
# Secure Coding.
# Practical steps to defend your web apps.


*Sponsored by Oracle*

SANS Institute Product Review:

# Demystifying External Authorization: Oracle Entitlements Server Product Review

*April 2012*

**A SANS Whitepaper**

***Written by: Tanya Baccam***

Real-World Scenarios ***PAGE 2***

Web Services Scenario ***PAGE 2***

Banking Application Scenario ***PAGE 3***

SharePoint Scenario ***PAGE 6***

Scenario Results ***PAGE 7***

# Introduction

Access control is a central requirement for many organizations in order to properly secure their data. Multiple compliance requirements dictate access controls must be in place to protect data; however, multiple challenges come into play when attempting to control access to protected data in applications.

First, access control must be flexible enough to allow dynamic changes based on business needs, and yet must not require substantial re-coding of an application as changes occur. Yet, in many applications, authorization decisions are hard coded into the application, making maintenance of the application complex.

Second, with a multitude of security technologies in use today, authorization controls need to be able to understand multiple systems, particularly those that store user data. A successful entitlement solution must have the capability to integrate with these technologies.

Finally, due to regulations and business requirements, very complex rules must often be applied. Any successful entitlement solution must be able to apply these rules in a flexible manner.

Oracle Entitlements Server (OES) meets these needs by operating outside of the applications—and without changing the applications themselves. Oracle OES allows the management of access policies based on business needs to be done as business demands dictate, rather than being held hostage by the limitations within their applications. OES provides the capability to apply privileges to specific users and under specific circumstances, thus controlling what they can do within the applications to which they have access. It also provides the capability to do additional auditing and provide governance for protected and/or regulated data.

# Real-World Scenarios

In order to better understand the role that OES plays among multiple applications, we set up and reviewed the server in three scenarios:

1. **Web service** – In order to get a basic understanding of OES, we used it to control access to a web service. We used Oracle Enterprise Gateway (OEG)[1] as a test client and mediator. OEG acted as a Policy Enforcement Point (PEP) that would block the web service request and ask for authorization. We chose a WebLogic web service to test the policies. Finally, the policies were stored and managed within OES Administrator Policy Server.

2. **Banking application** – The second scenario involved a banking application. We used a web browser to contact a traditional web application. After the user authenticated to the application, the web application made authorization decisions by making calls to OES as content was potentially being displayed to the user.

3. **SharePoint** – The final scenario leveraged SharePoint. OES provides a rich standards-based authorization service for SharePoint portals and document servers. We configured OES for SharePoint to insert a PEP into the SharePoint processing pipeline.

Throughout this paper, we examine some of the key components of OES that come into play for each of the scenarios and how it provided the required security and governance needs.

## Web Services Scenario

The web service, named **CustomerDataPortType**, was designed to extract customer data. In order to test OES, the first goal was to require authentication by integrating with the XML gateway (OEG). The Policy Studio management tool provided the capability to administer the policies and Oracle Enterprise Gateway settings needed to protect the web service. The service handler facility allowed the declaration of interfaces and abstract classes. Any calls to these interfaces or abstract classes were forwarded to the invocation handler for processing. In this case, we set the system so all requests from the client would go through an authentication policy that required a username and password. Using OES, we were able to add the authorization component without adding any code to the web service itself.

The second goal was to ensure only certain data was displayed back to the client based on who the user was. We created an interception point that called the OES Policy Decision Point (PDP) and filtered the data before returning it to the web service.

1 www.oracle.com/technetwork/middleware/id-mgmt/oeg-300773.html

Data was redacted based on the user and the matching authorization policies. Again, OES was able to filter the data returned without additional code being added to web service. Companies adopting Software-as-a-Service (SaaS) technologies commonly encounter these types of problems. In the past, web services were designed for intranets, with the presumption that data did not leave the company. But in the SaaS model, data is often shared with external companies hosting these services. For liability and compliance reasons, companies need to be very careful about data being sent out of the company. A product like OES allows companies to use rich semantic policy models to encrypt or redact data flowing to external service providers without the need to modify the web service.

To test the policies, we queried the data from the web service without authenticating using a generic Simple Object Access Protocol (SOAP) client. Without providing authentication, no data was returned. When a user with elevated access was authenticated, customer data (including credit card data) was returned. Finally, when the request was authenticated with a user with basic access, customer data was returned, but OES had removed the credit card information.

Based on the web service testing, we determined that OES can be used to add authentication requirements and to choose what data can be displayed to specific users. The policy was stored in the database, but was pushed out to the runtime PDP. When pushed to the PDP, the policy was stored in an encrypted format, locally on the gateway. This automates policy provisioning and life-cycle management by providing centralized policy administration with federated enforcement.

## Banking Application Scenario

OES can also provide fine-grained access to specific resources within the application. To further test the capabilities of limiting data that was returned, we reviewed OES in a banking scenario. We created three bank accounts for a user (Nicole) and three bank accounts for another user (Luke). Nicole was set up as the parent, and Luke was her son. In this scenario, we set up a fictional bank that had locations in the U.S. and Germany.

Table 1 provides a summary of the accounts that existed for test purposes.

| Owner | Beneficiary | Account Type | Guardian | Location | Balance |
|---|---|---|---|---|---|
| Luke | | Savings | Nicole | U.S./California | $1,000.00 |
| Luke | | Savings | Nicole | U.S./Texas | $2,000.00 |
| Luke | | Savings | Nicole | Germany | $3,000.00 |
| Nicole | Luke | Trust Fund | | U.S./California | $4,000.00 |
| Nicole | | Fixed Deposit | | Germany | $5,000.00 |
| Nicole | | Savings | | Germany | $6,000.00 |

*Table 1. Accounts Created for the Banking Application Scenario*

The policy model in this case was designed to be more complex to further test how to leverage OES for more complex regulatory requirements.

In order to set up OES to conduct this filtering, we set up the user accounts so access could be granted. Additionally, we ensured that the objects being protected were configured so the policies could be enforced against the objects. After the conditions were created, the policies could be applied. OES had "obligations" that provided the filter predicate to limit the data being returned. The following banking scenarios were tested with OES:

1. **Zero account balance check** – Because banks do not allow customers to complete a transaction in which their balance would go below zero (i.e., on overdraft), we built a policy that allowed funds to be withdrawn only when the final balance would be greater than zero.

2. **Time requirements** – Because time requirements often exist, we also created policies to meet these requirements. In this case, no withdrawal of the FD (Fixed Deposit or CD) was allowed prior to the maturity date.

3. **Location-based requirements** – Authorization is often restricted based on location. For example, in the U.S., banking regulations are different from state to state. We were able to create authorization policies that are tied to individual geographic locations.

4. **Hierarchical requirements** – Security artifacts frequently have interdependencies and are often related to each other. In our evaluation, we were able to structure secured artifacts into hierarchies and utilize underlying relationships to protect the required objects. For example, we mapped the user interface hierarchy directly into OES. This mapping allowed us to use the hierarchical relationships between web application, web pages and individual content within each page. If a user did not have access to a web page, the policies automatically denied access to individual content in the page. Vice versa, if a user did not have access to any content in a web page, OES automatically denied access to the main page.

5. **Reporting requirements** – In certain situations, accounts must be audited. In our test, fund transfers greater than $3,000 needed to be audited, and fund transfers greater than $10,000 needed to be reported to the Internal Revenue Service (IRS) for U.S. accounts.

6. **Additional regulatory and/or business impacts** – Additional restrictions needed to be applied for minors. In this case, we assumed minors could not withdraw funds from their accounts. Minor age was assumed to be 16 in Germany and 18 in the U.S. In the case of California, the minor age was assumed to be 21. Finally, we also assumed that the custodian could view a minor's accounts. Once the minor reached the age of maturity, the guardian could no longer access the accounts.

7. **Trust fund regulation** – Trust funds often have specific rules. The trust fund Nicole set up included the requirement that the funds could be accessed only after Luke turned 21.

For each of these scenarios, OES was able to create a policy that enforced these specific requirements without having to change the application code itself. Each scenario was designed to view a single policy or set of policies. The individual policies were then tested to ensure OES could enforce each of the requirements. A few of the policies are shown in Figure 1.

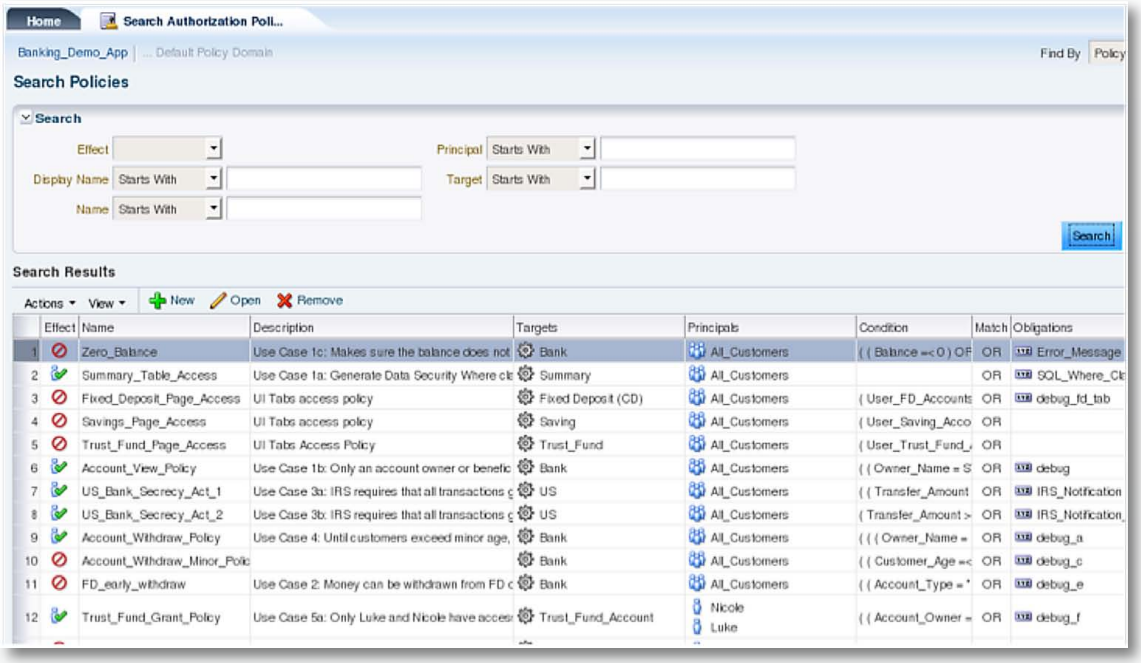


*Figure 1: Policies Created for Banking Requirements*

OES provides two facilities for bulk authorization. Several authorization requests can be pooled together into a single bulk request, which allows for efficient computation of hundreds of authorization requests at once. However, in some situations millions of authorizations need to be computed quickly and efficiently. For these use cases, OES policies can generate data filters that can be processed by a back-end data repository such as a database. In this evaluation we have used data filters in the form of SQL where clauses. When a user logs into the banking application, we may have to sift through millions of accounts to find the subset of accounts based on security requirements. For this we used OES data security policies to generate the right set of SQL filters that correctly identify the set of authorized bank accounts.

The policies were relatively easy to set up and enforce with OES. Roles can also be inherited in order to make enforcing access easier. Essentially, OES used fine-grained access control to apply a predicate to the WHERE clause to filter the data being returned from the database.

## SharePoint Scenario

The final scenario tested how OES could be used with SharePoint. When it comes to fine-grained access controls, SharePoint is difficult to customize and consistently enforce compliance reporting (which is mostly done manually) among large numbers of users. We originally authenticated through SharePoint; however, once authentication occurred, all authorization decisions went through OES. Specifically, in this case, OES reviewed the document attributes to determine whether specific users were authorized to access specific resources. OES performed entitlement functions as requested and delivered information relevant to the interface.

For SharePoint, we created an environment with the following security requirements, which cannot be done out of the box with SharePoint:

1. Only employees belonging to the engineering team should have access to engineering documents.
2. Only employees belonging to the sales team should have access to sales reports.
3. Only the person being reviewed and his or her immediate manager should be able to access performance review documents.
4. Only employees with a clearance level of 4 can access press release drafts. The company periodically issues press releases (PRs) containing information about company's revenue for the last quarter and future income projections. PR drafts are treated as confidential because they contain information that can impact the company's stock price.
5. After a PR release date, the PR becomes publicly accessible and anyone can view it.

Based on these requirements, we used OES to create, enforce and test the business requirements shown in Figure 2.

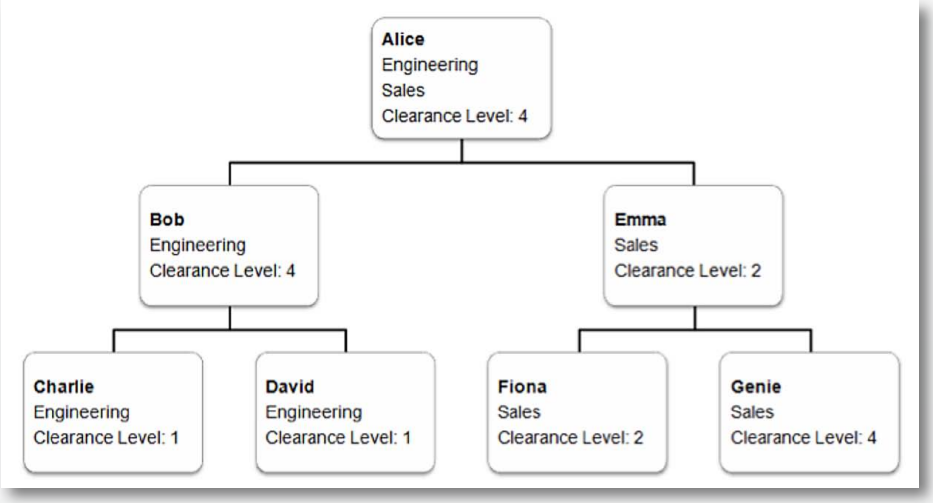


*Figure 2: Business Requirements Leveraged via OES*

For each of these requirements, OES enforced the business requirements, again without having to make any changes to SharePoint itself. Furthermore, SharePoint lacks the functionality to support these controls.

Companies create large numbers of documents daily. It is very difficult to create authorization policies specific to each document. OES uses a unique Attribute Based Access Control (ABAC) approach in which it is able to rely exclusively on identity and document attributes for authorization. This not only allows OES to scale across millions of documents, but also it allows for construction of policy models based on industry best practices.

OES plug-in (PEP) allows SharePoint sites to use rich standards-based policy models to secure documents and portals. Specifically, OES brings XACML (ABAC) and NIST RBAC support to the SharePoint ecosystem. Standards such as XACML address limitations with SharePoint's Claims Based Authorization.

## Scenario Results

Throughout this process, we successfully tested the following items:

- **Data redaction in web services** – OES allowed for data to be redacted for specific users. For example, as part of the web service testing, only certain users received the credit card data.
- **Fine-grained rules support** – OES allowed for the fine-grained access we needed for specific resources. For example, in the banking application, once a child was no longer a minor, the custodian could no longer view the child's account information, resulting in successful deauthorization.
- **Applicability for multiple industries** – OES was flexible in performing authorization for many different scenarios. For example, as part of the banking application, we were able to build policies around time requirements, age requirements, regulatory requirements and business requirements.
- **Application interaction** – OES provided the capabilities to provide authorization decisions for different types of applications. In this case, we tested "applications" including web applications, web services and SharePoint. OES was able to limit access in all cases based on business requirements.

# Conclusion

The ability to centrally manage access down to the specific resource level has, in the past, seemed unachievable beyond a system-by-system basis. An integrated tool that does not require changes to applications is a welcome improvement in administration and risk management. Oracle Entitlements Server (OES) made the process of controlling access easier—and more manageable across multiple applications and scenarios within those applications—with no retooling of applications required.

The key aspects we noticed during our evaluation were as follows:

**1)** OES allows for centralized policy management with federated enforcement. Authorization policies across the enterprise were managed from a single administration user interface and were centrally stored in a database. But the enforcement was done within the application or service.

**2)** Application and security life cycles can be separated by externalizing authorization. We were able to add new roles and entitlements to the application without modifying any application code.

**3)** Support for popular authorization standards such XACML and RBAC not only allows for flexible policy modeling, but also it helps companies to build out their enterprise applications based on standards and avoid vendor lock-in.

**4)** When building enterprise applications, audit is often an afterthought. By externalizing authorization, all access requests and security policy changes are automatically audited. This frees application developers from having to worry about audit and compliance.

Products such as OES not only secure the enterprise, but also reduce the burden on application development teams—which means companies investing in this technology will get security benefits and a strong ROI. Overall, we felt this technology was mature for broad industry adoption.

# About the Author

**Tanya Baccam** is a SANS senior instructor as well as a SANS courseware author. She is the current author for the SANS Security 509: Securing Oracle Databases course[2] and is the lead in a line of audit courses.[3] Tanya works for Baccam Consulting, where she provides many security consulting services for clients, including system audits, vulnerability and risk assessments, database audits, and web application audits. Today much of her time is spent on the security of databases and applications within organizations. Tanya has also played an integral role in developing multiple business applications. She currently holds the CPA, GCFW, GCIH, CISSP, CISM, CISA, and OCP DBA certifications.

## SANS would like to thank its sponsors:


2 www.sans.org/security-training/security-oracle-74-mid

3 http://it-audit.sans.org

Last Updated: July 26th, 2017

# Upcoming SANS App Sec Training


| | | | |
|---|---|---|---|
| **Community SANS Minneapolis DEV534** | **Minneapolis, MN** | **Aug 25, 2017 - Aug 28, 2017** | **Community SANS** |
| **Community SANS San Francisco DEV541** | **San Francisco, CA** | **Aug 28, 2017 - Aug 31, 2017** | **Community SANS** |
| **SANS Network Security 2017** | **Las Vegas, NV** | **Sep 10, 2017 - Sep 17, 2017** | **Live Event** |
| **Secure DevOps Summit & Training** | **Denver, CO** | **Oct 10, 2017 - Oct 17, 2017** | **Live Event** |
| **SANS Seattle 2017** | **Seattle, WA** | **Oct 30, 2017 - Nov 04, 2017** | **Live Event** |
| **SANS Cyber Defense Initiative 2017** | **Washington, DC** | **Dec 12, 2017 - Dec 19, 2017** | **Live Event** |
| **SANS Cyber Defense Initiative 2017 - DEV522: Defending Web Applications Security Essentials** | **Washington, DC** | **Dec 14, 2017 - Dec 19, 2017** | **vLive** |
| **SANS Security East 2018** | **New Orleans, LA** | **Jan 08, 2018 - Jan 13, 2018** | **Live Event** |
| **SANS San Francisco Spring 2018** | **San Francisco, CA** | **Mar 12, 2018 - Mar 17, 2018** | **Live Event** |
| **SANS OnDemand** | **Online** | **Anytime** | **Self Paced** |
| **SANS SelfStudy** | **Books & MP3s Only** | **Anytime** | **Self Paced** |