# Security and Identity Management Applied to SOA - Part II

*by Jose Luiz Berg, Project Manager & Systems Architect, Enterprise Application Integration (EAI)*

**Web Services**

To understand how to integrate Web Services with security infrastructure, we must first define some fundamental concepts. We have already said in the previous chapter that the great challenge of security with respect to Web Services, is that they break the boundaries between applications, transforming all applications in a single big one. This statement is not true only regarding to Web Services, but as for any technology allowing remote execution of routines. In this document, when you read Web Services, we are meaning remote services, whatever the technology used. According to Oasis, a service has the following definition:

> "A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.1 A service is provided by an entity – the service provider – for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider."

So, despite the objective of this document is the integration of Web Services with security infrastructure, where allowed, the term "service" is used to designate remote functionalities made available by an application, so that the same definition can be applied to any technology used. The term Web Service (WS) is used only when we drill down into the form of operation specific to Web Services.

When we talk about WS, we are assigning sets of functionalities made available by applications, which may be consumed by sending messages using high-level protocols such as SOAP or REST, and a means of transport such as HTTP or TCP/IP.

The challenge of building the security architecture for WS is to reconcile the internal systems development standards with market standards and the functionalities provided by security systems, in order to obtain an efficient pattern, easy to deploy, and where possible, compatible with other solutions available in the market. To meet these requirements we are going to consider the use of the WS-Security standard, developed by Oasis and a well know reference in the market today, being supported by the majority of the products.

*WS-Security*

The WS-Security standard was developed by Oasis, for addressing security requirements to WS. Unlike other standards such as Liberty Alliance and OpenID, which can also be used in Web pages, WS-Security is geared directly for use in service calls, made by a program, without human interaction.

As the standard was designed to be used in SOAP WS, data is always added within the tag "Header" of the message, using the schema ".XSD "defined by Oasis. As an industry standard, is implemented in numerous application servers and application firewalls, ensuring that the infrastructure will be compatible with market products. Does not fit within the scope of this document detail the WS-Security standard, but only the main services that are relevant to our study:

- *Encryption* – allows the partial or total encryption of the message by setting the encrypted blocks and algorithms required to perform the decryption. Public keys can also be included in the message, avoiding that they need to be previously known for decryption.
- *Digital signature* – the same way as in the encryption, signatures may be applied over the entire message or part of it, generating hashes using asymmetric encryption, and also including in the header of the message all information necessary to perform hash validation.
- *Authentication* – supports various authentication formats, through the inclusion of the user data in the message, using a component named "token". Supports several types of tokens, such as login/user binary tokens (X509 or Kerberos) or XML tokens, supporting the SAML assertion standard. In all cases, the tokens are digitally signed, ensuring that they cannot be changed over the wire.

As the necessary information for operations are always included in the header of the message, it is possible that all security validation can be done by a server without even knowing the rest of the message content. There are also several libraries of routines available in the market that implement the pattern, and may be used in the client or in the application server, to generate or validate messages. One of the most modern library today is Apache XCF. With XCF, is possible to handle many features and message formats, with support for the following technologies:

- Support for JAX-WS 2. x client and server
- JAX-WS API 2. synchronous, asynchronous and one-way x
- JAX-WS API 2. x Dynamic Invocation Interface (DII)
- Support for JAX-RS RESTful clients
- Support for wrapped styles and non-wrapped
- Support for XML messaging API
- Support for JavaScript and ECMAScript 4 XML (E4X)-client and server
- Support for CORBA
- Support for JBI with ServiceMix

The main problem for the implementation of Ws-Security standard is the complexity in the construction of the message, which is quite easy in the case of Java, with the use of XCF, and .NET systems using Microsoft WSE library. For PHP applications, can be used the WSO2 WSF/PHP, implementing a smaller set of functionality, however, reaching normal needs.

**Security components**

Established the standards which may be used by services, we are now re-examining security components, establishing how and where they will be implemented in the architecture. Whenever possible, the term service is used to denote a generic service, on any technology, and WS when is a specific detail to Web Services.

*Confidentiality*

The use of encryption in the communication channel is a requirement that strongly affects the performance of application servers. The cost of decrypting the entire message is high, and then should be used whenever the data is quite sensitive, giving preference to encrypt only necessary data within the message. In case was considered

that channel encryption is necessary, one should consider the possibility of accelerating the URL through reverse proxies and use SSL only to then, forwarding the message using HTTP to the application servers, centralizing the payload of encryption and exempting the servers responsible for implementing the business routines. With this separation of tasks, you have full visibility over the cost of communications and business processing, and for high loading implementations, you have the option of using hardware-accelerated decryption. In some cases, the services are executed both from clients and from other servers. In this case, you may mix various endpoints with different encryption schemes for each case.

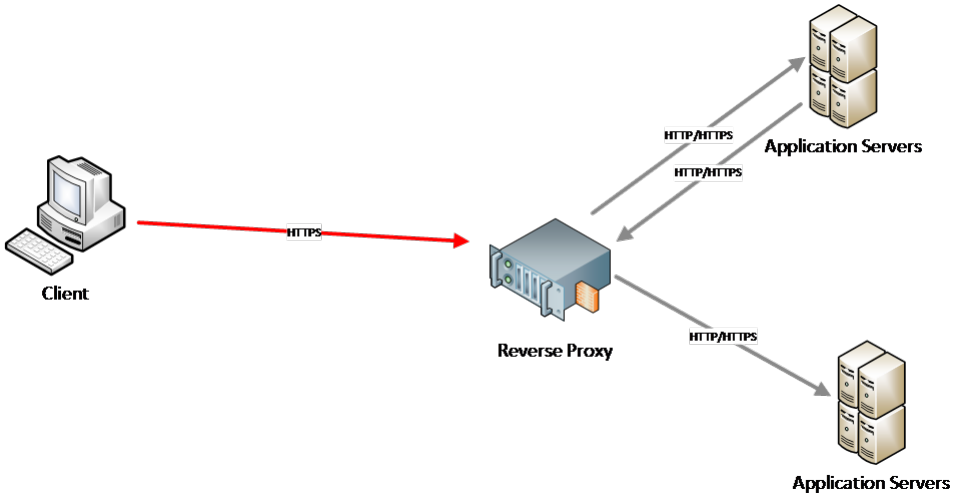


*Figure 1 – Executing a WS from a client and among application servers*

The definition of which data needs to be kept confidential is part of the definition of business service being implemented, and should be part of its requirements specification. In addition to obvious fields such as login and password, there may be numerous other fields that should not be disclosed, usually involving monetary values, internal identifiers, private personal data or even internal application passwords.

A possible attack with the breach of confidentiality would be monitoring valid messages searching for relevant information, such as credit card numbers, customer code, valid transaction numbers, and then build a fake message using these data, which could be accepted by the application, as it contains valid data.

*Integrity*

Data integrity is another requirement that must be answered along with the requirements specification of the service which will be built, because most of the time this vision is only possible for those who know deeply the meaning of the data to be processed.

The mechanism to ensure integrity is the digital signature, which can be applied over the entire contents of the message, or only on the parts indicated as sensitive. Unless the cost of processing become infeasible, a good practice is to always make the signature of all the content of the message, ensuring that can never be changed in transit.

One of the possible attacks that may be used about a WS is to intercept a message along the way, change any field not encrypted and send it again to the same destination. Another common attack is called “replay”, which consists of simply resubmit a message without changes, causing problems for the application, or even as a form of DoS (denial of service) attack. If this type of attack is relevant, the application may use control fields, dates, or even the hash of the message to identify and discard duplications.

In a B2C site, a WS can be used to finalize an order, including quantity of items sold. By intercepting the message, an attacker can increase the quantity. In this case, you can use the hash to identify the breach of the integrity of the message, refusing the operation. On the same site, the hacker could buy a product and resending the finalizing message many times. In this case, the hash of the message is valid, and the operation will be accepted unless any replay control was implemented.

*Non-repudiation*

The use of reciprocal certificate signatures depends on your client presenting valid public certificates for being used in the operation. This is easier in a B2B scenario, but not in B2C, where the end user as no experience in handling this kind of technology. This feature should be used in critical processes, normally involving high value monetary operations, where needs to be ensured that the user cannot repudiate the operation later.

The most common attack in this category is breaking the secrecy of the certificate store. Many users rely on weak passwords, write then down in a paper or simply lend their credentials for other people perform tasks in his own. Another common problem is using the same password everywhere, including sites on the Internet with inefficient credential storage. Upon discovering passwords for any user at a single site, a hacker will always try to find other places where the user has a record and try the same password. Another common way to discover the password is using free e-mail systems: many users use easy-to-remember passwords for these services because they are not critical, but later registers in other sites using the same e-mail address. After guessing your weak e-mail password, a hacker can access the functionality “forgot my password” in other sites, and the password reset will be sent to the compromised mail service.

Once again, the definition of when a mutual digital signature should be used or not, must be in business requirements, and should be established before building the service, defining which data should be signed and which type of signature applied.

*Authentication*

In the world of services, authentication is a lot more complex than in regular applications, because must be performed by a program, without a user to enter the password, and there is no session object to store data and control the access.

An easy solution to this problem would be to send the login and encrypted password in all services, but the problem is that to decrypt and validate password, applications would have to negotiate digital certificates, and once an application has your plain password, it may use in the wrong way, treating unsafely or booking in log files. A service is a black box to the requester, so sensitive data, such as passwords, should never be sent to services where we have no control of how they will be handled.

To resolve this problem, the solution was the use of “assertions”. An assertion is simply an XML snippet, usually containing the user ID, the date of authentication, the start and finish dates of the validity, the server and the type of authentication that was issued, and a unique identifier from authentication. A digital signature validates this XML, ensuring that it cannot be changed in the transmission. When you receive an assertion, a server can identify where authentication was issued (IDP), and validate it using the server public certificate. If it is valid (your

hash is correct), the IDP is trusted, is within the validity and the type of authentication matches the expectations, he then can trust that the authentication was done by the caller, and the received user is the consumer of the service. If it is necessary to execute a cascading service, the assertion may be included in the message, ensuring that the requesting user is known to all services in the chain.

The validation of assertions inserted in messages may be done in two different ways:

- *Using a reverse proxy* – before being forwarded to the application server. In this case, all the WS will be accelerated by him, and any call will be forwarded to the service provider only if contains valid assertions according to the specification of the service.
- *Directly in the application server* – using WS-Security libraries available for validating the assertion.

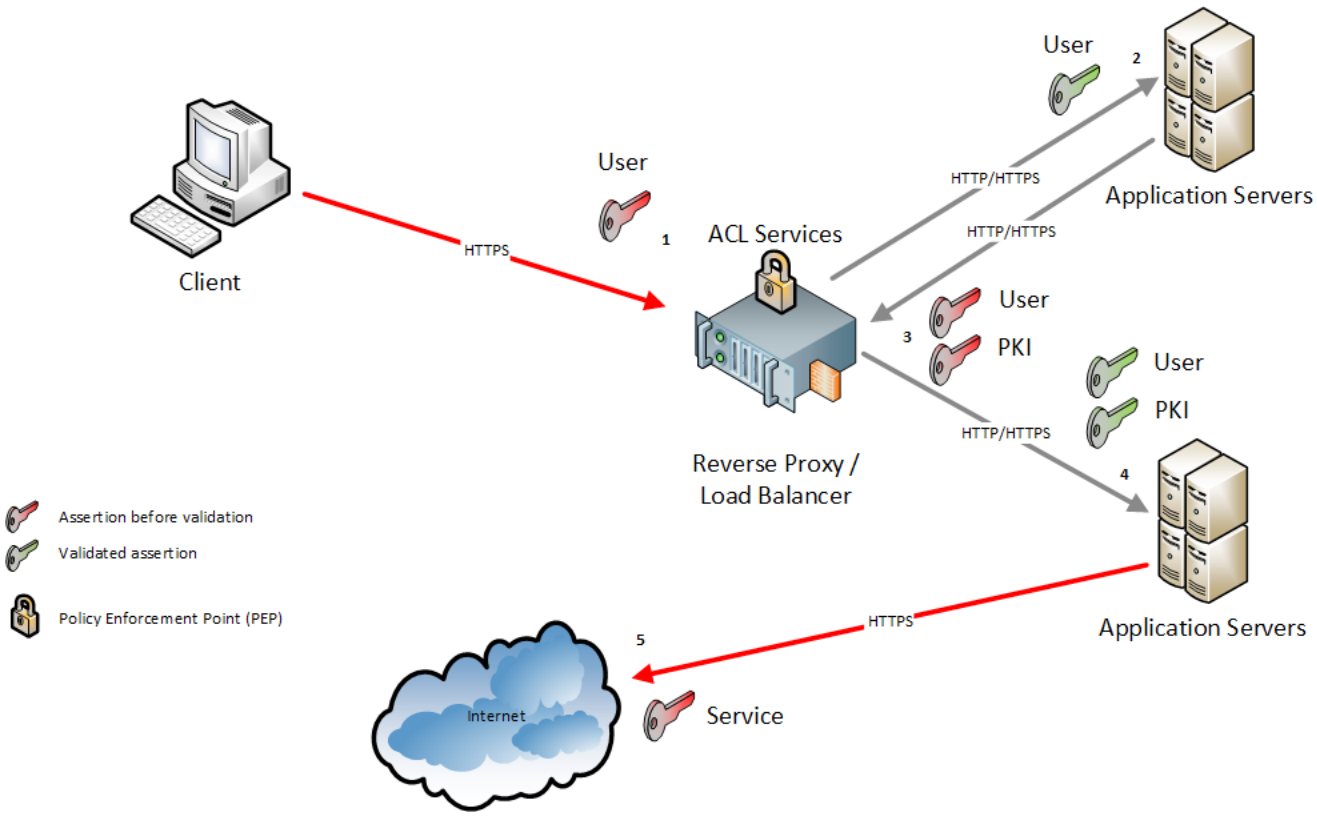


*Figure 2 – Assertion is validated in the reverse proxy*

In both cases, the application will never have to worry about authentication, because if the call gets into it, implies already contain the assertions specified and they are valid. The only reason for an application to access the assertion will be to seek some further details about the authenticated user necessary for its implementation.

However, before authenticating a service, we need to identify which users are required for authentication. There are several possibilities:

- *No authentication services* – not all services require authentication. A simple service that returns the list of states for a country, as an example, does not need to identify the user who is requesting the information. Some services of very low criticality, and usually to query data, do not require authentication.
- *End-user authentication* – is the most common case of authentication. Requires the credential of the user who authenticated and is using the application.
- *Service credential authentication* – some services require specific credentials to run, instead of the authenticated user. In this case, the service credential should be authenticated by WS, using login and password, or preferably via digital certificate linked to the server that will consume the service. However, even in this case, it is important that the service be aware which user has requested the operation, so the assertion of the end user must also be included in the message, facilitating audit trails and reporting on whose behalf the task is being performed. Another reason we should include the assertion of the end user, is that the service being executed may need to chain a second service requiring this credential.
- *Authentication with multiple users* – in some special cases, multiple credentials may be required to perform a service. When you call a call center and requests an operation, what is happening in fact, is that an operator is logged in the system, performing the operation on your behalf. The operator then requests oral confirmation of your data or typing a password or access code to confirm your identity. As we have seen above, this is also a form of authentication, which can generate an assertion. During operation, the system needs to perform some service that use service credential, so we have three assertions that can be sent: the operation is performed by the service credential, by request of the attending, on behalf of the end user. Still exists other forms of multiple authentication, as in cases of shared responsibility, in which two or more people need to authenticate simultaneously to request an operation.

Once again, the decision of which type of authentication and what credentials will be required for each business operation must be taken in accordance with business requirements, before the construction of each service.

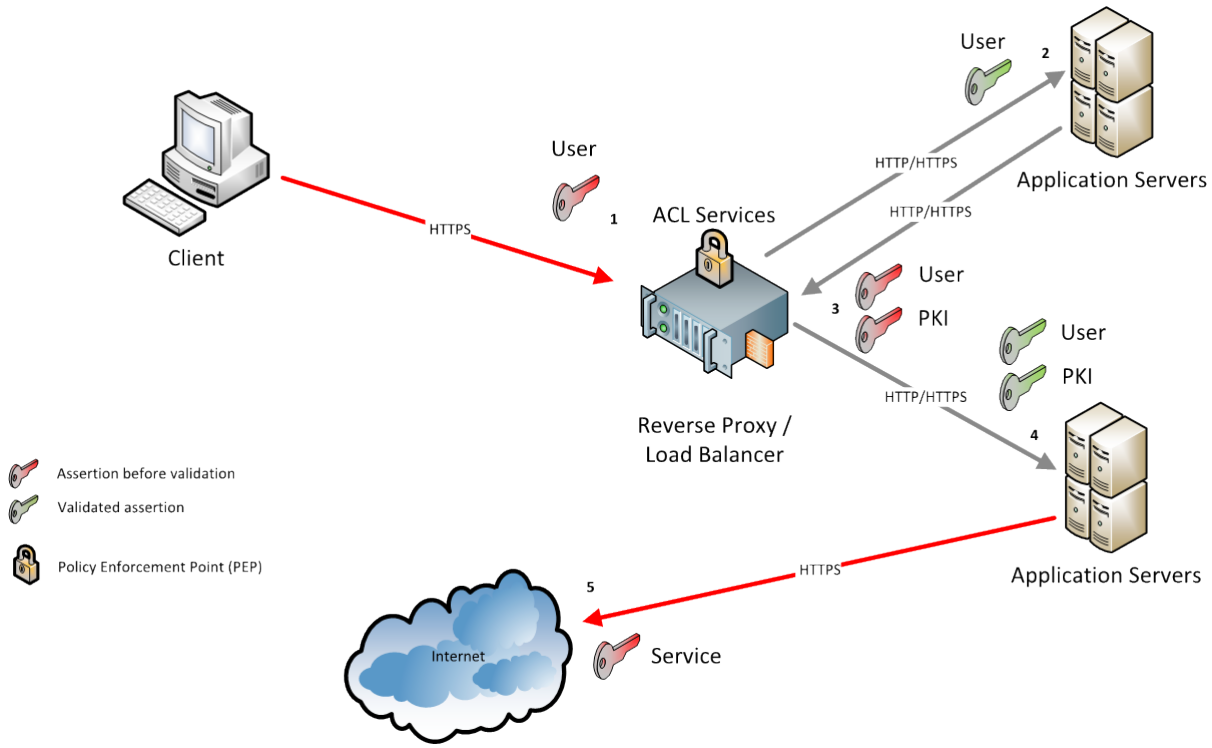


*Figure 3 – This is an example of using multiple assertions: the user executes the service A using his assertion; however, this service authenticates using digital certificate and executes service B, including both assertions; then service B needs to execute another service outside the network, authenticated with a service credential; although not needed for the service B, the user assertion needs to be sent, for identifying the requestor*

A critical point in the use of assertions belongs to its validity: an assertion is actually an XML that represents an access ticket. This XML can be transmitted, stored, or treated in any way, and if not changed remains valid within its period of validity. As we have no control over all locations where this assertion can pass, one of the possible attacks is the “credential hijacking”, i.e. by capturing an assertion a hacker can submit requests using it as the authenticating user. To prevent this type of attack, the assertions should always be sent encrypted, and have a short validity (typically between five and fifteen minutes). With that, even if one is caught, can only be used during this period. This short expiration time however creates a technical problem: user authentication in a Web server is attached to the browser session, and the expiration of the session is calculated relative to the last operation requested. However, the expiration of the assertion is absolute, calculated by the date of issue. Then we can have a valid session with an expired assertion. As the applications should never store the user’s password (even in memory), in this case the application should perform the logoff and forward the user to the login screen again, forcing a new authentication and receiving a new assertion. Some IDM systems allows the applications to extend the validity of an assertion, without presenting the credentials again. This must be used carefully, because if an attacker gets an assertion, he can keep renewing many times, bypassing the validity control.

When we use a service credential the task is a little easier, because in this case the application has the user's password or certificate, then just request a new assertion using the credentials. If an assertion is received in a server and its validity expires during processing, since it is not possible to request new user authentication, an error must be generated and the operation should be refused.

As we already defined to meet confidentiality, the decision about encrypting the entire message or only parts is from business, and must be taken case by case, but it is important that the assertions be never transmitted unencrypted. If the message is not encrypted, so it is recommended that at least the assertion be.

*Authorization*

Execution of authorization policy is a task usually accomplished by applications, but today this policy is mainly oriented to the presentation layer, hiding or disabling UI elements the user does not have rights to execute. However, WS does not have UI, so the challenge is moving this traditional authorization to the code, enforcing that be authorized even when the execution bypass the presentation layer. Of course that, UI elements must still be controlled according to user rights, so a duplicate validation must be performed.

One of the great advantages of modern IDM systems is using RBAC (role based access control) paradigm. This means that access rights should be granted according to the user's functional roles, regardless of the permission in each system. Thus, by assigning a role to a user in HR, he automatically would receive all the permissions that are required on all systems to perform the assigned role, and additional rights requests would be required only for exceptions, or any temporary tasks. Using this model, the management of profiles would be much easier than using the traditional model of assigning system roles and groups. However, this cultural change takes time, and the vast majority of IDM implementations keep the concepts of system roles. Therefore, every application need to set their roles and assign them to users who requested. These are the roles that are normally validated in the application servers, typically using ACLs.

When we map this functionality for services, not much changes, because the roles to be validated are the same, but each routine of an application that is provided as a service, necessarily must perform the authorization before its execution. The validation can be made through the same ACLs used for UI, but it is important that the roles required for the execution of a service must also be defined in the requirements specification of the service, so that they can be created and included in the validation.

When generating a SAML assertion for authentication, the IDP (identity provider) may include any necessary user attribute as additional parameters. With this functionality, would be possible to include all the roles a user have assigned, facilitating the authorization process. The problem is that as our roles are still dependent on the applications, and using SSO (single sign-on), we don't know what applications the user will access, you need to include all roles in assertions, which would increase the size of the message, reducing performance. Therefore, it is reasonable that there is some mechanism that allows the PEP (policy enforcement point) to check which roles the user is assigned to, for validating against ACL.

In addition to checking the user roles against the ACL, there are several other business authorizations, normally mixed to the code of regular operations. It is common to check for approval limits, areas of actuation, discount limits, and many others situations where the authorization belongs to business rules. Chaining service calls is a critical case of authorization policy, because the authorization should be validated before any service is executed. If a service for which a user has rights is executed, performs a part of the transaction, and executes a chained service, and the user does not have rights to execute this second service, may be necessary to roll back the first operation to maintain data consistency. Therefore, the permissions that are required to execute a service must also include permissions to perform all the cascading services. For this reason, separating the authorization code from operation code is a good practice, which can facilitate this task and avoid inconsistencies.

*Privacy*

The implementation of privacy criteria depends almost entirely on the definition of business, because only by knowing the information we know its privacy level, and under which conditions may be used.

The normal tools to ensure privacy are encryption and RBAC access restrictions, but we must also be careful especially in recording audit trails and logs and also in data storage in databases and other types of files, so they are made according to the privacy level required for each piece of information.

*Availability*

Service availability does not generate many constraints for their development, but it is important that operations with some critical requirement in this sense be monitored to ensure they meet the requirements. This is only possible if this requirement has been identified before the construction of the service.

*Audit*

Using services affects directly the audit routines, primarily for its distributed nature. For the generation of audit trails to be effective you need to consider all the existing systems and monitor all services and servers to identify when an operation started in one application, but also performed tasks in other applications inside the same business transaction.

The easiest way to do this is to create transaction identifiers, normally associated with the assertionID attribute, which is part of the assertion and is created at the time of authentication. In legacy systems, the record of transactions is done through user's login, which may cause confusion if the user is authenticated in more than one station. The assertionID, however, identifies each particular authentication. If the user opens two different browsers, logs to the same system, and executes the same operation in both, each operation is going to have different assertionIDs. The challenge is that generating this kind of audit trail is not usual for developers, who normally considers that regular logs are enough for auditing. There are many systems on the market specialized in capturing and generating audit events, through the receipt of messages from applications containing audit data. To receive these messages, these systems utilize transaction ids to define correlations between data and identify each business operation. Of course, that this can also be done using log files, but would be very much more easy and effective if you develop your system already including these information. Once more, identifying the boundaries for the transactions and which operations must be done by business specialists, and defined in business requirements.

**Technical Recommendations**

So far, we have identified the components of security, and mapped out how they affects the construction and use of Web Services, and how they should be implemented in the corporate infrastructure seamlessly to IDM. Now let us get down to some safety recommendations, indicating some best practices.

The security of services is a new discipline, and several gaps still exist that need to be filled to establish standards that can be considered relatively safe. In addition, until all these practices are assimilated by the systems architecture and internal development teams will take some time, so until there, some trade-offs can be made which may help in your implementation:

- Before there is a culture of using the WS-Security standard, it can be assumed that any WS built that requires authentication should use SSL at the transport layer. Thus, we avoid the complexity of partial encryption of messages for clients, unless there are specific requirements (i.e. performance).

- Any WS that can be called from outside the corporate network necessarily needs to be authenticated, and then, in accordance with the previous recommendation, use SSL.
- Special care must be taken with the security standards used in the market, because some are old and have known vulnerabilities, so the minimum configuration should consider AES or 3DES encryption, SHA256 signatures and certificates with minimum 2048-bit keys.
- Digital certificates for use as service credential should be generated related to the server where it will be used, have not too long validities (one or two years), and the revocation list must be made available regularly.
- Upon implementation of services security, the certificate infrastructure must be strengthened, because it will be essential for the operation of internal applications. Therefore, it is important to design a more robust structure of PKI, including the possibility of adding an HSM (hardware security module) to architecture, to handle the creation and safe storage of these certificates.
- To record application logs, the best existing technology today is the Log4J, or their variations: Log4NET and Log4PHP, which can be used for Java, C# or PHP, respectively. However, they serve mainly for the application log. For the audit trail must be negotiated with the audit team the best technology to be used. A simple solution would be to use the Log4J configured with Syslog loggers, but however by establishing a structured message pattern completely different from normal texts written in application logs.
- One of the main points of weakness in applications today, much used by hackers in attacks like “cross-site scripting” or “SQL injection” is the validation of data entry. As Web Services also serve as input for the systems, the same way as in the pages of the Web, applications should constrain and validate the data received before processing them. How to control and validate data entry is just out of scope of this document, but it is important to establish that makes no sense to implement security if the services remains open to such attacks.

**Conclusion**

The purpose of this document was not to establish standards for implementing security of services, but rather provide teams of systems architecture and development with technical allowances for these security standards be established. After this step, standards, patterns, norms and artifacts should be built for each case, aligned with your security policy, which should be disseminated to software factories and development teams, and be verified when the application is released, to ensure their adherence to the standards.

In addition to the definition of standards, it is important that architectural components be constructed, for making all these tasks as easy and transparent as possible to developers. If possible, these components should be installed on application servers, in order to enhance their use and adherence to standards.

The main message of this document is that makes any sense to use the most sophisticated firewalls and network controls, if your system maintains services that run without any security. Is the same as locking the front door, but leaving the back door open. The most vulnerable point will always initiate an attack, and that point will be the security level of your company. Today, the lack of knowledge and safety standards in the development of systems is one of the leading and most critical security failures of companies.

Building services is not an easy or cheap activity. A service is a piece of code that is executed by a request that comes from another computer, and has no display or user to validate their execution. In fact, it runs silently, and how implementation is not cheap, is used to run critical business operations. Therefore, unlike the existing common sense today, the safety recommendations should be specially strengthened for the services, because any irregular operation will only be identified by its result, usually a long time after, hindering the identification of the author, and therefore its subsequent correction.

For all these considerations, it is very important the definition of strict standards and best practices, and the involvement of the company's business areas to ensure that requirements are identified and met. In virtually all components of security, business information are necessary for its effective application, then real security is not made with technical features like encryption or fingerprint readers, but is a set of actions and information that must be used in combination to achieve the goals.

As it is common to hear in the area of security that "If simply closing doors would mean security, games at major stadiums should have no audience". Security is exactly maintaining only the required ports open, but having absolute control of who is coming in, what he can do and what he had done. This control can only be achieved with correct and up-to-date information, and when the standards are established and followed by all.

**Bibliography and References**


- O'Neill, Mark (1/31/2003). Web Services Security (Application Development). McGraw-Hill.
- Stuttard, Dafydd; Pinto, Marcus (8/31/2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. Wiley.
- Jothy Rosenberg; Remy, David (5/22/2004). Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption. Sams Publishing.
- Harding, Christopher; Mizumori, Roger; Williams, Ronald. Architectures for Identity Management. The Open Group.
- Skip Slone & The Open Group Identity Management Work Area. Identity Management. The Open Group.
- OASIS Web Services Security (WSS) TC. WS-Security Core Specification 1.1. Oasis.
- OASIS Web Services Security (WSS) TC. Username Token Profile 1.1. Oasis.
- OASIS Web Services Security (WSS) TC. SAML Token profile 1.1. Oasis.
- OASIS Reference Architecture Foundation for Service Oriented Architecture Version 1.0, Committee Specification 01, December 4, 2012
- Navigating the SOA Open Standards Landscape Around Architecture, a Joint Paper by The Open Group, OASIS, and OMG, July 2009
- OASIS Reference Model for Service Oriented Architecture 1.0, Official OASIS Standard, October 12, 2006

# Jose Luiz Berg

*Jose Luiz Berg is a long term project manager and a systems architect with Enterprise Application Integration (EAI). In the past few years, Jose focused his work on implementing Service Oriented Architecture (SOA) for large Brazilian telecommunication companies. He graduated in computer networks, but also has a lot of experience working as a programmer in commercial programming languages, in last 25 years. Jose believes that SOA is one of the most important advances in software development in last decades. As it involves not only a change in the way we work, but also a significantly changes how companies see themselves and their IT resources. This advancement may be a risk, as many companies are being convinced by bad software vendors that SOA is only creating Web services, however they are not focusing on what it really stands for. By doing so they are not realizing that this is important part of the history in the making.*

**Contributions**

- Security and Identity Management Applied to SOA - Part II
- Security and Identity Management Applied to SOA - Part I
- The Integration Between EAI and SOA - Part II
- The Integration Between EAI and SOA - Part I