XML Firewall Architecture and Best Practices for Configuration and Auditing

Don Patterson

XML Firewall Architecture and Best Practices for Configuration and Auditing

GSEC Gold Certification

Author: Don Patterson, donnyboy98@hotmail.com

Advisor: Dominicus Adriyanto

Accepted: February 28th 2007

Outline

1. Introduction

More and more organizations today are adopting Service Oriented Architectures (SOA) for mission critical business applications. By definition, SOA is not a technology, but rather the underlying framework made up of attributes such as dynamic service discovery, composition and interoperability. This framework promises autonomous interactions between software agents and computer systems to support information exchange and facilitate web transactions. Web services is the principal SOA platform that businesses and enterprises are now using to seamlessly automate end-to-end business processes, offer E-government services across the Internet, and conduct transactions such as financial trading and e-commerce purchasing. While providing advanced business functionality, Web services introduce significant security considerations and challenges that need to be effectively managed by the business to achieve tangible Return-On-Investment (ROI) by their use. For example, vulnerabilities can manifest in various layers of the architecture, such as the operating system, the network, the database, the XML parser, the firewall, or any other component in the Web services implementation.

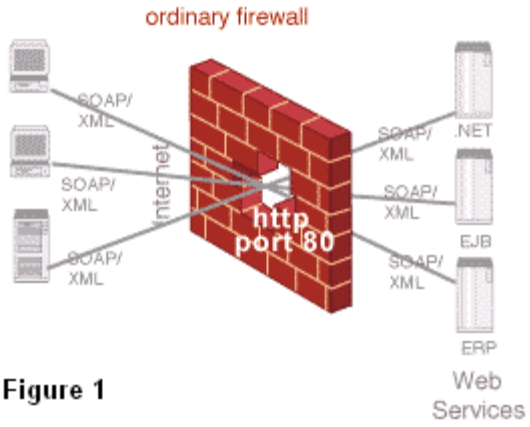

Figure 1

The traditional web security model filtered web traffic at the perimeter by deploying an IP packet-filtering firewall and restricting it to only allow

web traffic through certain ports, such as port 80 (HTTP) and port 443 (HTTPS). Unfortunately, this has proved to be an inadequate security control for Web services. Sophisticated hackers have crafted packets containing malicious content such as hostile payloads and executables, which traditional firewalls allow through to a vulnerable Web service (see Figure 1). This threat has brought about the need for sophisticated application-layer firewalls that can scan deep into the packets' payload and examine Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI), Security Assertion Markup Language (SAML) or other Web service protocols for attacks. These firewalls are referred to as XML firewalls or gateways.

XML firewalls are gradually becoming the norm as a first line of defense for Web services architecture and can even perform various security services such as authentication, authorization, auditing, XML schema validation and more. However, unless this technology is securely deployed, configured and tested correctly, web service threats may still pose considerable risks to critical web service applications. This paper will discuss the building blocks of Web services, Web services threats and security requirements, the XML firewall for first-line perimeter defense, best practices for configuring an XML security gateway device, and industry-recommended security testing procedures for ensuring the effectiveness of this security control.

2. Web Services Primer

An XML web service is a software component or system designed to support interoperable machine - or application -oriented interaction over a network. A Web service has an interface described in the Web Services Description Language

(WSDL) format whereby other systems can interact with the Web service using Simple Object Access Protocol (SOAP) messages. SOAP messages are typically transmitted using HTTP or HTTP(S) in conjunction with other Web-related standards.

XML web services are also referred to as: Web services, SOAP services, or WSDL services. A simple way to explain Web services is to compare them to web pages.[1] A web page takes an HTTP or HTTP(S) request and returns data in a comprehensible HTML format for a human to read and understand. Similarly, a Web service takes a SOAP request and returns data in an XML format to another software application, which allows it to perform tasks it has been programmed to do. A web service can have one or more operations, or web methods, as defined in the WSDL for the Web service.

Web services are based on technology that has been around for many years, such as HTTP, EDI, RPC, XML, SOAP.[2] In 2001, the major software vendors decided to work together and agreed on a standard for web services, which solidified the concepts and eased the implementation burden for web services. Such standards include WSDL, UDDI, and WS-* specifications promoted by Microsoft and IBM.

Most web services use standard SOAP syntax for formatting XML messages. SOAP essentially acts as a wrapper and container for the actual XML data being transferred. WSDL is the industry standard format to describe the operations and parameters that a web service supports. Any web service can be discovered and its' methods invoked by examining its' WSDL file. UDDI was developed as an industry standard for developing WSDL interfaces so that other web services can

dynamically locate and use a particular web service.

As previously stated, Web services messages are sent over HTTP or HTTP(S) across the network in SOAP format, an XML format defined by the World Wide Web Consortium (W3C). Although HTTP is the most common transport method, other transport protocols such as SMTP, FTP, and even IBM WebSphere MQ can be used. In most Web services, there are two types of SOAP messages: requests and responses. When a SOAP request is received, the Web service performs an action based on the request and returns a SOAP response. In many implementations, SOAP requests are similar to function calls while SOAP responses return the results of the function call. Web services are usually advertised by publishing a WSDL entry in a registry that contains a description of the service interface, data types, binding information, and network locations. UDDI is an example of a service registry. Below is an example of a typical web service transaction.

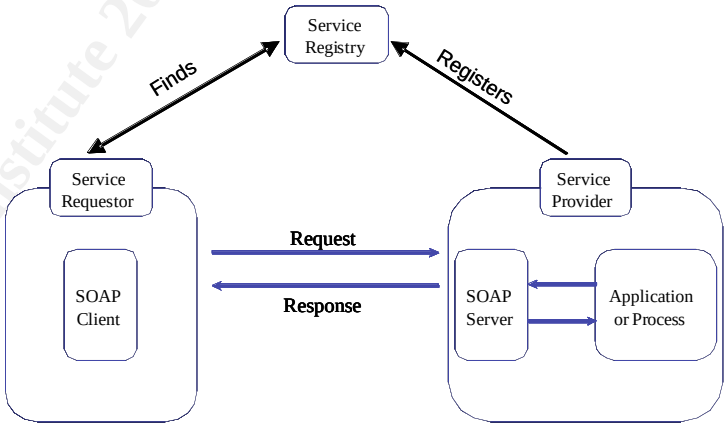

Figure 2 Generic Web Services Transaction

There are several SOA implementation case studies available to guide future implementations. [3] Most of these projects are in the initial stages of

implementation and have yet to realize full potential until major impacts on the enterprise and business partners are further analyzed. [15] The four main applications for web services are:

- *Application-to-Application Integration*
  e.g. a loan application may automatically send a request for a credit check to another application in order for an individual to receive the loan. The systems may be based on completely different platforms.

- *Business-to-Business (B2B) Integration*
  e.g. a Unix-based financial system could allow batch-or real time-processing of transactions to be sent from third parties, such as business partners.

- *Information Distribution*
  e.g. a federal agency can expose its' legacy system as a web service to enable information sharing between agencies, the defense, and intelligence communities.

- *Application Functionality*
  e.g. a standard function (e.g. encryption, digital signatures) can be exposed as a web service, so that any application, regardless of platform or hardware capabilities can use this function over the Internet.

As was mentioned, web services technology is gradually being accepted as a viable technology that will help businesses yield tangible ROI from an SOA implementation. There are three stages on its' road to maturity[4], of which we are well into the first stage:

- Stage 1: *Internal use and point-to-point integrations.* Most companies are using web services internally, in the form of test or pilot projects.

- Stage 2: *Cross-Enterprise*. Web services will be used as middleware to integrate the supply chain, and connect back-office systems between trusted partners.

- Stage 3: *Service-oriented architecture.* All applications allow dynamic web service discovery and integration for sharing of data and functionality. There are no technology or platform barriers between businesses or applications.

The following table provides a summary description of web services architecture and their main components.

Web Services Platform Element	Description

Web Services Platform Element	Description
SOAP	• SOAP stands for **Simple Object Access Protocol** • SOAP is a communication protocol • SOAP is for communication between applications • SOAP is a format for sending messages • SOAP is designed to communicate via Internet • SOAP is platform and language independent • SOAP is based on XML • SOAP is simple and extensible
WSDL	• WSDL stands for **Web Services Description Language** • WSDL is written in XML • WSDL is an XML document • WSDL is used to describe Web services • WSDL is also used to locate Web services

Web Services Platform Element	Description
UDDI	• UDDI stands for **Universal Description, Discovery and Integration** • UDDI is a directory for storing information about web services • UDDI is a directory of web service interfaces described by WSDL • UDDI communicates via SOAP
Consumer service (requester)	• Initiates a web service request on its' own or on behalf of a user via a web portal • Must send the proper syntax and follow security protocols required by the provider
Provider service (producer)	• Accepts a request from the requester and provides a response • Responsible for setting standards for security • Communicates its' requirements via WSDL and/or UDDI
Intermediary service (e.g. XML Gateway)	• Invoked at run-time depending on the needs of the user or application • Results in chain of Web service invocations unbeknownst to the requester web service.

3. Web Services Threat Analysis

Despite the value of web services, which includes greater accessibility of data, dynamic and ad hoc application-to-application connections, relative autonomy, and platform independence, it is obvious that wide sharing of data leads to more exposure of that data. The threats to Web services security include both the traditional exploits associated with the underlying protocols such as SSL/TLS (HTTPS), as well as new threats associated with message level protocols and services, such as SOAP and XML.

To begin with, XML and SOAP traffic are most commonly transmitted over HTTP or HTTP(S), which is allowed to flow without restriction through most firewalls. Traditional IP firewalls cannot distinguish between malicious and non-malicious XML and SOAP content tunneled via HTTP/HTTP(S); therefore, an attacker can bypass the perimeter IP firewall and gain access to sensitive data by embedding exploits within the payload. Furthermore, SSL/TLS (HTTPS) is typically used to provide confidentiality, integrity and authentication support from endpoint to endpoint at the transport level only. The downside of depending only on SSL/TLS (HTTPS), is that for more complex web interactions, such as web services forwarding messages to multiple intermediaries simultaneously, it is quite impossible for all intermediaries in the chain to have the necessary certificates to decrypt the message, process it, encrypt the message again, and send it over SSL to the next intermediary in the chain. Since SSL/TLS was more or less designed to be an end-to-end protocol, another risk in using SSL/TLS for SOAP transactions is that this could potentially allow an intermediary service to read or alter the SOAP message before relaying it through SSL/TLS (HTTPS)

again. Moreover, well-known vulnerabilities at the transport layer pose threats to the confidentiality and integrity of the message. Common attacks at the transport layer include man-in-the-middle attacks, replay attacks and session hijacking. Lastly, additional drawbacks are that SSL is not able to encrypt specific parts of the message, nor can it use different keys, which adheres to more of a defense-in-depth security model.

Although not as mature as transport level security, message level security has begun to receive much needed attention. In fact, several message level security standards, e.g. XML Encryption, XML Signature, WS-Security, have come about as a result of threats posed at the message level. Since SOAP was not originally designed with security in mind, without the proper security protections, SOAP messages are vulnerable to capture and modification by attackers as they traverse the Internet. Security challenges inherent to SOAP include:

- No authentication between SOAP endpoints or intermediaries.
- No way to verify the origin of a SOAP message
- No mechanism for ensuring data integrity or confidentiality either at rest or in transit.
- No mechanism for preventing SOAP messages from being resubmitted.

Vulnerabilities at the message level are similar to those of TLS. Typical attacks include message capture and replay attacks, message tampering, brute-force decryption, weak cryptanalysis, and message eavesdropping. Additional

risks include access to the web service by unauthenticated users (authentication), access of the service by unauthorized users (authorization), and un-audited use of the web service (accountability).

As we peer deeper into the application layer, there is a plethora of security vulnerabilities to consider when deploying web services. Typical application layer security vulnerabilities include, but are not limited to:

XDoS/XML Bombs/Recursive Payload - Because of the resources required for the XML parser to process XML, it is much easier for the attacker to create and transmit malicious XML than it is for the defender to process and reject the XML. XML supports rich and complex document structures, including recursion, which can potentially cause infinite loops during input processing. While IP-based denial of service (DoS) attacks usually require large numbers of messages, an XDoS attack can be launched on a Web service with a single 2KB malformed XML message.

WSDL Scanning - Typically, using an automated tool, a hacker attempts to retrieve and run through all of the operations in the WSDL document to gain information. Modern Web services use WSDL and UDDI to share information about one another so that services may dynamically bind to one another at run-time. The WSDL of a web service openly reveals the entire application programming interface (API) of the Web service-even parts that are disabled or to be used strictly for debugging purposes. Too much information can also provide information about the design and security requirements of a Web service, which can lead to successful attack or to privacy violations. UDDI registries openly provide details about the purpose of a Web service as well as how to access it.

Parameter tampering - An attacker modifies parameters in a SOAP message in an attempt to bypass input validation in order to access unauthorized information.

Replay attacks - An attacker attempts to resend SOAP requests to repeat sensitive transaction and overload the web service, similar to a network 'ping of death'.

Oversized payload attack - An attacker attempts to perform DoS by sending large XML document messages designed to overload the XML parser and deplete system resources.

External reference attack - An attacker attempts to bypass protections, such as XML validators, by including external references that will be downloaded after the XML has been validated, but prior to its being processed by the application.

Schema poisoning - An attacker attempts to compromise the XML schema in its stored location such that the XML validator will use the compromised schema.

SQL Injection - An attacker provides specially crafted parameters that will be combined within the Web service to generate a SQL query defined by the hacker.

Buffer overflows - An attacker provides specially crafted parameters that will overload the input buffers of the application and crash the Web service-or potentially allow arbitrary code to be executed.

Routing Detours - An attacker attempts to misdirect a SOAP message by overtaking one of the intermediaries assigned within the WS-Addressing routing instructions and inserting bogus routing instructions to point a sensitive document to an unauthorized location or routing it to a non-existent location and causing a denial of service.

Malicious content/attachments - An attacker sends a SOAP attachment, such as an image, executable or application-specific document, which contains malicious code or data, such as viruses or Trojan horse programs that are transmitted within a valid XML messages.

XPath/XQuery Injection - An attacker passes a malicious parameter to a web service in an attempt to change the semantics of an XML-specific filtering engine to access unauthorized information.

XML Injection - An attacker is able to insert XML tags into an XML document as a result of improper input validation. The inserted XML tags may allow an attacker to alter the structure of the XML document such that subsequent processing is affected and the application behavior is modified.

4. Web Service Security Requirements

Based on the above threats to web services, it is clear that several security requirements need to be met to mitigate the risks of exposure to web service attacks. Security administrators in particular need to look beyond the obvious and be proactive in identifying exposure points and attack targets via XML parsers, SOAP processors and WSDL-enabled applications. [5] Additionally,

having the ability to manage web service policy from a single point of control is recommended to reduce the risk of errors and omissions that are likely to manifest when configuring security at the application level for each web service.

Adherence to web services technology standards is also a recommended practice due to the fact that they have undergone a great deal of research and scrutiny before being adopted into the industry. [6] Compliance to the OASIS WS-Security specification, for example, is considered much safer than implementing custom security into the web services application. Moreover, a custom implementation simply will not interoperate with other systems, which may use a different standard. Another standard to consider is the WS-I Basic Security Profile (BSP) for interoperability, which restricts the W3C and OASIS standards in a manner that favors stronger security practices. The BSP also provides a list of security considerations to consider when deploying web services.

According to the National Institute of Standards and Technology (NIST), the following are elemental security requirements on which web services should be founded:

- Verify that principals (humans or application components) are who they claim to be through appropriate proof of identity. Determine the identity or role of a party attempting to perform some action, such as accessing a resource or participating in a transaction (Authentication).

- Determine whether some party is allowed to perform a requested action

or access particular resources, such as viewing a Web page, changing a password, or committing a transaction (Authorization).

- Log activity and any damage from successful attacks (Accountability).
- Ensure that the information is intact and that it has not been changed in transit, either due to malicious intent or by accident (Integrity).
- Produce or verify an electronic signature to prevent system users from later denying completed transactions (Non-repudiation).
- Ensure that only legitimate parties may view content, even if other access control mechanisms are bypassed and guarantee exchanged information is protected against eavesdroppers (Confidentiality).
- Limit access and use of Personally Identifiable Information (Privacy).

5. Enter the XML Firewall

XML firewalls are essentially high performance proxies, which perform security services such as authentication, authorization, auditing and XML validation at the message level. [7] They are used to protect back-end web services from XML-based threats. An XML security firewall is typically deployed behind an existing IP firewall, and secures all XML traffic before it reaches the Web service on the application server (see Figure 3). A few years ago, many of the products on the market today were labeled "XML firewalls"; however, the popular industry term now is "XML security gateways", because they are expected to do more than conventional firewalls. [8]

XML security gateways are stand-alone network appliances that go beyond a traditional XML firewall in securing XML and Web services. XML security gateways can typically terminate SSL/TLS sessions, validate XML Schema, perform XML encryption/decryption, sign XML/validate XML signatures, provide support for Extensible Stylesheet Language Transformation (XSLT), perform access control based upon SAML, route Web services requests based upon content, and perform firewall filtering. As a first line of defense, an XML gateway is the simplest and most effective way to enforce web services security.[1] Forrester Research recommends XML security gateways as standalone devices. A hardware-based XML security gateway has many advantages[9], including:

- *Performance*: An optimized hardware solution that addresses both XML acceleration and cryptographic processing will improve latency and throughput of XML security processing by a factor of at least 10 over software implementations. In many cases the performance improvements are considerably larger. IBM DataPower claims their appliance has wire-speed traffic performance.[10]

- *Scalability*: By deploying a high-capacity XML security gateway, the number of application server platforms may be significantly reduced. An XML security gateway can handle an increased Web services transaction load without needing to add additional application servers. Stand alone gateways also provide for ease of replacement.

[1] Although the first level of defense for Web services belongs on an XML security gateway, there are important cases where it makes sense to have a second level of defense on the Web services application platform. This paper focuses on XML gateways as a first level of defense.

- *Manageability*: By channeling all Web services traffic through a small number of high-capacity gateways, the number of security enforcement points is reduced. This simplifies the security configuration and makes changes easier to manage. For large implementations, appliances provide better value due to reduced maintenance costs and greater manageability is provided due to the centralized view of Web services rules and traffic.[11]

- *Simplicity*: An XML security gateway can enforce the majority of Web services security requirements, thus avoiding the need to write security code within the Web service applications.

- *Security*: An XML security gateway essentially virtualizes the Web service acting as an intermediary between the Web service (producer) and the consumer (requestor). Removing security from the back-end applications is a best practice, and improves the security assurance of the architecture. As in the case of an IP firewall, an XML security gateway is a hardened security platform that protects potentially vulnerable application servers.

- *Availability*: XDoS is a significant threat to Web service availability. An XML security gateway provides high-performance XDoS checking to protect Web services applications.

- *Interoperability*: Web services security standards and technologies are a moving target, and will continue to evolve. An XML security gateway is a natural place in the architecture to translate across multiple

transports and security standards.

- *Monitoring*: Because Web services traffic passes through the gateway, it provides an effective central enforcement point for audit logging and accountability. The gateway can also be configured to notify appropriate personnel when an attack has been attempted.

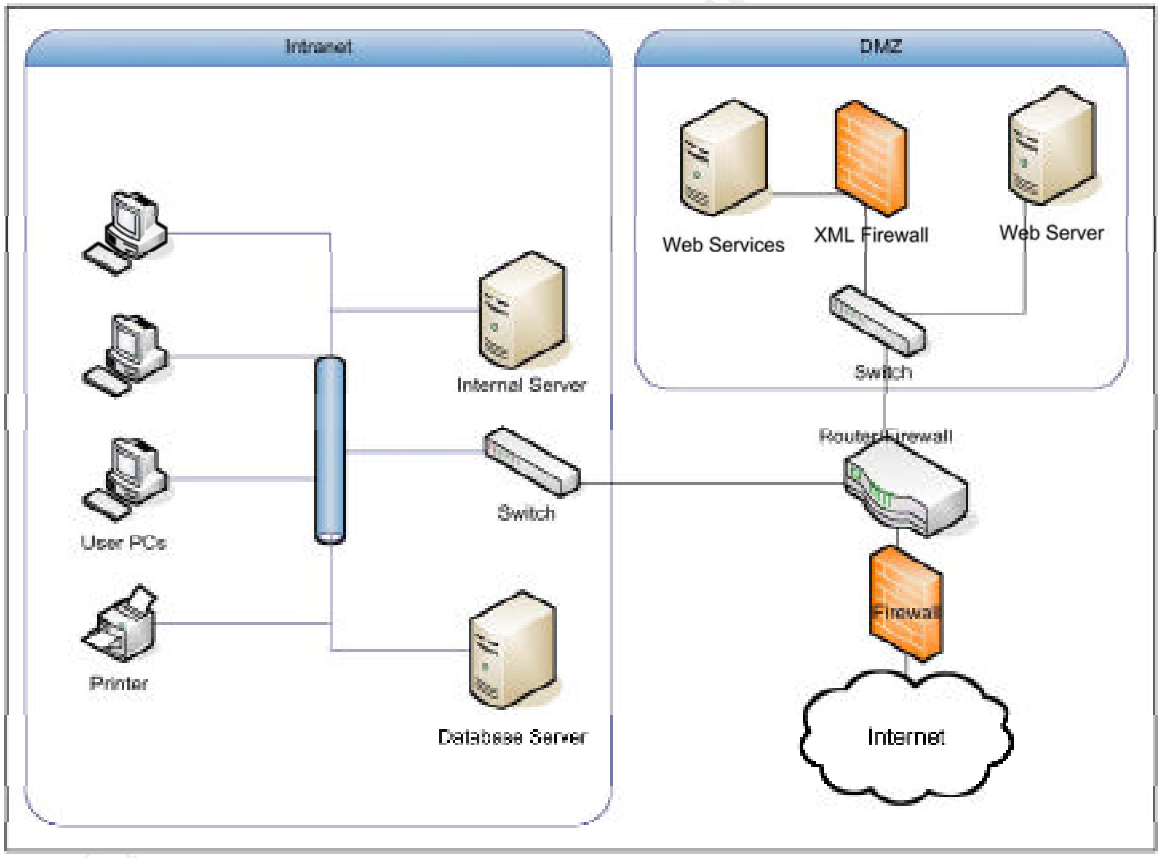

Figure 3. Stand-alone hardware XML Firewall appliance

The alternative to dedicated standalone hardware XML firewalls is the use of XML firewall software installed on servers (see Figure 4). Commercial software firewalls, such as those from Vordel and Forum Systems, can be installed on one server or deployed as agents on separate servers and managed

centrally. Although this approach may have a lower entry cost, the continuing maturity of web services makes maintaining consistent security standards and policies difficult. Also, from a strategic point of view, selecting such products may not offer the most flexibility as this market matures and vendors merge with larger companies. [12] Another option is to custom develop a software XML firewall, using a product such as the Microsoft Internet Security and Acceleration (ISA) Server 2000, which allows implementation of custom Internet Server API (ISAPI) filters. A model ISAPI filter, provided by Microsoft, can be used and customized to validate SOAP/XML messages at the ISA server.[13] Software XML firewalls can also be implemented on the servers providing the web service. For example, ModSecurity v2.0 is an open source product that can be implemented as a module on an Apache server. [14] Although this approach saves somewhat on hardware and infrastructure costs, it also increases the risk of performance slowdowns.

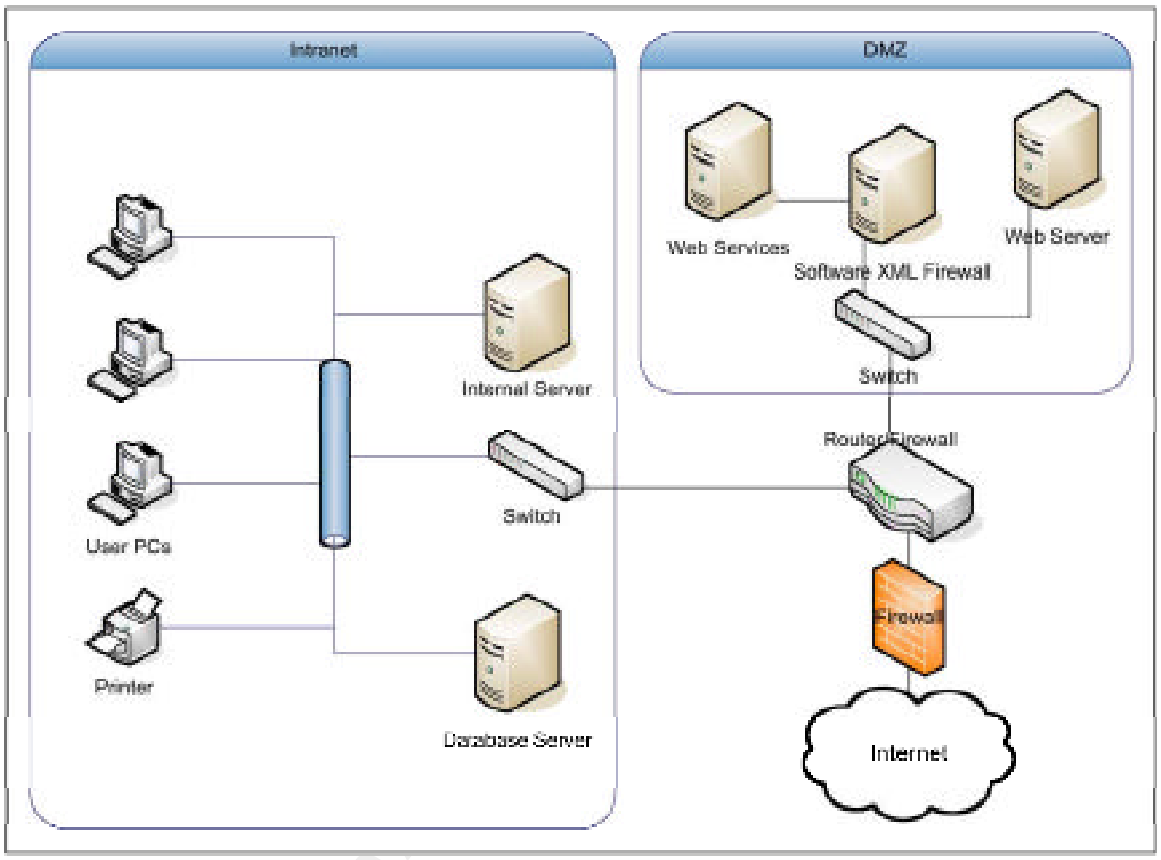

Figure 4. Dedicated XML firewall implemented by software

6. Best Practices for Deployment and Configuration

Deployment

The two most common deployments of an XML security gateway are (1) as a proxy within the enterprise De-Militarized Zone (DMZ) or (2) as business-to-business (B2B) deployment to integrate business partner infrastructures across an enterprise. [15]

In the DMZ configuration depicted in Figure 5, the XML security gateway protects the application server against Internet-based XDoS attacks and enforces

incoming access control, including authentication and authorization. The XML security gateway may also be deployed as a proxy to protect access within the corporate intranet.

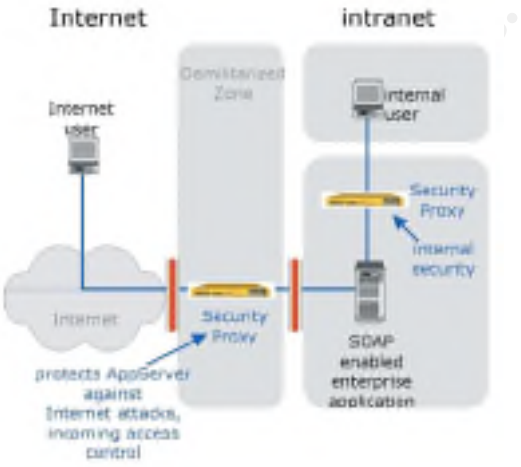

Figure 5 -XML Firewall DMZ Scenario

The more advanced B2B deployment, as depicted in Figure 6, requires that an XML security gateway also be installed on the Web services client side. In this scenario, the sender-side gateway provides outgoing access control, limiting the transmission of sensitive data to the Internet. The gateway can also be used to secure a federated extranet, where the Web services client and server environments do not share common security policies and mechanisms. To address federated extranet security, the XML security gateways can use SAML as a common standardized security token to map client-side security policy to server-side security policy. SAML is a framework for exchanging authentication and authorization information.

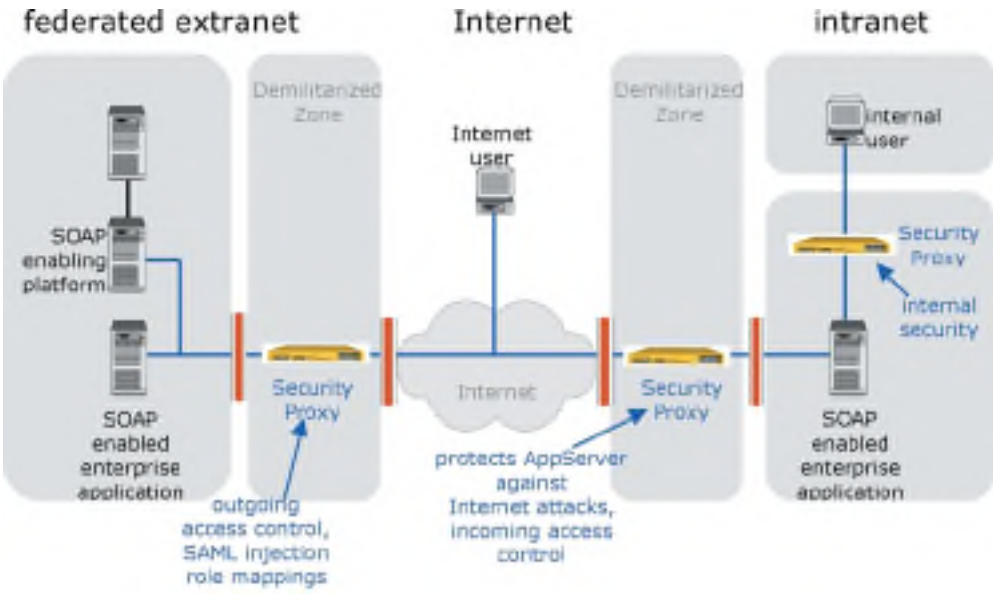

Figure 6 - XML Firewall Federated Trust (B2B) Scenario

According to Anne Thomas Manes, Vice President and Research Director[16], The Burton Group, when designing the architecture for an XML gateway and SOA management, a properly layered defense includes:

- Network perimeter defenses
- Identity-based defenses at centralized entry-point
- Identity-based defenses at each intermediary and endpoint
- Security monitoring for attack and fraud detection
- Transport-level and application-level message protections

The Burton Group also recommends that authentication be performed at the perimeter, at the same place where XML firewall processing occurs versus

performing XML content analysis at the perimeter and then performing authentication after the message has been validated as non-threatening. The risk with not performing authentication at the perimeter is that non-authenticated XML messages would then be granted access to the internal network. This provides an avenue for attacks such as XML port scanning and unauthorized access to resources on the internal network. At a minimum, authentication should be conducted at the transport layer at the XML Gateway, using WS-Security and SSL/TLS. An XML Gateway typically terminates client SSL connections, and therefore is in the logical position to perform authentication and drop messages that were not authenticated. Furthermore, SSL/TLS can also be used between the firewall and the Web service as an extra layer of protection, so that all communication between the Web service and the XML gateway is trusted.

Additionally, it would not be practical to separate security services and identity management for Web Services from the security service and identity management infrastructure, such as Public Key Infrastructure (PKI), IP Firewalls, IDS, Single Sign-On, LDAP, log server, and so on, already in place. For example, if an organization has already standardized on LDAP for authentication, it would not be practical to configure passwords to be stored on an XML Gateway appliance. The XML gateway should be configured to complement the existing management and security architecture in place at an organization.

Configuration

Although not a comprehensive list, the practices below can be a solid starting point to consider when configuring an XML firewall or stand-alone gateway appliance. Notable, these practices were collected from leading Fortune

500 companies across numerous industries. [17]

- *Secure the transport layer (SSL/TLS).* Deny un-trusted endpoints access to sensitive systems.

- *Mask internal resources using NAT.* Network Address Translation (NAT) is used to obscure internal IP addresses.

- *Mask internal web service details using service virtualization.* Service virtualization presents a different view of a Web Service to the user, compared to the actual view of the Web Service on the internal network. This can be useful in order to mask the internal details of a Web Service, such as its platform (e.g. a name ending in ".asmx" gives away the fact that the platform is .NET, whereas an XML Gateway can hide such details). Service virtualization also makes it easier to migrate Web Services to other platforms later.

- *Transform messages using XSLT.* Using XSLT, businesses can obscure internal schemas and object layouts from outside parties.

- *Protect against XML Denial of Service (XDoS) attacks by enforcing document limits.* Impose limits on parsed XML documents based on specified maximum depth of element nesting and maximum number of attributes allowed per element.

- *Validate all XML messages for well-formed structures using XML schema validation.* Develop robust local XML Schemas for the web service. Configure the XML parser to validate all incoming XML traffic against

the local Schemas rather than against remote Schemas provided by incoming traffic.

- *Perform XML filtering utilizing XPath-based content filters on inbound/outbound messages*. Start by setting up simple filters based on message size or XML Digital Signatures. [18] As application usage increases, filter based on content and other parameters, such as IP, SSL, HTTP headers and other metadata.

- *Protect message content* using digital signatures, timestamps and message encryption. Encrypt individual XML documents and data fields within documents with different encryption keys.

- *Utilize strong encryption algorithms* (RSA/DSA/3DES/AES) 128-bit or more, to sign, timestamp and encrypt all messages.

- *Enforce access control* (Authentication, Authorization, Accountability) via strong authentication, certificate-based authentication, Single Sign-On (SSO), SAML, role-based access, monitoring, logging, real-time alerts.

- *Centralize management*. Manage configurations across multiple devices from a central interface to reduce administrative costs and burden.

- *Ensure high performance and reliability* via load balancing, hot-standby failover support and configuration rollback.

- *Implement secure auditing* - Securely store audit/security event log

data using a secure distributed logging facility such as syslog. Use digital signatures and time stamping, to create secure e-business transaction logs that can be used for non-repudiation. Ensure logging of administrative actions taken on the XML gateway device.

A nice feature that comes with the IBM® WebSphere® DataPower XML Security Gateway XS40 is an XML Threat Protection option which can be enabled via a wizard or by editing an XML Firewall Service's configuration file. The XML Threat Protection feature protects against common threats such as XDoS, message tampering, SQL injection and XML Virus. "Beneath the hood" so to speak, is DataPower's proprietary XG3 XML acceleration technology, which boasts wire-speed performance. The device also has a separate management port and serial-console access. Additionally, the XS40 can encrypt an entire response or a single element within an XML document and perform transformation of XML through the application of XSLT.

There are several industry resources to consider when configuring or testing an XML firewall or gateway. In January 2006, the Web Application Security Consortium (WASC) published a document entitled *Web Application Firewall Evaluation Criteria (WAFEC)*. Although this document is intended to provide a set of web application firewall criteria that can be used to test or evaluate an XML firewall solution, it can just as easily be used as a guide for appropriately configuring an XML firewall. In August 2006, the National Institute of Standards and Technology (NIST) published a draft special publication (SP 800-95) to provide guidance and recommendations on securing web services. The document assists in providing an understanding of information

security practices that should be considered in SOA design and development based on web services and refers to the use of an XML gateway as means to provide a robust defense against XML attacks. Appendix B of SP 800-95 provides a list of common attacks targeting vulnerabilities found in web services, which can be used as a guide for mapping security requirements to XML security gateway configuration options.

7. XML Firewall Security and Compliance Testing

Now that we understand security threats to Web services and best practices for configuring the XML firewall, we need to examine techniques for establishing a standardized process to test the XML firewall for validating the correct operation in response to attempted attacks. Since an XML gateway acts as the Web service and forwards all communication to the internal Web service, testing XML firewalls for security is similar to testing web services for security. It requires an understanding of the threats web services face and how vulnerabilities can be identified at various levels (e.g. transport level, message level, application level) and then exploited. [19] To start, security staff should ensure that procedures are developed to ensure an effective web services security and compliance testing process is established, has management buy-in and is conducted regularly based on the security category of the network device. According to Table 3.2 of NIST 800-42 – Guideline on Network Security Testing[20], an XML gateway device would be considered a Category 1 system and therefore a penetration test should be conducted at least annually and possibly even more frequently as would be determined by the risk the device presents if it were to be breached and the cost of testing.

Testing an XML web service presents a much more difficult challenge for a penetration tester than for more traditional applications and requires a different skill set in XML, SOAP, WSDL's and other Web Services standards, in addition to experience in security issues unique to web services. Since the tester can make no assumptions as to how a web service is implemented, reliance on black box security testing is much greater than reliance on any other testing approach. [21] The ideal strategy for a penetration test is to test all WSDL interfaces and try out every possible input at least once to ensure it didn't cause an unexpected security violation.

In an article published by SYS-CON Media[22], Dr. Adam Kolawa relates some best practices for web services penetration testing, which can equally be applied to XML gateways. He states:

> "Understanding the security threats lets the tester design tests that can expose them with the help of good tools. For example, external entity attacks and XML bombs can be thrown at the service to see if the service refuses to process XML processing instructions or DTDs by returning a SOAP fault. WSDL access vulnerabilities can be detected by attempting to get a WSDL without the expected security channel if it's protected. For example, if the WSDL is protected with client-side SSL on port 443, it shouldn't be accessible on port 80; it's possible to forget an open connector in the Web server, which leaves multiple open channels. When it comes to thwarting WSDL scanning threats, then it's important to inspect the WSDL for redundant artifacts such as schemas or unused message definitions.
>
> Capture and replay attacks can be simulated by sending multiple requests with the same message identifier that determines its uniqueness. For example, if you're using Username Tokens, you should test the service by sending multiple messages with the same nonce values and verify that the service rejects such requests properly. The service should implement a sufficient, but limited cache size for the recently accepted nonce values. Many WS-Security implementations don't take this into consideration by default, which makes them vulnerable to capture and replay attacks.
>
> To test a Web Service's vulnerability to DoS attacks caused by heavy loads, such DoS attacks should be simulated in

fashion that's suitable to Web Services. You can't tell if a service can sustain a certain load scenario unless such a scenario has been tested. However, it's important to execute such load tests in a manner that's effective.

Some test engineers have the tendency to do load tests with the same static request to generate a load. Although this is a viable test scenario, it's not sufficient because such DoS attacks can be detected by network security appliances. Therefore, Web Service DoS attack simulations should be generated with dynamic request values that are semantically valid and can exercise wider code coverage in the Web Service's application logic to test the Web Services to its limits. Such attacks are difficult to generate by manual coding, but they're possible with load testing tools that are specialized for Web Services. In fact, the mere existence of such tools should alert Web Service engineers that such attacks can be done easily by a hacker if such tools fell into their hands. For example, to test a Web Service that accepts Username Tokens with timestamps and nonces, it's important to apply a load on the service where the timestamps and nonces are dynamically generated for each request. Otherwise, errors such as the ones caused by concurrency problems would go undetected. Another example would be load tests that send signed requests, where the hash and signature values should differ from one message to another.

Not only should Web Service load tests generate dynamic requests, but such tests should also simulate real-use case scenarios or usage patterns. For example, a use case scenario could be a Web Service client retrieving an authorization token (such as a SAML assertion) from a security authority, then using that token for subsequent Web Service invocations on different services. To test that scenario, load tests that keep using the same authorization token over and over again don't represent the real-world scenario since a real-use scenario would have multiple users requesting and using multiple tokens at the same time. Executing such a realistic load test can expose concurrency or scalability problems that result in vulnerabilities. In this example, it's possible for the Web Service to reject valid requests or accept unauthorized ones under a certain load even if such problems don't occur during regular functional testing.

To detect invalid responses during a load test, the load tests should be backed with sufficient response validations that ensure the detection of regressions from the correct behavior, because it's difficult to verify that all requests were met with the correct responses unless regression detection was done while the load is being generated. Without response validation, only network connections and HTTP errors would be exposed, which doesn't provide sufficient test coverage. For example, responses can be well-formed SOAP messages but with invalid data, or perhaps they contain an error message when they shouldn't. Without placing sufficient response validation during a load test, such incorrect responses can go undetected."

XML port scanning can be conducted to test for an inadequately configured XML gateway. This technique utilizes an XML parser to execute port scanning of

systems behind a restrictive perimeter firewall and can provide detailed information about systems within an organization.[23] Specifically, DTD external entity references can be created within an XML document that refer to ports on systems behind the XML gateway.

Another important issue to consider is testing compliance to standards such as WS-Security, SAML, and the Basic Security Profile (BSP) from WS-I. The BSP favors strong security practices and even lists a number of useful security considerations that should be taken into account when deploying secure Web services using WS-Security.

As was mentioned, the WASC has also developed the WAFEC document, which can be used by any reasonably skilled engineer to independently assess the quality of a web application firewall solution. According to Network World, the consortium is also considering plans to publish an application that simulates attacks on applications for potential customers to use to test individual Web Application Firewall (WAF) products. Their initial approach was to build a tool where customers can set up a test network with a PC running the attack simulator and a server being attacked on which WAF devices can be installed to see how well they protect the server. [24] More recently, in speaking with Ivan Ristic, project leader of the WAFEC, he indicated that an alternative approach could be to build a tool based on a database of web application vulnerabilities that performs several tests against a web server protected by a WAF and give marks based on the success/failure rate.

Tools

A number of tools that may be useful in conducting a web service penetration test include, but are not limited to:

- A HTTP proxy with SOAP extensions
- A scriptable SOAP generator
- A decoding/encoding library
- An SSL/TLS cipher specification enumerator
- A web method blindcrawler

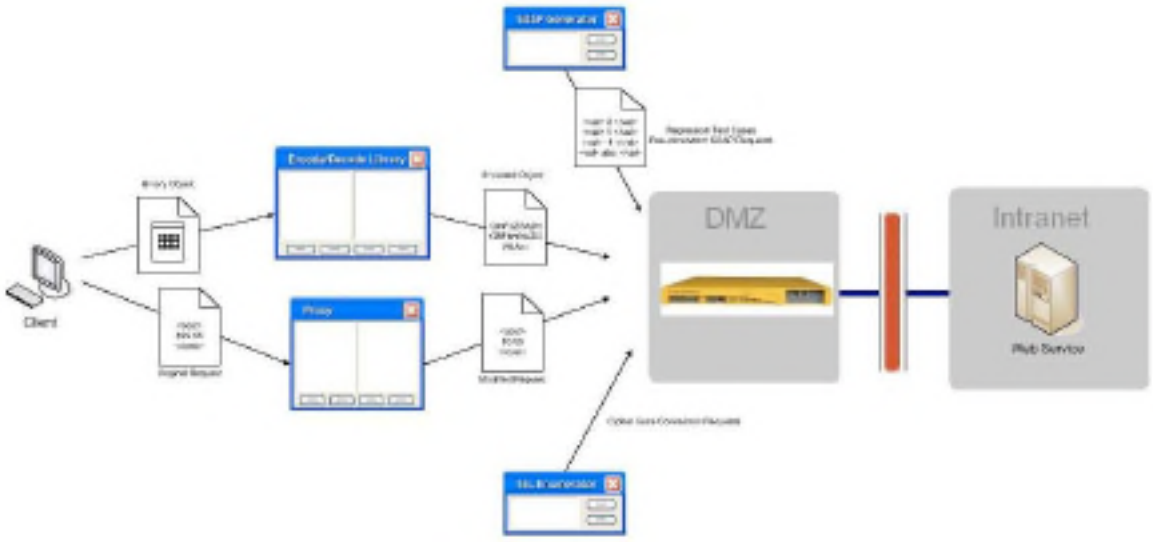

Figure 7 - XML Gateway Testing Tool Suite

Although several tools can be used together as a suite to automate security testing, there is no truly comprehensive tool suite available for automated web services security testing, not to mention that there is only a limited number of automated software and application security test tools in

general. [25] One useful product for XML security gateway testing is the Vordel SOAPBox tool, which can be used to test the defenses of an XML gateway and SOA security solutions. Although there are only a limited number of diagnostic and compliance test tools available, these tools can also be useful to exercise XML gateway functionality and adherence to security and interoperability standards. The automated tools listed below may prove useful for supporting security testing against an XML security gateway.

Web Services Penetration Testing

WSBang/WSMap - http://www.isecpartners.com/wsbang.html
http://www.isecpartners.com/wsmap.html

WSDigger - http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/s3i_tools.htm

SOAPBox - http://www.vordel.com/soapbox/index.html

Curl - http://curl.haxx.se/download

Decoding/Encoding Library

OWASP CAL9000 - http://www.owasp.org/index.php/Category:OWASP_CAL9000_Project

HTTP Proxy

Achilles - http://www.mavensecurity.com/achilles

The Peach Fuzzer Framework - http://peachfuzz.sourceforge.net/

OWASP WebScarab -
http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

Fuzzer

OWASP WSFuzzer - http://www.owasp.org/index.php/Category:OWASP_WSFuzzer_Project

The Peach Fuzzer Framework - http://peachfuzz.sourceforge.net/

WSFuzzer - http://sourceforge.net/projects/wsfuzzer

SSL

Foundstone SSL Digger -
http://www.foundstone.com/resources/proddesc/ssldigger.htm

Testing for SQL Injection

OWASP SQLiX - http://www.owasp.org/index.php/Category:OWASP_SQLiX_Project

Sqlninja: a SQL Server Injection&Takeover Tool - http://sqlninja.sourceforge.net

Bellucci: sqlmap, a blind SQL injection tool - http://sqlmap.sourceforge.net/

Absinthe 1.1 (formerly SQLSqueal) - http://www.0x90.org/releases/absinthe/

SQLInjector - http://www.databasesecurity.com/sql-injector.htm

Web Method Crawler

SIFT Web Method Search Tool - http://www.sift.com.au/73/171/sift-web-method-

search-tool.htm

The proceeding sections map NIST 800-95 recommendations for securing Web services to secure configuration standards and security testing procedures for XML Firewall gateway appliances. This matrix should assist security professionals in securely configuring XML firewall gateways and then regularly testing for compliance against these standards. Each configuration standard and associated testing procedures are mapped to NIST recommendations and the exact reference within the Special Publication (SP) NIST 800-95, *Guide to Secure Web Services*.

Integrity and Availability Requirements

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Prevent processing (by application server) of malformed SOAP/XML messages and/or SOAP headers	B.4.2 - Coercive Parsing B.5.2 - Recursive Payloads Sent to XML Parsers B.5.3 - Oversized Payloads Sent to XML Parsers	Validate inbound and outbound XML messages based on the rules of SOAP specifications and XML well-formedness	1. Make SOAP Request to the XML gateway that does not comply with the WS-Security schemas to determine the response. 2. Verify that WS-Security compliant SOAP requests are required for the XML gateway to service requests.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Prevent processing of SOAP/XML messages that do not match schema specifications of the specific Web service application	B.7.2 Parameter Tampering	Configure gateway to perform basic validation of inbound and outbound messages based on the particular schema of the specified Web service	1. Make an HTTP POST call with an invalid schema to the XML gateway on the port allowing XML/SOAP requests and confirm that an error message is returned.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against coercive parsing attacks	B.4.2 - Coercive Parsing B.5.2 - Recursive Payloads Sent to XML Parsers B.5.3 - Oversized Payloads Sent to XML Parsers	Inspect message content to block malformed and illegitimate requests and enforce document size.	1. Submit malformed requests such as SOAP messages with: • Additional or missing angle brackets • Additional or missing closing tags • Missing required attributes • Additional or missing quotes

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against parameter tampering attacks	B.4.1 Parameter Tampering B.7.2 Parameter Tampering	Configure gateway to validate message to ensure that parameter values are consistent with the defined WSDL and schema specifications for Web service	1. Proxy communications between client and server, and systematically alter SOAP elements in requests to determine responses. 2. Attempt to insert values not permitted by the XML gateway such as out-of-range or random values and assess the gateways' responses returned.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against recursive payload attacks	B.5.2 - Recursive Payloads Sent to XML Parsers	Impose limits on parsed XML documents based on specified maximum depth of element nesting and maximum number of attributes allowed per element. Utilize XML Schema validation, message monitors to check thresholds on message rates, and content-based filters to detect recursion depths and other complexity measures of an XML document.	1. Send a deeply nested SOAP request to the XML Gateway.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against oversized payloads	B.5.3 - Oversized Payloads Sent to XML Parsers	Impose limits on parsed XML documents, i.e., document size, depth of element nesting, and number of attributes allowed per element.	1. Send a well-formed, yet overly large, SOAP request to the XML Gateway.
Protect against schema poisoning attacks	B.4.3 Schema Poisoning	Use URL Rewriting rule for Web services virtualization to hide the internal URL of Web service (map client request to protected back-end servers)	1. Attempt to access and tamper with schema documents to remove type checking, add/remove request elements or delete schemas altogether. 2. If successful, generate SOAP requests that comply with the new schema that contains malicious elements and gauge server response.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against Web service WSDL scanning attacks	B.1.4 WSDL Scanning	Use URL Rewriting for Web services virtualization to hide the internal URL of Web service Filter messages based on Web Service name or URL	Common web service platforms have well known naming conventions for WSDL documents. Attempt to retrieve WSDL documents from URLs including variations on the following: ·http://webservice-url/ServiceName?WSDL ·http://webservice-url/ServiceName.asmx?WSDL ·http://webservice-url/ServiceName.cfc?WSDL ·http://webservice-url/ServiceName.dll?WSDL ·http://webservice-url/ServiceName.exe?WSDL ·http://webservice-url/ServiceName.php?WSDL ·http://webservice-url/ServiceName.pl?WSDL ·http://webservice-url/ServiceName.m?WSDL

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against routing detours	B.4.7 Routing Detours	Enable WSDL virtualization to enforce strict routing behavior.	1. Send a SOAP request to the XML Gateway that includes routing instructions to a non-existent node.
Protect against external entity attacks	B.4.8 - External Entity Attack	Suppress external URI references to protect against malicious data sources and instructions; rely on well-known and certified URIs or even on box schemas and WSDLs	1. Send a SOAP request to the XML gateway with a reference to an external source. 2. Send a SOAP request to the XML gateway with a reference to a file that can possibly be de-referenced and its' contents returned (e.g. file:///etc/passwd) 3. Send a SOAP request with a series of overly large reference strings.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against SQL injection attacks	B.6.1 - Structured Query Language (SQL) Injection	Enable SQL pattern matching checks	1. Supplying a single quote in every possible parameter via an attack proxy. 2. Test some SQL reserved words. If successful, this will create a syntactically invalid SQL statement and cause an error or exception to occur. 3. Utilize automated SQL Injection tools
Protect against malicious content attacks, in which hackers imbed viruses, worms, or other malware as payload to SOAP messages	B.7 - Malicious Code Attacks B.7.5 - Malformed Content	Enable virus detection through interaction with an external virus scanning system to detect malicious content	1. Test how the XML gateway responds to an unexpected attachment (e.g. an executable sent in an attachment).

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against replay attacks	B.7.4 - Session Hijacking 3.1.2 - Security Concerns of WS-Security	Apply a filter that can detect a replay attack. The filter should guard against WS-Addressing MessageID and WS-Security UsernameToken Nonce elements as well as any other element in the message via Xpath. Implement a timestamp enforcement option for replay attacks against digitally signed messages.	1. Capture and replay previous messages to the server to test whether or not the XML gateway accepts them. 2. Attempt to repeatedly perform important actions such as financial transactions or authentication exchanges. 3. Check for the presence of nonces, timestamps or other measures of 'freshness' in SOAP messages and assess the ability to manipulate or bypass them. 4. Confirm that 'freshness' measures are protected against tampering.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against incoming messages from unauthorized IP address	B.4.6 Principal Spoofing	Configure network access control to accept incoming messages from a specific IP address or range of addresses, and/or to reject incoming messages from a specific IP address or range of addresses	1. Attempt to spoof a request from a legitimate client. 2. Determine if any form of address filtering is in place by testing from different points in the internal and external networks. 3. Attempt to proxy or tunnel requests through a valid node for an unauthorized node.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against unauthenticated/ unauthorized users access	2.2 - Elements of Security 2.3.2 Protecting Resources 5.2 - Authorization and Access Control in Legacy Applications B.2.2 - Exploiting Unprotected Administrator Interfaces	Perform Authentication, Authorization, and Auditing (AAA) action based on validating user identity via an authentication service such as LDAP, Active Directory Services, etc. Allow all authenticated users to access the Web service or implement role-based authorization.	1. Examine message formats to determine how credentials are presented to the XML gateway and attempt to create similar but unauthorized credentials. 2. Test if one factor of authentication can be independently used with another. 3. Attempt to enumerate access control lists, identify any deviations of ACL's to role definitions, and map the extent to which access to unauthorized data or functionality is granted. 4. Attempt to use unauthorized functionality such as user administration facilities to gain further access to the system. 5. Attempt to gain unauthorized access to remote management console.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect against requests made to unidentified Web services	B.1.2 Forceful Browsing Attack	Filter messages based on Web service name, or Web service URL	1. Make repeated requests to the XML gateway with the URL patterns of typical Web application components, such as CGI programs. 2. Observe the XML gateway responses. 3. Try to interpret error messages that are returned for information that would be of use to an attacker.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Protect administrative interfaces	B.2.2 - Exploiting Unprotected Administrator Interfaces	Configure ACL correctly. Configure SSL correctly to perform client/server authentication and properly separate administrator session/tunnels from user sessions/tunnels. Configure gateway to log error messages securely and send to administrator via an SSL-encrypted connection.	1. Attempt to gain less-than administrator privileges and if successful, attempt to elevate privileges to administrator level. 2. Attempt man-in-the-middle attack on administrator session. 3. Invoke error messages and review responses for potential 'reconnaissance' information. 4. Intercept error messages and ensure they are encrypted.

Confidentiality Requirements

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Provide transport layer encryption in the form of SSL	3.1 Service-to-Service Authentication 3.4.1 Transport Layer Confidentiality and Integrity: HTTPS	Configure SSL usage to encrypt the entire application payload and provides one- or two-way authentication between client and gateway. Utilize SSL/TLS for connections between the gateway and the web service to ensure added protection.	1. An assessment of the classes of cipher specifications supported should be performed and weak cipher support should be noted. Weak cipher specifications are considered to include all specifications that use no encryption, DES, RC2, anonymous Diffie-Hellman, or have symmetric key lengths of less than 128-bit. 2. Assess what items are encrypted. Seemingly innocuous unencrypted items can provide attackers useful information.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Provide WS-Security in the form of message encryption/decryption, and message signing and verification	3. Web Service Security Functions and Related Technologies 3.1.1 WS-Security for Authentication 3.4.2.1 XML Security Standards 3.5.2 Non-Repudiation of Web Service Transactions	Configure XML gateway to encrypt or decrypt all or part of a message. To encrypt or sign an entire XML message, or specific message elements, there are many considerations that need to be determined: • Credential choices and how they are to be passed • What part of the message needs to be encrypted • PKI certificates • Provisioning authority of PKI certificates	1. Attempt to extract usable information from unprotected portions of the communications stream and analyze what is disclosed. Significant items to look for include message integrity check fields, signatures, timestamps and nonces as these can disclose information about the data contained or other protections in place. Specifically, if the XML Encryption standard is utilized, identify the data included within the <xmlenc:EncryptedData> tags and the remaining unencrypted data within the SOAP body. 2. Determine if digital certificates are verified with the issuer during the authentication process. Attempt to supply a valid certificate from another issuer, self-generated certificate in place of an authentic certificate, or an expired (but valid)

Logging and Auditing Requirements

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Provide real-time monitoring of messages and create message and system logs.	3.5.1 Audit in the SOA Environment	Configure XML gateway to generate log messages during normal operation. These messages range everywhere from the extremely verbose Debugging to the infrequent Critical or Emergency. Configure XML gateway to monitor and generate messages when counter thresholds or events are triggered.	1. Attempt to poison the log by inserting carriage returns, line feeds, null values, HTML tags, XML elements, excessive spaces, and unprintable characters. 2. Using white space characters, such as the tab and space, supply overly long messages, which will be used in logging. After the messages have been processed, examine the generated logs to identify how the white space was handled. 3. Repeatedly supply data to web methods that log their inputs until the log has reached its capacity. Once past capacity, analyze the resulting behavior of the logging mechanism.

NIST 800-95 Web Services Security Recommendations	NIST 800-95 Reference	XML Gateway Configuration Action	Security Testing Procedure[2]
Export system and error logs to a central management system	3.5.1 Audit in the SOA Environment	Configure XML gateway to send log files to various targets via SMTP, SNMP, SOAP, or the syslog protocol.	1. Attempt to analyze the processes employed in logging functionality and use them to gain access to log entries where possible. Logs and log entries should be tested in transit and in storage and attempts made to determine their contents.

[2] The majority of these assessment procedures are based on recommended web security testing techniques from SIFT Information Security Services publication "A Web Services Security Testing Framework" version 1.0 by Colin Wong and Daniel Grzelak, 10/11/2006.

8. Conclusion

While an SOA implementation based on XML and Web Services can offer various benefits, it can also introduce new avenues of attack for hackers to exploit critical business applications and processes. An XML firewall (or gateway) is an effective first line perimeter defense against XML and Web Services threats. However, best practices and industry standards need to be followed when it comes to the configuration and security testing and monitoring of such devices. This paper has discussed the building blocks of Web services, Web services threats and security requirements, configuration standards for an XML gateway device, and assessment procedures for ensuring the adequacy of this security control. Since security will always be a moving target, the deployment of an XML gateway should not be considered "bullet-proof". Rather than having an unrealistic sense of security, organizations should adapt this technology to their overall security strategy and architecture to ensure a more holistic approach for defending against the enemy at the gate.

9. References

[1] "XML Web Services" BindingPoint. Retrieved January 12, 2007, Website: http://www.bindingpoint.com/xmlwebservices.aspx

[2] "Introduction to SOAP" W3CSchools. Retrieved January 12, 2007, Website: http://www.w3schools.com/soap/soap_intro.asp.

[3] "Service-Oriented Architecture" 4js.com. Retrieved January 12, 2007, Website: http://www.4js.com/exclude/en/tmpl4/blocs/files/Info_World_SOA_Strategy_Guide.pdf

[4] "The three levels of SOA maturity" By John Dix. Network World. Retrieved January 12, 2007, Website: http://www.networkworld.com/columnists/2006/051506edit.html

[5] "Anatomy of a Web Services Attack" ForumSys.com. Retrieved September 27, 2006, Website: http://www.forumsys.com/papers/Anatomy_of_Attack_wp.pdf

[6] "Security Concepts, Challenges, and Design Considerations for Web Services Integration" By Howard Lipson and Gunnar Peterson. Retrieved January 12, 2007, Website: https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/assembly/639.html?branch=1&language=1.

[7] "Securing Web services: A job for the XML firewall" SearchWebServices.com. Retrieved January 12, 2007, Website: http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci955191,00.html

[8] MacVittie, Lori. "XML Gateways" Network Computing. Retrieved January 12, 2006, Website: http://www.networkcomputing.com/showitem.jhtml?articleID=161501098&pgno=1.

[9] Hartman, Bret. "Securing Your Enterprise Web Services in a Suspicious World." SOA World Magazine. Retrieved January 12, 2006, Website: http://webservices.sys-con.com/read/43958.htm.

[10] "WebSphere DataPower XML Security Gateway XS40" IBM DataPower. Retrieved January 12, 2006, Website: http://www-306.ibm.com/software/integration/datapower/xs40/index.html.

[11] Smith, Randy Franklin. "SOAP/XML Firewalls." WindowsITPro. September 2003. Penton Media, Inc. Retrieved January 12, 2007, Website: http://www.windowsitpro.com/Windows/Article/ArticleID/39755/39755.html

[12] Loftus, Jack. "Report recommends standalone XML security appliances." SearchSecurity.com. May 6, 2004. TechTarget. Retrieved November 3, 2006, Website: http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci962936,00.html.

[13] Smith.

[14] http://www.modsecurity.org

[15] Xtradyne.com. Retrieved January 12, 2007, Website: http://www.xtradyne.com/products/ws-dbc/WSDBCarchitecture.htm

[16] "Architecture for SOA Management XML Gateways and Web Services Management." Vordel. Retrieved January 12, 2007, Website: http://www.vordel.com/downloads/sdaolnwod/vordel_best_practices_paper1.pdf.

[17] "XML Web services security best practices." By Eugene Kuznetsov. ZDNet. Retrieved January 12, 2007, Website: http://news.zdnet.com/2100-1009_22-5345253.html.

[18] "Essential Web Services Security Practices". By Rich Salz. Retrieved January 12, 2007, Website: http://www.idealliance.org/papers/dx_xml03/papers/05-04-02/05-04-02.pdf

[19] OWASP.org. Retrieved September 27, 2006, Website: http://www.owasp.org/index.php/Web_Services_Security_Testing

[20] National Institute of Standards and Technology (NIST) Special Publication 800-42, *Guideline on Network Security Testing*, October 2003

[21] "Black Box Security Testing Tools" Cigital Inc. Retrieved September 27, 2006, Website: https://buildsecurityin.us-cert.gov/daisy/bsi/articles/tools/black-box/261.html?branch=1&language=1

[22] "SOA Best Practices - Four Steps to Securing Your Web Services." By Adam

Kolawa. SYS-CON Italia. Retrieved January 12, 2007, Website: http://it.sys-con.com/read/219090_2.htm.

[23] "XML Port Scanning - Bypassing Restrictive Perimeter Firewalls." By Wong Colin. Sift. Retrieved January 12, 2007, Website: http://www.sift.com.au/assets/downloads/SIFT-XML-Port-Scanning-v1-00.pdf.

[24] "Consortium hopes to lift Web application firewall confusion." By Tim Greene. Network World Inc. Retrieved January 12, 2007, Website: http://www.networkworld.com/news/2006/011306-web-application-firewall.html

[25] NIST SP 800-95 (Draft), *Guide to Secure Web Services,* September 2006

Last Updated: January 28th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS New York City Winter 2020	New York City, NYUS	Feb 10, 2020 - Feb 15, 2020	Live Event
SANS Northern VA - Fairfax 2020	Fairfax, VAUS	Feb 10, 2020 - Feb 15, 2020	Live Event
SANS London February 2020	London, GB	Feb 10, 2020 - Feb 15, 2020	Live Event
SANS Cairo February 2020	Cairo, EG	Feb 15, 2020 - Feb 20, 2020	Live Event
SANS Dubai February 2020	Dubai, AE	Feb 15, 2020 - Feb 20, 2020	Live Event
SANS Brussels February 2020	Brussels, BE	Feb 17, 2020 - Feb 22, 2020	Live Event
SANS Scottsdale 2020	Scottsdale, AZUS	Feb 17, 2020 - Feb 22, 2020	Live Event
SANS San Diego 2020	San Diego, CAUS	Feb 17, 2020 - Feb 22, 2020	Live Event
Open-Source Intelligence Summit & Training 2020	Alexandria, VAUS	Feb 18, 2020 - Feb 24, 2020	Live Event
SANS Training at RSA Conference 2020	San Francisco, CAUS	Feb 23, 2020 - Feb 24, 2020	Live Event
SANS Manchester February 2020	Manchester, GB	Feb 24, 2020 - Feb 29, 2020	Live Event
SANS Secure India 2020	Bangalore, IN	Feb 24, 2020 - Feb 29, 2020	Live Event
SANS Jacksonville 2020	Jacksonville, FLUS	Feb 24, 2020 - Feb 29, 2020	Live Event
SANS Zurich February 2020	Zurich, CH	Feb 24, 2020 - Feb 29, 2020	Live Event
Blue Team Summit & Training 2020	Louisville, KYUS	Mar 02, 2020 - Mar 09, 2020	Live Event
SANS Northern VA - Reston Spring 2020	Reston, VAUS	Mar 02, 2020 - Mar 07, 2020	Live Event
SANS Munich March 2020	Munich, DE	Mar 02, 2020 - Mar 07, 2020	Live Event
SANS Secure Japan 2020	Tokyo, JP	Mar 02, 2020 - Mar 14, 2020	Live Event
ICS Security Summit & Training 2020	Orlando, FLUS	Mar 02, 2020 - Mar 09, 2020	Live Event
SANS Jeddah March 2020	Jeddah, SA	Mar 07, 2020 - Mar 12, 2020	Live Event
SANS St. Louis 2020	St. Louis, MOUS	Mar 08, 2020 - Mar 13, 2020	Live Event
SANS Prague March 2020	Prague, CZ	Mar 09, 2020 - Mar 14, 2020	Live Event
SANS Paris March 2020	Paris, FR	Mar 09, 2020 - Mar 14, 2020	Live Event
SANS Dallas 2020	Dallas, TXUS	Mar 09, 2020 - Mar 14, 2020	Live Event
Wild West Hackin Fest 2020	San Diego, CAUS	Mar 10, 2020 - Mar 11, 2020	Live Event
SANS Doha March 2020	Doha, QA	Mar 14, 2020 - Mar 19, 2020	Live Event
SANS London March 2020	London, GB	Mar 16, 2020 - Mar 21, 2020	Live Event
SANS Norfolk 2020	Norfolk, VAUS	Mar 16, 2020 - Mar 21, 2020	Live Event
SANS SEC504 Nantes March 2020 (in French)	Nantes, FR	Mar 16, 2020 - Mar 21, 2020	Live Event
SANS San Francisco Spring 2020	San Francisco, CAUS	Mar 16, 2020 - Mar 27, 2020	Live Event
SANS SEC401 Lille March 2020 (in French)	Lille, FR	Mar 16, 2020 - Mar 21, 2020	Live Event
SANS Secure Singapore 2020	Singapore, SG	Mar 16, 2020 - Mar 28, 2020	Live Event
SANS Security East 2020	OnlineLAUS	Feb 01, 2020 - Feb 08, 2020	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced