business and governmental solutions that integrate internal systems. To ensure security, these enterprise applications must implement complex access control rules that originate from both business logic and integration of business transactions. At this point access control rules become so called “enterprise-level security policies”. However as access control logic becomes closer to enterprise level, policy rules become more dynamic, more domain-specific, and more context dependent. These rules are enforced organization widely, and are usually embedded in application systems. As a result, the traditional approaches in execution of these rules are costly and error-prone.

Beznosov [2] defines domain (application)-specific factors in security decisions as follows; “An application-specific factor is a certain characteristic or property of an application’s resource, produced, modified and processed in the course of normal application execution and not for the sole purpose of a security policy decision.” According to this point of view, all business objects in enterprise applications can be a source of access policies with their underlying business rules. And collection of these domain-specific access policies defines the “enterprise-level security” policies. According to the separation of concerns principle [3] “enterprise-level security” policies should be handled by a uniform, fine-grained, and transparent way.

The employment of domain-specific factors in security decisions is not new; one of the earliest examples can be found in OSI access control framework [1]. From there on, various distributed application systems try to encapsulate domain-specific factors. These affords can be classified into three categories:

- **Middleware infrastructures**: Most common distributed application technologies, such as J2EE, .NET, COM and CORBA [2] integrated access control engines that manage object interactions. However, they all suffer from low expressiveness for controlling enterprise applications that execute business transactions and business services that require much more abstraction to be controlled by an object interaction access control.

- **Access control frameworks**: The major aim of access control frameworks [4-6] is to supply a centralized authorization engine, which is a uniform access control interface that asks for access permissions. Authorization engines are able to interpret and execute enterprise-policy rules that are defined by policy specification languages such as Ponder [8] and eXtensible Access Control Markup Language (XACML) [7]. These frameworks are powerful choices for expressing enterprise-
level security policies. However they are not transparent solutions. So they are error-prone and are hard to be organization-wide.

- **Commercial Access Managers:** Most of the commercial application server vendors [12-14] have access manager products and also other vendors [15][16] have products that can integrate into variety of application servers. The common strategy of these products is managing user identities and roles assigned to appropriate privileges based on RBAC [11]. However, RBAC fails to separate enforcement function and decision function that is needed to evaluate domain-specific access policies [2].

On the other hand, as organizations have been increasing their reliance on web applications and are confronted with steadily maturing network-layer defenses, attackers are presumably turning their attention to the application layer and the corresponding business applications that are being served. According to SANS Institute [17] statistics; from 1Q05 to 1Q06 there has been a 20% rise in the number of application-specific vulnerabilities identified, and over 50% of these are based on web applications. The statistics also show that over 80% of all malfunctions that emerged in the past year were focused on exploiting application-layer vulnerabilities.

The Open Web Application Security Project (OWASP) [18] is one of the foundations that is dedicated to find and classify possible web application attacks, and offers countermeasures for them. OWASP publishes “Top Ten Most Critical Web Application Security Vulnerabilities” list to inform the public about the most dangerous vulnerabilities. According the latest list [19], the most critical one is considered as invalidated input. Since request parameters is the only input source for web application, sniffing HTTP request and validating each parameter is the most critical step toward securing web applications.

The most common solution is web application firewalls. According to web application security consortium (WASC), a web application firewall is "An intermediary device, sitting between a web-client and a web server, analyzing OSI Layer-7 messages for violations in the programmed security policy. A web application firewall is used as a security device protecting the web server from attack." [20]. Nowadays there are both academic proposals for web application firewalls [21], as well as open-source [22] and commercial ones [23][24].

Access control decisions can easily be manipulated or even bypassed, if the demander application is vulnerable. In other words, to decide on enterprise-level security policies, web requests must be free of application-level security vulnerabilities. A correct access decision can only be granted if a request satisfies both “enterprise-level” and “application-level” security policies. Most web applications use different solutions to provide “enterprise-level” and “application-level” security. However, this solution reduces manageability, reusability and scalability of the whole system. RAD implementations can be extended to apply both enterprise-level and application-level security policies in a single solution so that complex access control rules that originate from enterprise-level policies, and security rules that originate from application-level policies can be evaluated at the same place.
3. Access Control and Security Solution Based On RAD

The Resource Access Decision (RAD) specification released by the Object Management Group (OMG) is a mechanism for obtaining authorization decisions and administrating access decision policies [10]. In Beznosov’s work this facility is rated as one of the best solutions that can be used by security-aware applications [2].

Our solution EYEKS ("Erişim, YEtkilendirme ve Kişiselleştirme Sistemi" in Turkish, meaning "Access, Authorization and Personalization System") uses the authorization engine, CSAAS, which is in fact a RAD implementation with RBAC [11] capabilities (presented in Akademik Bilişim Conference 2005 [9]). As mentioned before, RAD implementation does not force applications for authorization. It is the responsibility of the application to invoke authorization function and to take appropriate actions. However, EYEKS forces each request to be authorized and takes responsive actions to satisfy whole security policies in a transparent way.

Mapping Policies to RAD

RAD specification requires resources and their valid operations to be well defined. Resource can be any entity in the computer system, and an operation defines a valid procedure performed on any resource of the system. Every resource-operation pair can be associated with a number of “policies” that define access policies to do the requested operation on that resource. Access is granted only when that operation satisfies the associated policy rules on the specified resource. Policies are evaluated using attributes of an operation. These attributes can be dynamic (attribute value is evaluated at the time of the request) or static (parameters that are passed directly with the operation.) A policy grants or denies an access, based on the values of these attributes. A conceptual model of these relations is given in figure 1.

![Diagram](image)

Fig. 1.

3.1.1 Enterprise Policy Mapping

RAD specification suits well for security requirements of web applications. Resources and operations can be defined to fulfill the application requirements, and a
central access control layer can govern access rights of the whole web application. Request parameters can be mapped to attributes of security policies, and policy rules can be chained together to control access decisions of the web application. Determination of resources and operations is a critical step before designing an access control system that depends on RAD. They can be chosen according to the behavior of the web application. Two different approaches can be taken:

- Valid URL’s can be chosen as resources and HTTP methods (PUT, GET, POST, DELETE) can be chosen as valid operations on that resource.
- Web Forms can be chosen as resources and two conceptual operations can be defined as operation on that resource; VIEW and SUBMIT.

It is possible to go a step further in abstraction on this subject. If the web application is served on more than one web context, or the whole system consists of more than one web application, web applications or web contexts can be handled as resources and valid URLs can be chosen as operations. This approach is more suitable for web farms. Either way, all HTTP request parameters must be passed to policy evaluator to be evaluated at each policy chain. After successfully naming the resources and possible operations, the web application becomes directly mapped to RAD domain. At this step, business access rules, so called enterprise-level security policies, can be added to the system. An example is given in Table-1.

Table 1. An example of enterprise policy mapping.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Operation</th>
<th>Parameters</th>
<th>Policy</th>
</tr>
</thead>
</table>
| doeft.jsp | VIEW      | USERID
ACCOUNT_INFO          | EFTTimeCheckPolicy
VIEWAccountPolicy       |
|          | SUBMIT    | USERID
ACCOUNT_INFO
TRANS_ACC_INFO
TRANS_AMOUNT | EFTTimeCheckPolicy
VIEWAccountPolicy
TransAmountPolicy       |

According to Table 1, doeft.jsp is responsible for an EFT operation and the possible operations on the page are VIEW and SUBMIT. <doeft.jsp,VIEW> resource-operation pair can be linked with an enterprise security policy (EFTTimeCheckPolicy), which defines when a view operation is allowed (for example between working hours eg. 9 am-5 pm), and also with another enterprise security policy VIEWAccountPolicy, which checks whether the account really belongs to the specified user. <doeft.jsp,SUBMIT> pair can also be controlled by the same policies as <doeft.jsp,VIEW>, and additionally, can be linked with a security policy that checks whether the transfer could be allowed (TransAmountPolicy) (the policy that checks whether the transfer amount is less than the upper limit of user defined). As seen in the example, all enterprise-level security policies can be associated with any related resource-operation and they are reusable. These policies can be implemented by using policy evaluators defined in RAD specification [10] and
can be managed by RAD implementation. Policies can be added, removed or changed dynamically without altering the enterprise web application code.

3.1.2 Application Security Policy Mapping:

Nearly 80% of web application attacks are because of parameter manipulation, which are generally caused by data validation vulnerabilities. A careful centric design of data validation would free web application from these vulnerabilities. However checking against possible vulnerability exploits and validating input at every point of entry to web application is costly, error-prone, and unmanageable. These “application-level” security policies that eliminate web attack risks must be taken into consideration. An access control system, that depends on RAD specifications could possibly work for evaluating “application-level” security policies. Since it is guaranteed that all HTTP request parameters are passed to the RAD facility, the security officer can define DoEftViewSecurityPolicy on <doeft.jsp,VIEW> pair, and DoEftSubmitSecurityPolicy on <doeft.jsp,SUBMIT>, which define possible parameters and their expected values for each pair.

For general use, some policies that check for known security exploits has been already implemented and built into the system. Security officers can link these policies to any <resource, operation> pair in the application. These pre-implemented policies start with SECURITY tag, and target injection and input validation attacks. These policies can be extended or altered according to application security needs.

Carefully designed security policies, which define safe values for each parameter to be free from web vulnerabilities, can be written and forced to be used for every possible resource-operation pair. If any parameter for an operation on any web page is found to be malicious, that access would automatically be denied by the RAD system before actual execution.

System Architecture

The architecture of the proposed solution, which incorporates RAD facility to be used for access control, is shown in figure 2. A specific layer, so called “application security layer”, is created and placed in front end. Backend layers consist of real web applications and databases and have no direct access to the outside world. All communications from outside world to backend web application must be intercepted and authorized from application security layer. RAD service has placed in this layer and can only communicate with EYEKS. This layer is also responsible for handling session security. It eliminates the risk of information disclosure and implements safe authentication methods.

EYEKS is designed as a logically layered structure. All user requests are captured at the uppermost layer and are processed through the inner layers, and are then dispatched to the backend web applications. The logical layers are (from uppermost to innermost layer):

Fig. 2.

- **Request Listener Layer** is the entrance point of all client requests. This layer consists of two Java Servlets and can be configured to listen to any port by editing server configurations of the application server. The first servlet, i.e. the request listener servlet, is configured to hold root address ( / ), so that all requests to application security layer can be intercepted. All page and login requests are captured and handled by this servlet. The second servlet is a file upload servlet. All file upload operations are handled by this servlet.

- **Request Parser Layer** is where all requests are parsed to find out to which backend web application it refers, and also to find out to which scenario it belongs. Parameter-value pairs and backend context information (which web application does the request belongs to) are passed to the next layer.

- **Security Layer** is responsible for authentication and authorization. Authentication and authorization module checks if the user has enough credentials to make the coming request, by using a RAD implementation. Access to any page can be controlled by more complex policies like policies about allowed parameter-value pairs. Any request that does not satisfy these policies are denied.

- **Session Management Layer** is responsible for creating and handling user sessions. User sessions are created by the login scenario and are handled by the page request scenario. User sessions are stored in database or LDAP to be retrieved on every request. This layer also provides a unique encrypted token to be sent to client. The system provides these tokens that will be sent back by overwriting the HTML file generated by backend web applications. So the token mechanism for handling session security is fully transparent.

- **Request Dispatcher Layer** is the last layer of EYEKS structure. This layer consists of two modules and handles HTTP tunneling, requesting the original page from backend web application, modifying and returning it back to client. Log Manager is the other module which logs every access to the database or file. So detailed access logs of the whole system are handled by a single source.
System Details

EYEKS executes 4 main scenarios; Login, Page Request, File Upload and Logout. Login can be configured to work with hardware-token authentication, LDAP authentication, or password based authentication. Page Request and File Upload scenarios are fully controlled by RAD implementation and distributed session management semantics. Logout basically deletes user session.

Authorization strategy of EYEKS is based on combination of RBAC (Role Based Access Decision) and RAD (Resource Access Decision) standards, which are implemented by CSAAS [10]. Access policies written on resource and operation pairs give full force to the system on closing security breaches of the backend web applications. In these policies, all parameters can be checked for their allowed values. Also time based access policies can be implemented to control the permissions. For example, the web application can be closed to users having a specific role for some time period. IP based policies are other common types of policies that can be frequently used. Authentication module of EYEKS combines RAD with RBAC standard. So users can be assigned to system defined roles. And the roles a user assigned to define which permissions of RAD standard the user have. These roles can be hierarchically arranged. The authorization system of EYEKS can be fully configured by the system administrator according to the security needs of the system.

HTTP tunneling is an important strategy for securing a backend application. Client requests come only to application security layer, and then, the application security layer maps this request to the appropriate web application, and creates a new connection to the backend server. Firewall between application security layer and backend servers must be configured only to allow requests from application security layer and to deny everything else. Request dispatcher module, described in System Architecture section, is responsible for HTTP tunneling. The mapping between coming requests to backend web applications is described in a property file.

EYEKS offers a new more secure method for handling user sessions. The method not only considers security but also considers distribution execution so that web applications can be installed in clustered servers. Session management is handling by using encrypted token that hold user credentials such as user id and request sequence number. Using sequence information avoids session hijacking. Even if this encrypted token is hijacked by a malicious user, sending it back to the application security layer will not work. This token is inserted in every response of user request and it is granted that it will send back with the next request of user.

4. Test Results

EYEKS has been used in one of the biggest e-government projects of Turkey. The system became online on October 2004, and has been used for nearly 2.5 years. Table 2 gives statistics about how many business transactions, login and page request has
been done per month in last year respectively. The last column stands for the total request numbers on a peek day.

**Table 2. EYEKS Usage Statistics**

<table>
<thead>
<tr>
<th></th>
<th>Start</th>
<th>Average</th>
<th>Peek</th>
<th>Peek Day</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transactions</strong></td>
<td>34,534</td>
<td>1,856,324</td>
<td>3,124,236</td>
<td>649,024</td>
</tr>
<tr>
<td><strong>Login</strong></td>
<td>131,448</td>
<td>3,559,883</td>
<td>6,910,198</td>
<td>833,670</td>
</tr>
<tr>
<td><strong>Page Request</strong></td>
<td>975,481</td>
<td>39,046,279</td>
<td>63,808,771</td>
<td>4,128,295</td>
</tr>
</tbody>
</table>

Project is started with 13,466 registered users and by January 2007, 181,747 users have been registered. 4 resources (project consists of four different web applications), 67 operations, and 8 different roles are defined in RAD implementation. Enterprise security rules of the project were implemented by 11 user defined policies and by 210 permission mappings to policies.

EYEKS is installed on 3 servers. Two machines have four Solaris Ultra SPARC CPUs with 8GB RAM and one machine has two Solaris Ultra SPARC CPU with 2GB RAM. EYEKS scales and responses well though the heavy load as stated in table 2. Average peek CPU usage on peek days is 27%, where backend servers (8 servers) usage is 92%. On the other hand, artificial load test showed that under the load of 300 concurrent users, EYEKS payload is only 8%. EYEKS logs also showed that in last 3 months, total number of 865,327 requests, and 938,787 incorrect password tries per month were classified as malicious and were denied.

**5. Discussion & Conclusion**

In this paper, a fully implemented solution, EYEKS, to secure a web application is presented. Adding a new layer to web application, which deals with all security aspects, frees application developers from thinking security aspects of their applications. This approach also results in a more functional and structured system.

EYEKS is designed to eliminate the threats stated in problem definition section. Creating a separate layer, using HTTP tunneling and making service proxy for backend applications, eliminates information disclosure of information type of attacks such as common file query, link traversal, directory enumeration, and path truncation. On the other hand, parameter filtering and input validation mechanisms, implemented by application security policies that executes on RAD solution, eliminate invalidated input type of attacks such as cross site scripting, SQL injection, parameter passing, and forcing a parameter. User credentials are protected from session stealing attacks such as session hijacking and cookie tampering by a secure implementation of distributed session handling mechanisms.

A centralized view of security aspects enables the web application to be more manageable. Business-depended enterprise-level security policies and protection mechanisms (application-level security policies) can be added together to form a full
security policy chain that can be managed by RAD specification. RAD implementations offer high available, fine-grained, extensible, and dynamic access control mechanism which suits well for web application authorization needs.

As a result, adding an application security layer that controls organization-wide security policies, can give great benefits such as reusability, manageability, scalability to all kinds of web applications.

6. References

[16] WebDeamon, Integrative Security Management for Web-Based Enterprise Applications