SOA Management and Security Enforcement

MARIANA GORANOVA, JULIANA GEORGIEVA, BOGDAN SCHISCHEDJIEV
Department of Programming and Computer Technologies
Technical University of Sofia
Sofia, Bul. "Kl. Ohridski" 8, bl. 2
BULGARIA
mgor@tu-sofia.bg, july@tu-sofia.bg, bogi@tu-sofia.bg

Abstract: - This paper discusses the importance of integrating SOA management and security enforcement into an existing IT management environment. It is proved why the thinking about management and security earlier helps to establish effective governance. The task of unifying the management and the security enforcement in a SOA is been resolving using the mechanism of JaxView and Security Token Services. Architecture schemes and JaxView functions for achieving the solution of the raised issues are suggested. Using JaxView to centralize security for Web services saving time and easily security management instead of implementing and managing security functions on individual service interfaces is enabled and depicted.

Key-Words: - SOA, service, management, security enforcement, security token, JaxView, proxy gateway

1 Introduction

The most effective way to manage a service oriented architecture (SOA) infrastructure is to manage it without having to disrupt the SOA [5]. There are more than a few SOA management solutions. Most of them require an agent to be installed on all the application servers or gateways to be able to provide visibility [1], [2]. Managing the agents and deploying the agents is both time consuming, disruptive and intrusive [9], [10]. With these agents installed you could then get visibility into different protocols such SOAP, REST, JDBC, RMI etc. But the over head the agent adds to the server to be installed on and the time to configure all the agents is too much for many companies that they just go about their SOA without a management tool.

A better way to go about SOA management is to manage the SOA environment without using any agents by monitoring at the network layer instead of the server [3]. Many network monitoring tools use this architecture to monitor *tcp/udp* packets. The question to use the same mechanism to monitor other protocols that are used within the SOA is raised. That is exactly how the agent less deployment from JaxView monitors the SOA.

Some reasons in favour of agent less approach are generalized in [4].

- **Platform Independent** Obviously since nothing is installed on the server there is no dependence on what platform the server is installed on.
- **No Change to Provider or Consumer** Since this is managing and providing SOA visibility by monitoring out of band traffic there is any change that occurs to either the consumer or the provider.
- **No Load – No Latency** Since the monitoring is done on the network and only out of band traffic this adds virtually zero latency or load to the system as a whole.
- **All Protocols** By dissecting all *tcp* protocols this deployment can provide visibility into SOAP, REST, JMS, JDBC, RMI etc.
- **Service Discovery** This is the best way to auto discover *rogue* services in the SOA environment.

Security remains an ongoing concern in SOA connected with the followings: 1) Web services have to apply security policies consistently; 2) the changes to security policies have to be implemented easily across all services; 3) the SOA developers have to implement security functions for individual services.

The delivery of Web service monitoring and management is closely knitted to effective identity management and perimeter security. That division between application development and operational security has worked so long as security could be embedded in a network hardware appliance, but when it runs across both hardware and software solutions, the implications aren't confined to the operational domain. Most Web service vendors agree that the only viable solution in the long run is to set up and manage user identities and access policies as a separate dedicated resource, and then implement the Web service infrastructure so that it enforces those security rules [2], [8]. When using

Web services, the next challenge comes when an authenticated user wants to access a specific resource. This needs a way of securely passing the appropriate credentials from the access management system through to the requested Web service [4].

The very nature of the dynamic SOA world, consisting of many component services that can be recombined and reused, gives rise to potential SOA responsiveness, management issues and business risks, for instance: 1) how can you monitor what's going on in the heterogeneous, distributed SOA environment, to find and resolve problems quickly?; 2) how can you know and ensure that your SOA business processes are delivering the right level of customer service?; 3) how can you guarantee that all services are responsive, secure and compliant with appropriate regulatory and business policies?

2 Problem Formulation

The thinking about management and security earlier helps to establish effective governance – assigning clear decision-making rights. Management, as a part of the SOA lifecycle, should not be thought of as a discrete step in a linear sequence. Instead, tightly integrate management and security with the modelling and design of *Web services.* This is a relatively new approach which has not been examined and depicted in the literature sufficiently.

2.1 Service Manager

One example of environment in this connection is *Service Manager*, which is the industry's leading SOA management and security product [6]. It intermediaries provide the foundation for SOA management, security, and run-time policy enforcement. Service Manager provides the industry's most comprehensive SOA security solution for trust enablement of Web service providers and consumers. Service Manager provides: 1) comprehensive end-to-end security of Web service messages (authentication, authorization, privacy, non-repudiation, and audit); 2) built-in PKI (*Public Key Infrastructure*); 3) security token exchange and mediation services; 4) first-mile security; 5) last-mile security; 6) in-transit security.

The raised issue of unifying the SOA management and the security enforcement will be resolved using the mechanisms of JaxView and Security Token Services.

2.2 JaxView – security and monitoring tool

The proposed *JaxView* tool is easier to be used and provides quick time-to-value while is offering a deep and wide set of governance features that are otherwise found on tools with a much larger footprint and price. It can be deployed very quickly and monitors and secures all service applications. *JaxView* is the most cost-effective yet comprehensive SOA governance tool on the market [7].

The *JaxView* server installs as a service on Windows and is immediately ready to use as a service proxy gateway. The management console is hosted on the same server and, once you log in, you see the interface depicted on. Web services under management appear in the left pane and can be grouped into folders for convenience. The main frame of the "Services" tab gives you a simple view of the performance and operability (errors, alerts, and faults) for the level of the Web service hierarchy (all, folder, or individual service) view that you choose in the left navigation pane. A summary report of how a service operation is performing could be received as well. The most important services could be classified in the following groups: 1) monitoring services with theirs three categories: performance, transaction, and availability monitoring; 2) managing services involving applying policies to service-consumer interactions and including security polices as well as routing and transmission ones; 3) scheduled and on-demand performance and operability reports for the services.

2.3 Security Token Service (STS)

Web services need to authenticate clients in a heterogeneous environment so that additional controls such as authorization and auditing can be implemented. How does the Web service verify the credentials presented by the client? The answer is to use brokered authentication with a security token issued by a Security Token Service (STS). The STS is trusted by both the client and the Web service to provide interoperable security tokens.

The client sends an authentication request, with accompanying credentials, to the STS. The STS verifies the credentials presented by the client, and then in response, it issues a security token that provides proof that the client has authenticated with the STS. The client presents the security token to the Web service. The Web service verifies that the token was issued by a trusted STS, which proves that the client has successfully authenticated with the STS.

ISBN: 978-1-61804-081-7

The protocol used for issuing security tokens is based on WS-Trust. WS-Trust is a Web service specification that builds on WS-Security. Fig. 1 illustrates the process by which a security token is issued to the client by the STS and then is used to authenticate with a service, which then returns a response to the client.

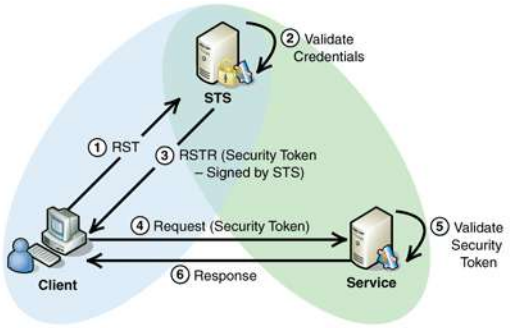

Fig. 1

2.3.1 The client initializes and sends authentication request to the STS

It is in the form of an RST message (provides the means for requesting a security token from an STS or directly from the server). This step can be performed by presenting the client's identifier and proof-of-possession (such as user name and password) directly to the STS or by using a token issued by an authentication broker (such as an X.509 digital signature or Kerberos tokens).

2.3.2 The STS validates the client's credentials

After the security token service determines that the client's credentials are valid, it may also decide whether to issue a security token for the authenticated client. For example, the STS may have a policy where it issues tokens only for users who belong to a specific role or for valid X.509 certificates that can be validated through a specific trust chain.

2.3.3 The STS issues a security token to the client

If the client's credentials are successfully validated, the STS issues a security token (such as a *Security Assertion Markup Language)* for exchanging authentication and authorization data between security domains in a RSTR message (returns the requested token and supporting state) to the client, typically, the security token contains claims related to the client. The security token is usually signed by the STS; when the security token is signed by STS, the service can confirm that the token was issued by the STS and that the security token was not tampered with after it was issued.

2.3.4 The client initializes and sends a request message to the service

After the client receives a security token from the STS, it initializes a request message that includes the issued security token, and then it sends the request message to the service.

2.3.5 The service validates the security token and processes the request

The security token is validated by the service that verifies that the token was issued by the trusted STS and that the token was not tampered with after it was issued. After the token is validated by the service, it is used to establish security context for the client, so the service can make authorization decisions or audit activity.

2.3.6 The service initializes and sends a response message to the client

A response is not always required. Frequently, the response message contains sensitive data, so it should be secured.

3 Problem Solution

There are two raised issues to be resolved: 1) to propose the architecture scheme for achieving SOA management and security enforcement simultaneously; 2) to select JaxView' functions supposed to fulfill all requirements as far as SOA management and security enforcement are concerned.

The questions that have to receive an answer in a flexible SOA environment could be summarized as follows: 1) are your Web services available to the applications that need them; 2) can your end-users complete transactions that flow through services; 3) how do you identify problems and resolve them quickly; 4) do you know who is accessing your services and can you prevent unauthorized access; 5) can you report and quickly respond to changes; 6) who is accessing your Web services and how often; 7) how are your services and transactions performing; 8) are you able to audit message communication between the consumer and the providers of service; 9) are end users having problems and how can they be fixed; 10) are you looking to discover rogue services automatically.

The solutions of the raised problems will be developed consecutively, showing some schemas and functions to be used. The first goal is to manage the SOA environment without the use of any agents by monitoring at the network layer instead of the server.

3.1 Agent less deployment architecture for monitoring services and enabling security and policy enforcement

Fig. 2 shows: 1) how JaxView could be deployed as a service proxy or XML firewall assuring security

and policy enforcement for services; 2) how it could communicate with the installed *agent* components; 3) how it could fulfill the role of a *switch* in order to listen to the network traffic. The enforceable security policies include authentication, authorization, single sign on, encryption, digital signatures, and integration with identity management systems.

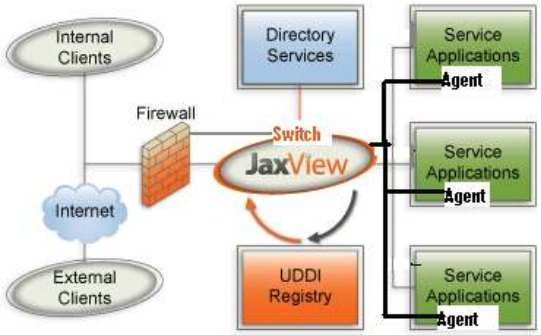

Fig. 2

3.1.1 JaxView as service gateway / proxy

The role of the proxy server is to forward messages from clients (internal and external) to the service endpoints (service applications). In this architecture, all service messages are routed through the JaxView proxy where copies of message data are forwarded to the processing modules and service requests are forwarded to service endpoints. When installed in this configuration JaxView can be used as an XML firewall. The server allows security giving end-to-end control of all aspects of SOA management and SOA governance as: visibility, availability, security, and brokering (explained later in this paper). The directory services allow more easily management of network resources. Management of service infrastructures through bi-directional communication with UDDI registries is enabled as well.

3.1.2 Message agent

It is a component that is installed on the container where a Web service is run. No code modifications are needed at the Web services. It is used to forward service message data from Web application servers to the JaxView server.

3.1.3 Monitoring Web services as a network appliance

JaxView is configured to passively listen to network traffic, monitoring packets for Web service requests and responses. It then records the service information and message data for management purposes. It enables auto discovery of Web services and can help uncover rogue services in the environment.

3.2 SOA visibility

Service processes that are composed of interactions with multiple service end points create new challenges for understanding the many layers of service activity. It is more important to have complete visibility into all Web service activity and end user experiences, and be able to diagnose the source of problems when they happen.

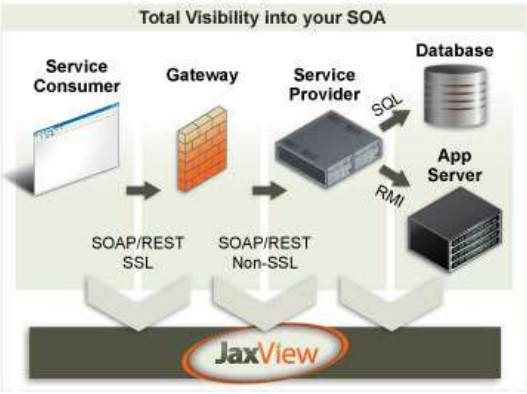

Fig. 3

Fig. 3 shows how and where JaxView delivers visibility across the SOA environment. Some of the issues resolved by JaxView could be generalized as follows: monitoring SOAP service messages and REST messages; enforcement of Service Level Agreements (SLA); visibility into backend protocols such as RMI, and SQL; auto discovery of existing Web services; automatic identification of rogue services; centralized view of all Web service activity; client usage rate monitoring; monitoring of service throughput per client; monitoring fault totals and fault rate; fault type and content monitoring including fault code and fault text string; the number of faults as a percentage of requests.

3.3 SOA security enforcement

The goal is to save implementation time and ease security management by using JaxView as a service proxy to check and enforce policies on XML message content [11]. The three main tasks deployed by JaxView are: 1) authenticate and validate users and services requests; 2) enforce security policies for service access and usage, and 3) automatically enforce security policy changes.

Fig. 4 shows the two main points which have to be validated: 1) the service requests from the consumer ("first mile"); 2) the application server (“last mile”). JaxView is able to extend the security enforcement functionality to include the security policies as a XML gateway in the first side and a XML firewall in the second. Some of the needed and used essential features for SOA service security, assured by JaxView could be summarized as follows: 1) security proxy and XML firewall

functions (authenticate and authorize consumers using STS; encrypt or decrypt XML message content for both request and response; insert digital signatures in the request or response xml messages; validate XML digital signatures and WS-Security headers etc.); 2) consumer authentication and authorization**;** 3) threat protection from: replay attacks; XML bomb attacks; SQL injection; XPath injection; denial of service attacks etc.; 4) embedded secure token services; 5) runtime policy enforcement functions (used to combine enforcement of automated policies on service access and usage with visibility into policy compliance); 6) Web service policy profiles (created and assigned to a group of Web services).

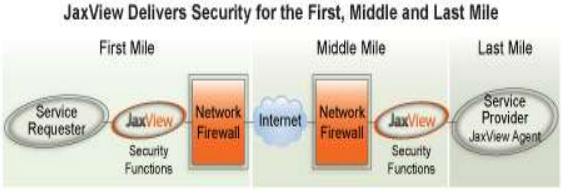

Fig. 4

3.4 SOA message brokering

JaxView acts as a XML Gateway to enable runtime message mediation and content-based brokering for Web services (Fig. 5). This feature allows to: easily integrate different systems; convert protocols from http to non-htp and vice versa; create virtual services from artefacts of other services; dynamically modify message schemas etc. Some of the used *runtime service mediation* and *brokering* functions are: Modify service message schema in runtime to mediate between different systems and protocols; Reroute service requests in runtime based on content checking rules such as service version; Allow only a specific number of messages to reach the service in a specific period of time; Provide load balancing and failover support for back end services.

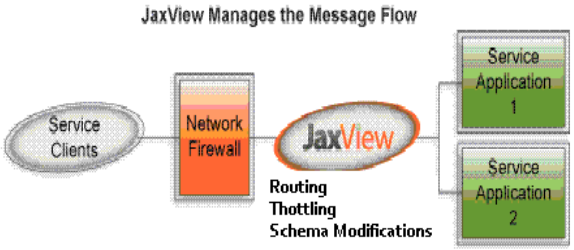

Fig. 5

3.5 Web services availability monitoring

A few issues are raising here: about the availability of Web services; about altering when the problems happened; about the correctness of services' responses. Some of the used functions are: 1) service end point availability monitoring; 2) multi-step transaction availability; 3) service performance metrics; 4) event notification alerts; 5) reporting functions.

3.6 Infrastructure model for JaxView Security Token Service

JaxView includes a Secure Token Service for creation and validation of tokens for single sign on functionality. You can have a single JaxView work both as a STS and a gateway or separate the functionality. The proposed model is given on Fig. 6.

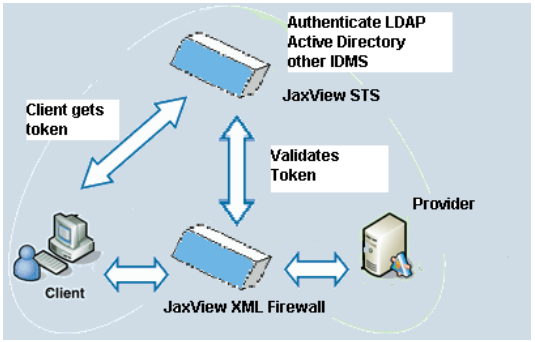

Fig. 6

Client (consumer) sends authentication request to JaxView STS. The RST message contains a security token that holds the client's credentials, which are required to authenticate the client. Claims in the client's credentials, such as a password, may be sensitive in nature, so it is very important to secure the RST. The specific security mechanism used for securing the RST depends on the relationship between the client and the STS.

JaxView STS validates client' credentials and authenticates the user through: 1) *LDAP* (*Lightweight Directory Access Protocol*) for accessing and maintaining distributed directory information services over an Internet; 2) active directory (a service included in most WS Server operating systems which is responsible for authenticating and authorizing); 3) *IDMS* (*Identity managed system*) assuring immediate the provisioning of unnecessary rights disallowing the availability and use of user accounts or availability of non-locked accounts etc.

After the authentication the information goes back to the consumer and so the client gets token usually signed by the STS. The token is embedded in the request.

Next step is sending the request to JaxView XML Firewall which is needed its validation as well.

So JaxView XML Firewall sends the request to JaxView STS in order that the last validates token.

The request is sending to the service provider. The response message should be secured because it

ISBN: 978-1-61804-081-7

contains sensitive data. That is why: 1) this message is sent to JaxView XML Firewall; 2) the XML encryption and decryption for the authenticated request are done.

3.7 Algorithm based policy enforcement

JaxView XML Gateway enforces policies based on most standards (for example WS-Security standards which can automatically evaluate authentication parameters).

The options for algorithm based policies within JaxView include: 1) dynamic schema modification – the ability to invoke an algorithm to modify the request or response for the service as a policy of the service; 2) user name token retrieval – the ability to invoke an algorithm to retrieve the username and password from the request header or body; 3) schema validation – invokes an algorithm to validate the schema of the request or response as a policy of the service; 4) XML encryption – invokes an algorithm to encrypt or decrypt part or all of the request/response payload; 5) exception based routing policies – JaxView allows the user to look for specific faults from the service to re-route the request; 6) STS integration – JaxView can request and validate tokens from a STS using WS-Trust but an algorithm policy can be set for proprietary STS integration protocols.

4 Conclusion

The proposed approach in this paper is to manage the SOA environment and security enforcement in a single step without using of any agents by monitoring at the network layer instead of the server. This truly agent less monitoring option using the so called switch has very low overhead and minimizes additional traffic on the network, monitoring packets for Web service requests and responses.

Using JaxView monitoring tool to centralize security for Web services to save time and ease security management instead of implementing and managing security functions on individual service interfaces is enabled. The infrastructure model for JaxView security token service is presented. The model of JaxView's deployment as a service proxy assuring security and policy enforcement and listening to the network traffic is proposed. The options for algorithm based policies enforcement within JaxView are composed.

The main contribution is the integration of SOA management and security enforcement into one environment and investigations of using a concrete tool in order to resolve this issue.

ACKNOWLEDGMENT

This paper is supported by the National Scientific Fund of Ministry of Education and Science in Bulgaria under the contract № DO 02-175/2008.

References:

[1] Keith Rodgers, *Security Rules in SOA management*, Loosely Coupled Monthly Digest, November 2004.

[2] *End-to-end Visibility, Security and Control of Your Service Oriented Architecture*, http://www.progress.com/en/Product-Capabilities/soa-management.html.

[3] *The Challenge of Securing SOA*, New Rowley Group, Inc., 2006.

[4] *Secure and Manage Services in an SOA Environment to Achieve Business Objectives*, IBM Corporation Software Group, 2006.

[5] Axel Buecker, Paul Ashley, Martin Borret, Ming Lu, Sridhar Muppidi, *Understanding SOA Security, Design and Implementation*, IBM Technical Support Organization, 2007.

[6] *Service Manager – SOA Management and Security*, 2001 – 2012 SOA Software, Inc.

[7] Paul O'Connor, *SOA Product Review: Managed Methods JaxView 4.0*, http://soa.sys-con.com/node/620354, 2008.

[8] T. Mouelhi, F. Fleurey, and B. Baudry, A Generic Metamodel for Security Policies Mutation, *IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08)*, 2008.

[9] I. Momtchev, Intelligent Agents – Issues Analysis and Architecture Proposal, *Proceeding of 18th International Conference Systems for Automation of Engineering and Research (SAER)*, 2004, Varna, Bulgaria, pp 201-206.

[10] A. Kazandzhiev, I. Momtchev, L. Popova, and D. Shikalanov, Distributed Multi-Agent Based Approaches, *Proceeding of KIMAS 05*, IEEE, 2005, Boston USA, pp 3-9.

[11] A. Georgieva, B. Georgiev, Nontraditional Approach to XML Web Services, Interactions, *Proceedings The Fifth International Conference on Internet and Web Applications and Services*, 9-15 May, Spain, Barcelona, 2010, ISBN: 978-0-7695-4022-1, pp.67-72.