

## IDENTITY PROPAGATION IN N-TIER SYSTEMS

Anil Patel, Malcolm McRoberts and Melissa Crenshaw  
Harris Corporation  
Enterprise Architecture Core Technology Center  
Melbourne, FL  
August 6, 2009

### ABSTRACT

*While SOA promises great benefits in productivity and flexibility, the tools for securing these systems continue to lag behind. The ideal of SOA security is to provide trusted containers and frameworks that enforce policies established during deployment, and remove security logic and policy from application code completely. Standards such as WS-Security address some of the issues, but enterprise systems don't stop and start with web services. In an N-tier the user is authenticated at the client platform, and this authentication will ultimately determine access to resources in back-end data stores. The challenge is to create a framework for the end-to-end propagation of user credentials across N-tiers, which doesn't rely on custom security code within applications. This paper will describe a working prototype framework that propagates user credentials through web application, web service and database tiers, and applies label-based access control (LBAC) policies within the database. The paper will also outline known gaps in web and SOA standards, and directions for future work.*

### ARCHITECTURE GUIDELINES

Before discussing any specific mechanisms or technologies, we want to establish a set of common principles that guide software industry leaders as well as some that guide our own work as solution architects and integrators.

#### *Application Code Transparency*

A major tenet of middleware security is the removal of security specific logic from application code. The policies should be stored as data and enforced by framework code, or external components (e.g. firewalls). Where existing frameworks can't provide full transparency for application code, required security logic should be factored out to create extended frameworks. Adherence to this principle provides benefits in both security and agility. By isolating security logic to framework code, security vulnerabilities are also isolated to the framework code. Smaller code means less code to validate, and fewer defects (potential exploits). This is especially important if the system must go through a formal certification and accreditation (C&A)

process. Less code also means this code can be more easily optimized, and more thoroughly tested.

This also applies to the development staff. A small, highly trained team can maintain any security framework code [1], while application developers can focus on application functionality without the need for specialized knowledge and training on SOA security technologies.

Ultimately, this principle allows for a much more agile application development process, since application and business process code are no longer involved in the enforcement of security policies, and therefore don't require the same level of security inspection and testing.

#### *Policy-based Security for Risk Management*

There are two kinds of risk associated with security policies. The first risk is that a policy is incorrect or inadequate. The second risk is that a policy is not implemented or is implemented incorrectly. By removing policies from application and framework code, and creating explicit, separately managed policy definitions, it is possible to have a rigorously tested security framework while allowing the agency/enterprise to adjust their policies to balance security risks with operational flexibility.

#### *Use Native Policy Enforcement*

Where possible the system should make use of containers, frameworks and components that provide their own policy enforcement mechanisms (PEPs) that have often been formally accredited or are in the pipeline. Native PEPs are as close as possible to the resources they are protecting making them more efficient and more difficult to bypass. For example, Oracle Label Security (OLS) provides a native PEP for label-based access control of row-level data. When the native PEP is not adequate the approach is to augment rather than replace it. For high traffic systems, hardware accelerators such as DataPower may perform better than native PEPs. This may warrant an exception to this guideline.

#### *Manage Connections for Performance*

N-tier systems require connections between the various tiers. Establishing new connections can be orders of magnitude more costly than using an existing connection and can increase the overall level of risk. Two well known examples are JDBC connections and SSL connections.

The performance and scalability of N-tier systems is largely determined by the ability of their containers and frameworks to effectively manage connections (and other key resources) using methods such as pools and sessions. This can be a problem if the framework for connection management does not propagate identities.

#### *No High Privileged (Super) User*

A key principle in modern Information Assurance (IA) is to avoid creating shared accounts that have “super” privileges. If such an account is compromised the exposure is enormous. This applies to database users as well. Connection pooling requires use of a shared account, but this account must not have access to ANY protected data. This account should also not allow other user’s authorizations to be asserted without authentication.

#### *Product Independence*

Where established standards exist, components should be selected based on their compliance with these standards. Unfortunately, there are still many gaps in security standards for N-tier systems. Some vendors have filled major gaps within their own product lines, but these solutions lead to “vendor lock-in”. The goal is for the security architecture and application code to be portable and interoperable among most popular products (including Open Source). Framework code that augments product security features is likely to require customization.

#### *NEAT*

The solution should follow the NEAT principle (*Non-bypassable, Evaluatable, Always-invoked and Tamperproof*). [2]

### **WEB SERVICES SECURITY**

The emergence of Web services introduced a new paradigm for exchanging information across multiple domains using open industry standards and emerging standard based technologies. Applying enterprise security and establishing trust among producers and consumers has caused new challenges, some of which, such as identity propagation across multiple tiers remain unaddressed by traditional security methods.

The Web services security model attempts to mitigate most of the challenges with the combination of security standards and technologies – XML Encryption, Security Assertion Markup Language (SAML), XML Access Control Markup Language (XACML) and OASIS Web Services Security (WS-Security).

WS-Security is emerging as the industry de-facto standard for securing web services. The WS-Security specification contains SOAP extensions required to implement security requirements like, message

authentication, message integrity, message confidentiality and secure token propagation. This paper describes patterns and mechanisms that leverage WS-Security for credential propagation throughout the enterprise tiers – web application, web service and database.

### **ENTERPRISE N-TIER ARCHITECTURE**

An N-tier application architecture is characterized by the functional decomposition of applications, service components (which include service to service invocation) and database access. WS-Security provides a mechanism to bridge common security requirements across the tiers. Figure 1 shows the common components of n-tier application components:

- User/Client tier: Typically a service consumer.
- J2EE application server: Component that hosts services and either delegates or enforces the security policy.
- Database Tier: An external data service provider, typically a physical database that is either integrated into enterprise security infrastructure for managing policies or in most cases, standalone with its own security policy enforcement, as is in the case of Oracle Label Security to perform fine grained access to data based on user’s credentials.

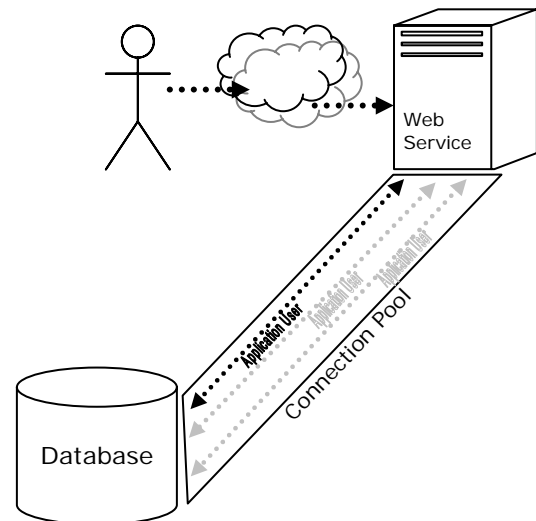


Figure 1 N-Tier Architecture

Each tier in a normal deployment, implements security enforcement using credentials passed in the request – an example would be a web service request with WS-Security containing SAML assertions [3]. Common security strategies for J2EE are:

#### *Application Server Tier - Security Strategies*

The J2EE platform allows establishing authentication in all application tiers and components. The J2EE

environment provides support for the following types of service strategies:

**Container-Based Authentication:** This is the standard service provided by J2EE server infrastructure. This allows J2EE environment to authenticate users for access to deployed applications.

**Application-Based Authentication:** This strategy relies on application programmatically retrieving the user credential and setting the security context that maps to the security principal in the container.

**Agent-Based Authentication:** J2EE application uses external security providers for authentications via agents.

Regardless of which security strategy is deployed, the request from the client application utilizes WS-Security with a SAML Assertion as shown in Figure 2.

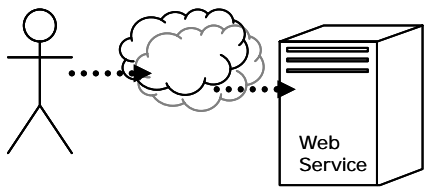


Figure 2 User and Application Tier

#### Security Assertion Markup Language - SAML

SAML provides an XML standards-based representation of security information that can be shared by application security domains. SAML Assertions [4] can be transferred cross domain between service consumers and providers.

Even though SAML does not provide underlying authentication mechanism, the Assertions can be used to exchange security information using SAML profiles [5].

#### WS Trust

WS-Trust is one of the many WS-\* specification related to the WS-Security extensions [6]. WS-Trust is an OASIS specification dealing with issuing, renewing and validating security tokens. It also specifies ways to broker trust relationship between security domains for facilitating secure message exchange.

#### Database Tier - Label-based Access Control (LBAC)

Major databases such as Oracle and IBM DB2 implement label based access control for retrieving/inserting identity aware data. In Oracle, the label security is implemented using Oracle's virtual private database [7][7] which acts as the policy enforcement and decision point. Users and roles are defined in the Oracle database and label policies are defined for each user. Oracle does provide extensibility to share the label security policies in an external repository such as a LDAP. Several common mechanisms for authenticating the users are

provided: Username and Password, X.509 certificates (being deprecated), and using distinguished name.

Data is accessed from the application using JDBC SQL queries. To perform an identity aware query from the application, the JDBC connection must be established using the user's database credential. This normally conflicts with JDBC connection pooling, since the connection is bound to a specific user and cannot be reused.

#### Current Framework – COTS/Vendor

Service providers deployed on COTS J2EE server (container) can be configured to utilize any of the security strategies mentioned earlier in this section. In case of application-based authentication, as the name implies, the application is responsible of extracting and propagating the credential in subsequent service calls.

In case of container-based authentication strategy, the container enforces the security. However, propagating the credential (part of the SAML assertion) to the application and subsequent tiers requires implementing vendor specific interfaces to extract the credential.

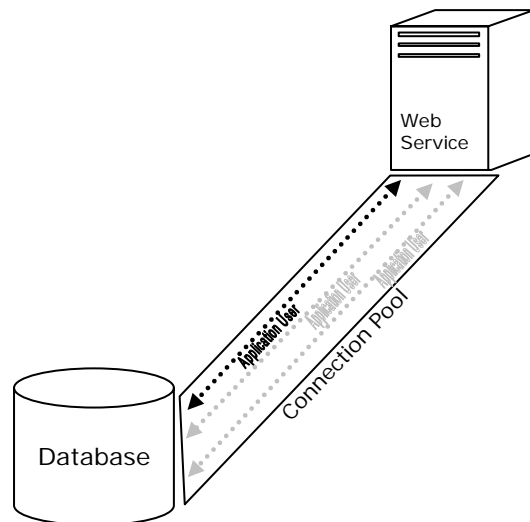


Figure 3 Database Tier Connection Pool

Now that the credential containing the identity is propagated to the application tier, the next challenge is to use it to perform identity based database query, as shown in Figure 3. using JDBC connection pools.

JDBC connections are typically very expensive to create, which is why JDBC connection pools are used rather universally in middle tier applications. This also allows application code to abstract away the task of creating and managing connections. While it is possible to create a new connection for each user based on their credentials, the resulting application will exhibit high latency, and poor scalability. Therefore a major design goal is to reuse connections. The disadvantage to the connection pool is

that, most JDBC connection pools are created using a shared-user with special privileges, in most cases super-user to perform database operations.

Vendors provide customized JDBC data source libraries that allow application to override the connection pool credential, utilizing proxy authentication mechanism [9]. This implies that the application code has embedded proxy authentication code, rendering it dependent on vendor proprietary APIs and libraries.

An alternative is to implement a credential mapping mechanism. The challenges in implementing credential mapping involve either duplicating or creating new identities per domain. Most security domains are hesitant to provide access to their enterprise identity repository, forcing federated partners to create new credentials that map to the identity included in the service request.

In some instances, security governance may not allow that. Either of the security strategies lacks fine grained access control for downstream resources unless implemented using additional vendor products or embedding the security policy in the application.

#### *Gaps in the Web Services Security Standards*

As illustrated above even though WS-Security is the de-facto standard for web service security, and, while SAML is gaining acceptance as a means for exchanging security information, it does not yet span across all the enterprise tiers. J2EE vendors have successfully implemented accepting SAML assertions and providing access control, but they lack mechanisms for propagating the identity to downstream service chains or to the database.[10]

To resolve this issue developers are forced to embed an identity propagation mechanism in the application.

### **PROPOSED FRAMEWORK FOR CREDENTIAL PROPAGATION**

The following section describes a proposed pattern to protect resources such as web applications, web services, and web service persistent data. These resources are protected using security enforcement points that interface with policy stores to enforce admission control to user requests as well as filtering result sets from persistent data based on policies. Figure 4 describes the information and security credential flow in the n-tier architecture.

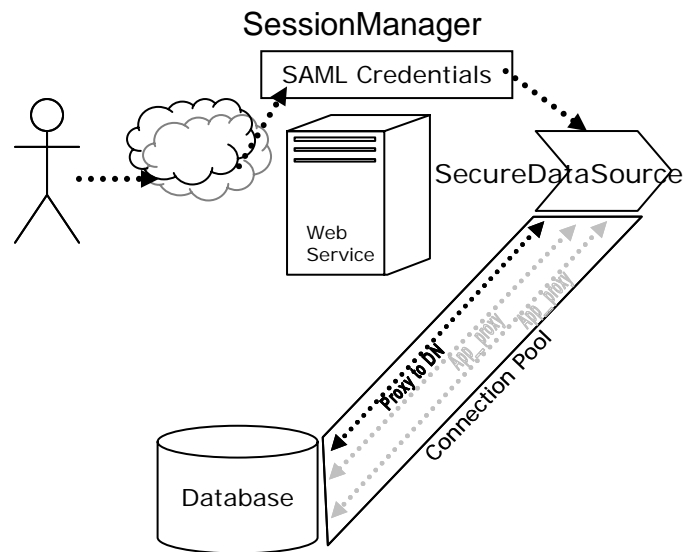


Figure 4 Identity propagation for N-Tier Architecture

A brief description of the tier components and configuration is warranted for the following tiers. The first tier (not shown) is the Web Application tier that is normally used to invoke a web application or a web service. The second tier which is a service provider, in this case an identity-aware web service hosted on a J2EE application server. The J2EE application server configured with a JDBC connection pool to a database server with label security extensions.

User/Application invokes a web service with a WS-Security SOAP message containing a SAML authentication statement and a Subject Identifier, shown as SAML Credentials in Figure 4. Typical deployments use policy enforcement point (PEP) [11], to intercept the SOAP request and performs authentication, authorization and auditing for the web service resource. Once granted access, the J2EE container invokes the web service passing it the SOAP message without the WS-Security header information.

Options available at this point to propagate the credential from the WS-Security header:

#### *Web Service Handler Using Session*

One option is to utilize a handler chain for the JAX-WS web service that parses the SAML Credential from the SOAP header and saves it as an attribute on the SOAP Message Context. The service implementation method can retrieve the credential from the SOAP message context session.

There are a number of issues with this approach. The first being that the standard Principal types provided in JEE don't include support for SAML. It appears that many containers have provided a custom principal that can

support SAML, but this would not be portable across containers. The second issue is that this places a dependency on EJB sessions, which are not normally required for web services. This could make lightweight solutions (without app server) problematic. The final issue is that still have to work out how to populate a Session Context from a SAML token in a web service call. The advantage of this approach is that the Session Context is Thread-Local and should therefore accessible from any code running in the same thread.

Alternately, the same JAX-WS handler approach would be used, but the SAML credential would be saved to a Thread-Local session object. As long as connection requests come from the same thread as the handler, the credential is available. The advantage of this approach is that it doesn't depend on any JEE infrastructure at all.

#### *Secure Data Source*

An elegant solution is to mitigate vendor dependency, by introducing a notion of wrapping a vendor standard JDBC Data Source with a secure plug-in (Secure Data Source) that utilizes the credential in the Thread-Local session as shown in Figure 4. For example, the Thread Local session object which contain a SAML assertion containing a Subject Identifier: `dn=testuser,cn=Users,dc=myorg,dc=com`, is used to override the connection pool credential. This is the same identity/user that the Oracle LBAC is configured with. The Secure Data Source performs the override mechanism, independent of the application. The application simply invokes JDBC call unaware of any security context.

Existing database vendors provide multiple identity provisioning and enforcement mechanisms. Currently, they lack accepting a credential in a SAML assertion, Secure Data Source provides this abstraction, as it can accept the SAML assertion or any other credential token that is in the Thread-Local session object.

Secure Data Source is a pure Java plug-in that is J2EE server agnostic.

#### **FUTURE DIRECTION**

Extending the current WS-Security mechanisms in database would simplify identity propagation, by minimizing and in most vendor implementation remove the need for credential mapping. This research area would focus encoding security tokens and assertions (SAML) in database operations.

Support for different session managers used for the identity propagation is another area research. Session managers identified for research are EJB and JMS sessions. In addition support for different credential types (Kerberos, X.509, Binary Security Token) that can be propagated as a

security Subject (`javax.security.auth.Subject`), which allows the solution to be extensible and portable.

Another research focus is the query augmentation to enforce fine grained label based access to databases that do not have built-in support for label security.

#### **SUMMARY**

The proposed N-Tier identity propagation approach fills key gaps that the current WS-Security standard does not provide, by keeping the security logic out of applications. Most vendors fill the gap by providing extensions in the form of either API or custom libraries. One might utilize the extension to fill the gap, but that could lock one into the vendor based solution, creating possible interoperability issues. The key strength of the proposed mechanism is that it provides a level of trust, since application development is transparent of any security policy.

#### **REFERENCES**

- [1] Poddar, I.; Goldszmidt, G., "Automated deployment and Aggregated access control for SOA composite applications," *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on* , vol., no., pp.833-847, May 21 2007-Yearly 25 2007 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4258616&isnumber=4258514>
- [2] Uchenick, G.M.; Vanfleet, W.M., "Multiple independent levels of safety and security: high assurance architecture for MSLS/MLS," *Military Communications Conference, 2005. MILCOM 2005. IEEE* , vol., no., pp.610-614 Vol. 1, 17-20 Oct. 2005 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1605749&isnumber=33743>
- [3] <http://xml.coverpages.org/saml.html>
- [4] <http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf>
- [5] <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
- [6] <http://xml.coverpages.org/WS-Trust200502.pdf>
- [7] [http://download.oracle.com/docs/cd/B28359\\_01/network.111/b28529/faq.htm](http://download.oracle.com/docs/cd/B28359_01/network.111/b28529/faq.htm)
- [8] Yuan, E.; Tong, J., "Attributed based access control (ABAC) for Web services," *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on* , vol., no., pp.-569, 11-15 July 2005 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1530847&isnumber=32665>
- [9] [http://download.oracle.com/docs/cd/B10501\\_01/appdev.920/a96590/adgsec03.htm](http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96590/adgsec03.htm)
- [10] Phan, Cecilia, "Service Oriented Architecture (SOA) - Security Challenges and Mitigation Strategies," *Military Communications Conference, 2007. MILCOM 2007. IEEE* , vol., no., pp.1-7, 29-31 Oct. 2007 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4455012&isnumber=4454733>
- [11] <http://www-01.ibm.com/software/integration/datapower/xs40/features/index.html#accesscontrol>