Filtering to Inspect XML: an Operational Framework for Service Oriented Architecture Network Security

Robert Bunge[1], Sam Chung[1], Barbara Endicott-Popovsky[2], Don McLane[1]

[1]Computing & Software Systems
Institute of Technology
University of Washington, Tacoma
{rbunge, chungsa, dmclane}@u.washington.edu

[2]Center for Information Assurance and Cybersecurity
University of Washington, Seattle
endicott@u.washington.edu

ABSTRACT:

This study proposes a new operational framework for Service Oriented Architecture (SOA) network security. It seeks to characterize the current state of practices in SOA network security by gathering information regarding known threats and defenses for SOA deployments. It works towards the practical implementation of SOA designs by creating training and testing scenarios for those preparing to work in this area. Finally, it frames these and other SOA security efforts with respect to a classic theoretical model of information security. The resulting synthesis includes recommendations on how best to process the XML network traffic typical of SOA applications. The proposed approach is Filtering to Inspect XML (FIX) at the network's perimeter. This framework contributes to the understanding of secure SOA designs by clarifying the responsibilities of both network managers and software engineers in orchestrating XML-based services.

KEY WORDS:
SOA Network Security; Network Perimeter, Filter and Inspect

INTRODUCTION

The movement toward Service Oriented Architecture (SOA) is growing apace, with over half of major United States corporations reporting SOA projects, according to a recent information technology survey (International Data Group, 2006). Oracle, Microsoft, and BEA are among the major vendors providing tools and platforms in this area (Kobielus, 2006). SOAs feature many complexities, however, and new security threats are likely to emerge as organizations struggle to master the unanticipated implications of this emerging paradigm. Under SOA designs, distributed data traffic often bypasses typical network defenses such as firewalls and Intrusion Detection Systems (IDS). Such commonly deployed network perimeter systems rely on a TCP/IP packet filtering model to detect threats. Because SOAs utilize eXtensible Markup Language (XML) formats such as Web services encapsulated in application layer headers such as Hypertext Transfer Protocol (HTTP), packet filtering systems targeted below the application layer will not detect XML-based vulnerabilities. How then should network defenses evolve to engage the emerging vulnerabilities associated with XML?

Secure network designs often require that organizations perform security operations such as authentication and authorization at the network's edge. SOA traffic such as Web services may utilize XML-encapsulated data to establish trust or to deliver secure payloads. Such data must be scrutinized by perimeter systems sensitive to XML. Although systems such as XML firewalls or XML gateways are beginning to emerge, the purpose and use of such systems may not be well understood by network administrators. This paper proposes a new operational framework to elucidate required processes for XML network security. This framework contributes to the understanding of secure SOA designs by clarifying the responsibilities of both network managers and software engineers in orchestrating XML-based services.

An important goal for any model of SOA network security is the maximization of assurance. Assurance can be defined as steps taken to increase trust in an information system (Bishop, 2003). Security experts generally agree that no system can be made one hundred percent immune to failure, and any security model must trust, to some extent, that implementation measures adopted function as intended. Nevertheless, additional steps (such as system auditing) may be taken to increase total assurance in security design and implementation. A general hypothesis guiding this project has been that a coherent, clearly articulated model for SOA network security provides greater assurance than a series of disconnected or *ad hoc* measures. *Ad hoc* measures may succeed well technically against a variety of potential threats, yet because they are not easily characterized in a general way, there remains the suspicion that some critical area of vulnerability may have been missed. Moreover, security operations mechanisms do not stand alone in a system of assurance. Rather, the use of such mechanisms should logically flow from threat assessments, to policy articulation, to system design, to implementation mechanisms, and only then to operations and maintenance. Therefore, an overriding goal of this research has been to articulate a set of design principles or policy considerations to frame SOA network security in a general way, thus contributing to systemic assurance.

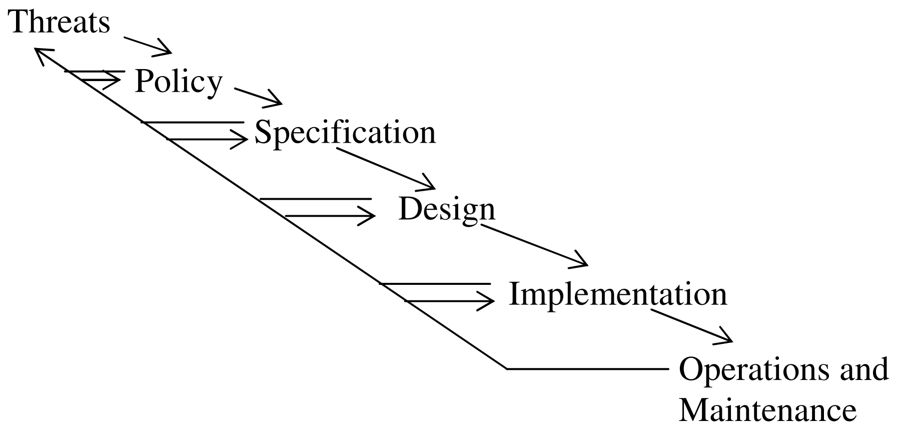


Figure 1. Security operations in context (Bishop, 2003)

PREVIOUS WORK

A growing body of research addresses concerns about SOA network security. Some of this literature focuses on threat assessment. Other sources analyze or recommend specific techniques for functions such as authentication, encryption, or verification of services. Still other works focus on high level modeling processes for engineering secure SOAs. Despite this growing research effort, however, practitioners of network security may remain somewhat at a loss when it comes to SOA. Vulnerability lists are very granular and only highlight security issues after they have emerged. Particular security operations, such as authentication or encryption, are necessary, but not sufficient for network security. High level modeling approaches provide a comprehensive view, but can be sketchy with respect to details. Each of these approaches has its strengths, but none provide network managers with guidance both comprehensive in view and succinct in practice. What follows is a selection of current research into SOA network security, illustrating both the gains already made and the gaps remaining to be filled.

Security Models and Goals

At the most abstract level, different security models may be needed to address different goals. Bishop (Bishop, 2003) provides a summary of the most common information security reference models. The Bell-LaPadula model, for example, is known as a military security model. This model is focused exclusively on confidentiality, with the main vulnerability addressed being the leakage of sensitive information. The Biba model is logically similar to the Bell-LaPadula model, but it focuses on an entirely different type of risk. Biba's is the prototype for commercial security models, because its main goal is data integrity. The problem for the Biba model is not so much to keep secrets, rather it is to assure that data is not modified without authorization. Various other high level models have been articulated, with commercial transaction security being an area often addressed.

One of these commercially focused approaches, known as the Clark-Wilson model (Clark & Wilson, 1987), has been selected as a basic framework for this study. One reason for this choice is because many Web services security specifications exist to promote transactional security. A more direct advantage of the Clark-Wilson model for this study is its key assumption of the need to "separate duties". Separation of duties implies that no one person or system should be trusted to undertake an entire transaction or even to engineer an entire transactional system. Rather, checks and balances should exist to verify transactional integrity. Auditing is one device to create such balances. Separate roles for system designers and system deployers is another. The Clark-Wilson model thus provides a natural starting point for this current analysis, for the simple reason that it offers a natural point of insertion for network administrators into the secure software lifecycle. If Clark-Wilson principles are adopted as a starting point for a security architecture, then network personal can perform implementation, verification, or auditing tasks as part of the needed separation of duties. The fact that separation of duties is a legal requirement for such significant commercial legislation as the Sarbanes-Oxley Act of 2002 adds additional impetus to this need for distributing information security roles throughout an organization (Contos, 2006).
Separation of duties, then, as defined in the Clark-Wilson model, will be a touchstone for assessing the operational utility of previous works on Web services security. Several general categories of such existing research are analyzed in the following section.

Type of Research	Granularity	Operational Impact
Threat Assessment Lists	Very granular	Many specific security operations required
Specific Security Techniques	Medium	Address specific classes of vulnerability
Software Engineering Models	Global	Implementation measures not specified in detail

Table 1. Summary of recent research into SOA security

XML Threat Assessment Lists

A starting point for assessing XML-based information security threats is examination of data from the major real time security notification centers on the World Wide Web. These include the Computer Emergency Response Team (CERT) Coordination of the Carnegie Mellon Software Engineering Institute (CERT, 2006), the SANS (SysAdmin, Audit, Network, Security) Institute (SANS Institute, 2006), and the National Vulnerability Database of the Department of Homeland Security National Cybersecurity Division (Department of Homeland Security, 2006). Queries on the vulnerability lists maintained by each of these organizations show multiple existing vulnerabilities related to XML, Web services, or SOA. Specific threat categories noted in these areas include XML Denial of Service, buffer overflow, and XML injection attacks. Although defenses against some of these types of threats are common at lower network layers, XML headers associated with SOA may contain attack payloads invisible to traditional perimeter systems. Real time monitoring of vulnerability lists such as these is a required practice for network security. However, granular response to such specific vulnerabilities falls short of a comprehensive, organized security design. To initiate more comprehensive design thinking about SOA network security, other sources must be consulted.

One such source for the assessment of emerging security vulnerabilities are reports by practitioners in information security. Stamos and Stender (Stamos & Stender, 2005) provide one example, detailing many specific threats to SOA and Web services designs. They group such threats into three layers—application, SOAP, and XML. Some specific attack vectors include targeting poorly implemented logic in applications, mining auto-generated WSDL files for vulnerabilities, and embedding malicious code in XML CDATA (Character DATA) fields. In general, as an integration framework, SOA encompasses all previous types of application security risk (like TCP/IP attacks or SQL injection), and it adds many more potential vulnerabilities on top of these. SOA security requires an understanding of how XML can mask otherwise well known types of malicious payloads. It also demands thorough scrutiny of XML itself.

Studies such as (Stamos & Stender, 2005) are useful for outlining the general scope of SOA network security issues. They also provide helpful advice on how to address various types of threats and vulnerabilities. What such overviews do not provide, however, is a succinct model to frame XML security issues with respect to SOA networks. Without a strong organizing principle, XML threat assessment literature remains very targeted on specific vulnerabilities, requiring many discrete security operations to address the many threats. Considering only sources such as these, it becomes incumbent upon network managers to orchestrate their own XML security designs, which may lay a heavy burden on practitioners already more than burdened with the escalating security risks associated with IP networks.

Moving toward more theoretical approaches, threats to Web services are also detailed in academic research. Beznosov et al. (Beznosov et al., 2005) note that Web services expose data that had previously been locked in corporate networks. This study outlines the structure of Web services architecture and shows how various XML specifications provide building blocks for Web services security. Epstein, et al. (Epstein et al., 2006) detail thirteen dangerous assumptions that can lead to insecure SOA designs. These include building proof-of-concept models that ignore security, putting too much weight on security standards and features, and misapplying vulnerability metrics. Holgersson and Söderström (Holgersson & Söderström, 2005) list several other areas of vulnerability for Web services. Such areas include: routing between multiple Web services, unauthorized access, parameter manipulation/malicious input, network eavesdropping and message replay, Denial of Service, bypassing firewalls, and the immaturity of the Web services platform. These authors conclude that the WS-Security specification is an "unfinished roadmap". This observation points to the need for more theoretical reflection regarding Web services security. In general, studies such as these have raised awareness of SOA network security problems and issues for further research, leaving much work yet to follow. Beyond such surveys and overviews, new research is indeed appearing, often with a much tighter focus on selected elements of SOA security.

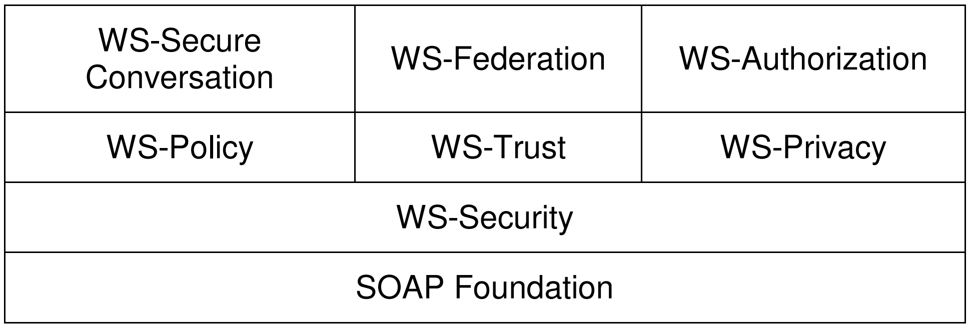


Figure 2. Web services security standards (Holgersson & Söderström, 2005)

Specific SOA Security Techniques

Much existing research into SOA security focuses on specific tools or techniques for performing XML message-layer security functions. Such mechanisms support dimensions of application security such as access control, confidentiality, integrity, authorization, and authentication. Frameworks and protocols to coordinate these functions are also analyzed, including emphasis on XML-based specifications such as XML Encryption, XML Signature, eXtensible Access Control Markup Language (XACML), and Security Assertion Markup Language (SAML). The WS-Security standard aims to encompass all of these and more within a universal Web services security framework. Representative approaches from recent research in the area of SOA security techniques will be detailed below.

Specification	Function	Operational Impact
XML Encryption	Encrypts XML elements	Granular data confidentiality
XML Signature	Signs all or part of XML document	Full or partial data integrity
eXtensible Access Control Markup Language (XACML)	Expresses access control policies in XML format	Enables access control within Web services
Security Assertion Markup Language	Associates users with identity-based attributes	Allows for federated trust models
WS-Security	Provides framework for more detailed WS-* specifications	Integrates security operations with SOAP

Table 2. Summary of selected XML-based security specifications

Skalka and Wang (Skalka & Wang, 2004) analyze the problem of access control for Web services. Under an SOA model, in which transactions could be processed via multipoint distributed services, how can a given transaction be authorized? This work proposes a "trust-but-verify" approach, in which authorization is separated into different online and offline phases. To expedite transactions, in the online phase, trust is granted relatively freely. In the offline phase, however, auditing is performed and rogue transactions can be reversed. This analysis points out that such a system is consistent with current practices in areas such as credit card verification. It can also be noted that the trust-but-verify model provides a perfect expression of the Clark-Wilson separation of duties approach to provisioning security. In (Skalka & Wang, 2004), front end transaction processing is performed by one set of systems, while auditing and verification are provided by another. Although this research does not address implementation mechanisms for creating the required audit trail, it can be noted that the logging of message traffic is already typically performed by network perimeter devices such as firewalls. If such edge devices included XML filtering capabilities, then they could play a role in the auditing process required for the trust-but-verify approach to Web services security.

Security Approaches	Operational Impact
Trust-but-verify	Transaction auditing
Attribute Based Access Control	Authorization
Information Flow Control	Confidentiality, integrity

Table 3. Operational impact of different security approaches

Yuan and Tong (Yuan & Tong, 2005) propose an approach called Attribute Based Access Control (ABAC) for Web services. This approach provides a finer grained security model than that of earlier access control systems such as identity based access control (IBAC), role based access control (RBAC), or lattice based access control (LBAC). Rather than deriving access rights from a single characteristic (user, role, or security classification) as in the earlier systems, ABAC allows for access policies to derive from any logical combination of subject, object, or environmental attributes. Under this model, Attribute Authorities (AA) create and manage attributes for subjects, resources, and the service environment. Logical calculus is performed on these attributes, resulting in access control policies. The resulting policies are deployed at Policy Enforcement Points (PEP) and Policy Decision Points (PDP). A logical diagram is provided, showing the relationship between various security services such as identity store, service registry, policy decision services, policy administration service, and policy store. Although not strictly required by this logical model, it seems very clear that any or all of these services could be separated out as duties to be performed by network perimeter devices. Also, given that the attribute-driven policies in this model are expressed with XML-based formats such as SAML and

XACML, systems to able to support this model would require XML processing capabilities. In short, it appears that deployment of XML-aware network edge systems would facilitate implementation of an ABAC model for SOA security.

Tari, et al. (Tari et al., 2006) go beyond access control to address the problem of information flow control for Web services. Whereas access control manages rights to resources on a one time basis, information flow control considers the entire data life cycle. Access control by itself is insufficient to provide security functions such as confidentiality and integrity, for the simple reason that once users gain access to information, there are no further checks on how they might distribute or modify the data. Information flow control invokes measures to secure data beyond initial access. This flow control for Web services is accomplished through labels attached to XML objects. Such labels pass between Owners and Readers of objects, specifying access rights and verifying message integrity. It can be noted that such a model is fully supported under extant standards such as XML Signature and XML Encryption. Each of these established XML security standards aims at granular application to specific XML nodes, elements, child elements, documents, or attachments. In contrast to the lower level Virtual Private Network (VPN) technology, the XML security standards do not require that entire messages by encrypted or digitally signed. Rather, XML security allows for portions of a message to flow in clear text, while other portions are obfuscated.

Useful as XML security standards may ultimately be, XML itself may prove elusive for network practitioners. Networkers are accustomed to managing devices, not data formats. How can XML security be implemented at a network's edge? Within the most common current network designs, functions such as encryption (VPN provisioning) and Authentication, Authorization, and Accounting (AAA) are often performed by network perimeter devices such as firewalls. Under the packet filtering model expressed by most firewalls, however, such security operations apply to entire messages, not to message parts or elements. To implement a more granular model such as XML label checking (Tari et al., 2006), XML capabilities need to be added to the network. Earlier Web services security research (Cremonini et al., 2003; Damiani et al., 2002) also noted the need to provide a network infrastructure for Web services security functionality. These studies suggested some typical features of Web services security architecture upon which later research has continued to build. Such features include a separation of duties between Policy Decision Points and Policy Enforcement Points. They also include one or more XML specifications for expressing security dimensions such as access control, flow control, or security policy. Thus, research into Web services security techniques leads naturally to the further consideration of service architectures and service orchestration processes.

Software Engineering Models for SOA Security

In considering SOA network security, it is natural to consider the original source of the challenges, the SOA application itself. Many studies cast attention on secure software design practices for SOA, focusing on architectural or engineering methodologies as the means to create secure services. Selected examples of such approaches are illustrated below. The need for coherent architectures and effective engineering practices is difficult to dispute. A question that remains open, however, is to what extent such software-centric security frameworks address the particular needs of application support over a network.

Epstein, et al. (Epstein et al., 2006), for example, suggest improved software engineering practices as the best means for securing SOAs. Among these suggestions are security-focused Quality Assurance (QA), penetration testing, automated vulnerability testing, source code analysis, defect density prediction, and development methodologies that identify security problems before they occur. This study, an overview, does not detail in an operational way what the suggested development methodologies are, however. Likewise, the study exemplifies an approach in which all weight is placed on application engineering and relatively little consideration is provided for the application's networked deployment. Such approaches do not separate duties between developers and network managers, leaving the network security teams short on guidance. Important as good QA and software engineering practices are, more comprehensive models are needed to support distributed and networked service deployments.

Software Engineering Model	Impact on Network Administration
Quality Assurance	Developers must build security into the application.
Model Driven Architecture	Network elements included in abstract design model.
Design Patterns	Specific network functions modeled in software design.

Table 4. Software engineering approaches to Web services security

Nakamura, et al. (Nakamura et al., 2005) aim at such a comprehensive modeling process through an attempt to synthesize SOA with Model Driven Architecture (MDA). This work defines four levels for SOA security discussion: strategy level, operation level, execution level, and deployment level. The authors propose a modeling approach called transformation in which abstract security requirements defined at the strategy or operations levels are mapped to more detailed executions and deployments. Unified Modeling language (UML) is used to express both the high level requirements and the implementation details. Although a separation of duties is beginning to take shape in this work (with network infrastructure being noted as one of the deployment level elements), the network dimension of the model remains a bit sketchy. While here the network is coming into view, it has yet to emerge as a fully-fledged, essential component of the total security architecture.

Fernandez and Delessy (Fernandez & Delessy, 2006) extend further towards a networking perspective, continuing the emphasis on UML modeling for SOA network security, but discussing more explicitly the role of network elements in the resulting models. This work identifies a design pattern called "Application Firewall", which is embedded in a larger system of relationships. The essential service flow is from Client through Application Firewall to Application. Application Firewall, in turn, is composed of subclasses such as PolicyAuthorization Point, PolicyEnforcement Point, and ContentInspector. In this way, UML and other model-driven techniques are brought to bear on the specific question of the network's security role. The framework proposed here specifies a similar service flow from Client to Application through Application Firewall, also extending the notion of PolicyEnforcement Point as one of its key features. Fernandez and Delessy thus provide a useful abstract framework for SOA network security, leaving room however for much additional work in the area of implementation.

Steel, et al. (Steel et al., 2006) provide perhaps the most comprehensive architectural framework for SOA security, discussing Web services security design patterns in the larger context of enterprise n-tier application security design. At the most abstract level, this work separates the duties of perimeter security from the services themselves and from security principles such as integrity, confidentiality, authorization, and authentication. In this model, (called "the Security Wheel"), the perimeter is at the edge of the wheel, with services at the axle and the various security principles as spokes. The implication is that a principle such as integrity is not provided by either the service or the perimeter acting in isolation from one another, but rather through an orchestration of duties between the service and the perimeter. This makes explicit the need to model network functions within the overall services context. Additional detail is provided toward such network infrastructure modeling through the discussion of design patterns such as Message Interceptor Gateway and Intercepting Web Agent. These patterns resemble Application Firewall (Fernandez & Delessy, 2006), but provide more component-level models with class patterns such as MessageInspector, IdentityProvider, and MessageHandlerChain. In this work, the duties of network perimeter systems for Web services processing are thus becoming more detailed and more clear. Where clarity fails in this model, however, is in its invocation of the Open Systems Interconnection (OSI) seven-layered network model to frame the protocol-centric interventions it proposes.

OSI Model Layers	**Protocols and Standards**	**Classification (Steel, et. al, 2006)**
Application	XML/SOAP/WSDL WS-Security XML Digital Signature XML Encryption SAML REL	Message Layer Security
	HTTP over SSL/TLS	Transport Layer Security
Presentation		
Session		
Transport	TCP	Network Layer Security
Network	IP	
Data Link		
Physical		

Figure 3. Message layer security. Adapted from (Steel, et. al, 2006)

As is commonly the case in SOA network security discussions, the entire XML infrastructure for Web services is encompassed in (Steel, et al, 2006) by OSI layer seven (application layer). This leads to an overloaded and confusing security specification at this level, because of the heterogeneous nature of the standards and protocols thus packed into the application layer. XML security, for example, is characterized as "message-layer" security, occupying the upper portion of the application layer. Included in message-layer security are XML/SOAP (Simple Object Access Protocol)/WSDL (Web Services Description Language), WS-Security, XML Digital Signature, XML Encryption, SAML, and Rights Expression Language (REL). This list intersperses security frameworks, languages, and technologies along with messaging and service description formats that in themselves may not necessarily contain security features. Such a miscellaneous list lacks the operational specificity needed for network perimeter security deployments. Although Steel et al. (Steel et al., 2006) represent an important step toward describing secure SOA designs or architectures, network managers charged with application layer security support still lack the kind of targeted framework that facilitates implementation. Sorting through competing XML standards would at best be a distraction for network operations. What is needed is a model that assimilates XML into action items for perimeter security. Such models are indeed beginning to emerge both in theory and in practice.

Implementing SOA Security

Research and development of XML security systems has grown in recent years along with the interest in Web services and service architectures. Nomenclature for this category is unsettled, however, with the terms firewall, proxy, and gateway being perhaps the most common for describing the type of product or system used to secure XML-based network application traffic. It is worth noting, however, that the potential product category of XML Intrusion Detection Systems (XML IDS) does not appear to have received much research attention as of yet. Recent searches on this expression yielded little. By contrast, the literature on XML Intrusion Prevention is relatively abundant. XML Intrusion Prevention turns out to be a generic description of what XML firewalls, proxies, and gateways are designed to do, namely blocking deleterious XML traffic at a local network's edge. Below then is a summary of some of the main implementation options recent research offers for XML Intrusion Prevention, reserving for later the question of XML Intrusion Detection.

Within the category of intrusion prevention, Firewalls are often considered the first line of defense for network security. Bebawy, et al. (Bebawy et al., 2005) describe an open source XML firewall, called Nedgty. This firewall is designed to ward off some of the simplest XML-based threats. Denial of Service, Buffer Overflow, and XML Denial of Service are three classes of attacks prevented by this system. Nedgty works by queuing incoming packets and then sending the packets to a validation subsystem. If the packets are allowed, then they are reconstructed with the firewall as the source address. In effect, the Nedgty firewall works as an XML proxy. Nedgty is built for Linux and communicates with the IPTables TCP/IP firewall feature in the Linux OS. The designers of Nedgty intend future enhancements such as enforcing XML security standards, including XML Signature, SAML, and XACML.

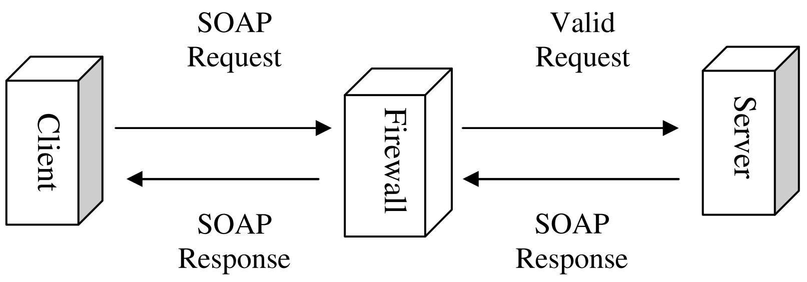


Figure 4. Scenario for an XML firewall (Bebawy et al., 2005)

Koyama, et al. (Koyama et al., 2005) detail a different approach to intrusion prevention called WS-Proxy. WS-Proxy is a design architecture for the control of Web services communications. Because XML is extensible, the authors argue that WS-Proxy must also be extensible. This is accomplished through plug-ins. WS-Proxy evaluates XML messages according to rules. It then selects which plug-ins are needed to process each message before forwarding it on. Problems for XML proxies include the need to efficiently parse and process XML. Common XML parsers impose much processing overhead. This may not be required, however, based on the depth of XML inspection demanded by various types of traffic. WS-Proxy defines a special plug-in interface to improve efficiency by invoking differentiated parsing routines. Policy declarations for WS-Proxy may be specified via XPath, the XML query language. For this research, the authors implemented a model of WS-Proxy in Java. Other elements of the prototype platform included Linux, Apache, and Axis.

In addition to research projects such as Nedgty and WS-Proxy, a growing number of commercial products are becoming available to process XML security specifications at the network perimeter. Some of these vendors include Forum Systems, DataPower, Reactivity, Layer 7, and F5. Heasley (Heasley, 2004) compares features on some these products. Heasley divides this product category into two types: hardware-based and software-based. Hardware-based XML firewalls are generally faster, but more expensive. An early offering by Forum Systems combined hardware and software features to selectively encrypt and decrypt sensitive XML fields. XML validation is also provided. A more recent offering by the same company supports many additional protocols and adds intrusion detection features. Another company, DataPower, creates a virtual firewall for every service exposed to the outside world. Each virtual firewall is configured with custom policy actions implemented through XSL stylesheets. A third company, Reactivity, features the ability to permit different firewall settings for different user groups for the same service. This would be useful for implementing any security designs based on user attributes.

In general, products such as XML firewalls, proxies, gateways, and appliances are becoming more prevalent. Lagging, however, are training and design frameworks showing network administrators why such products are needed and how they should be configured to improve security. Unlike the systematic prescriptions available for network or transport layer filtering, suggestions for application layer processing of XML reveal an ever-widening horizon of tools and protocols, with little true direction on how to begin engineering network security for SOA.

APPROACH

In this section, a proposal for SOA network security will be defined and detailed. This proposal aims to tighten the focus for network security designers, compressing the many XML security options into a manageable set of choices.

Definitions

What follows is an operational framework for designing and provisioning XML network security. The noun "framework" is intended to carry its common dictionary definition of a conceptual structure meant to simplify complex problems. The adjective "operational" is used to suggest that the desired results of the framework are action-oriented. The framework proposed below, then, aims at simplifying complex problems regarding network operations in the area of XML security.

The proposed framework begins with an acronym: FIX. FIX stands for Filtering to Inspect XML. The acronym is designed to encompass a wide range of security dimensions within a tight semantic space. The FIX model proposes a related nomenclature to facilitate analysis, modeling, planning, discussion, and deployment of XML-based security systems. This nomenclature includes:

FIX	–	Filtering to Inspect XML
FIX point	–	system or device whose function includes FIX
FIX list	–	a set of options for the configuration of a FIX point.

The acronym "FIX" may be used as a verb, including "to FIX", meaning to "filter and inspect XML". The term FIX point may be used in the abstract as a location in a network topology where there is a need to FIX. Concretely, hardware appliances, modules, or software subsystems may be deployed to instantiate FIX points in such locations. The term FIX list can be used to refer to 1) the superset of all potential actions to secure XML data traffic, 2) a specific subset of such actions technically available at a given FIX point, or 3) requirements for configurations or settings of a particular FIX point.

The expansive and intertwining connotations of the acronym "FIX" are all intentional. This protean acronym is recommended as a point of synthesis between design and deployment, theory and practice. Apart from its own native connotations, FIX can also rightly embrace many associated meanings of the common verb "to fix" The following definition from the *Free Online Dictionary* (Farlex, Inc., 2007) provides examples of such meanings. Among these are:

- to place securely; make stable or firm
- to secure to another; attach
- to put into a stable or unalterable form
- to direct steadily
- to capture or hold
- to set or place definitely; establish
- to determine with accuracy; ascertain
- to agree on; arrange
- to assign; attribute
- to correct or set right; adjust
- to restore to proper condition or working order; repair
- to make ready; prepare

All of these meanings of the common verb "to fix" apply literally and directly to one or more activities of XML processing required for network security. Such commonly recommended XML security techniques include:

- authentication, authorization, and accounting
- validation
- verification
- screening and filtering
- logging
- auditing
- encryption and decryption
- caching
- transforming

Each of these XML security activities is suggested by one or more meanings of the verb "to fix". To FIX, is thus, in a very real sense, to fix. The suggestive word play of the acronym may furnish a useful instructional device to help grasp the many ramifications of XML security. For this reason, the acronym FIX provides a handy, compact symbol for the underlying complexity of XML security design.

The Need for Simplification.

One might wonder, how does the list of verbs and activities above, headed by the acronym FIX, simplify problems of design and deployment for XML network security? Consideration of other currently available security design frameworks at similar levels of abstraction may shed some light on the matter. One alternative, for example, would be to associate network security interventions with one or more layers in the TCP/IP Network Reference Model. The chart below illustrates this strategy as currently applied for devices such as firewalls, IDS, and web proxy servers.

Network Layer	Protocols	Security Technique
Application	HTTP, HTTPS	Content Filtering, Encryption
Transport	TCP, UDP	Port Filtering
Internetwork	IP, ICMP	Packet Filtering
Data Link	PPTP, L2TP	VPN Tunneling

Table 5. Network layers, corresponding protocols, and security mechanisms

With respect to the well-known set of TCP/IP protocols, charts such as this are both common and useful. Entries in such a chart quickly summarize a range of security activities and are readily meaningful to networking practitioners. For example, blocking port addresses at the Transport Layer is a well-known security technique, which can be implemented by firewalls, routers, or even PC hosts. The implementation of such port address blocking is generally straightforward, as in this command intended for a Cisco router:

```
access-list 110 deny tcp any any eq 23
```

This command would close off network traffic in both directions on TCP port 23, effectively cutting off the Telnet application. The main point here is that communicating an idea such as shutting down Telnet on port 23 is neither complex nor time consuming. Simple command line tools, and in many cases, graphical user interface tools, are widely available to perform such operations. For the suite of TCP/IP protocols up through HTTP, a chart such as that shown above generally suffices to present the available options in a succinct and useful format.

Could we not then extend the same model a bit higher into the arena of Web services security? This would seem to be an obvious choice. To do this, however, we need to begin by placing Web services into the chart at one layer or another. Here, however, is where complications begin to arise.

Network Layer	Protocols	Security Technique
	SOAP, WSDL, UDDI, WS-Security, SAML	
Application	HTTP, HTTPS	Content Filtering, Encryption
Transport	TCP, UDP	Port Filtering
Internetwork	IP, ICMP	Packet Filtering
Data Link	PPTP, L2TP	VPN Tunneling

Table 6. Problem of locating Web services in a network reference model

As shown above, Web services are generally modeled as resting on top of TCP/IP application protocols such as HTTP. For Web services, HTTP (or alternatives such as FTP or SMTP) are regarded as transport bindings, not as fully fledged application protocols in there own right. This leads to some awkwardness of classification when the need arises to place Web services security into a networking context. Should Web services be located in the topmost TCP/IP layer, the application layer? Or do Web services require a new layer of their own beyond the application layer?

```
<soap:binding style="document" transport="http://example.com/smtp"/>
```

Figure 5. Example of SOAP binding element (W3C, 2001)

This problem is complicated further by the ever-growing complexity of protocols and specifications within Web services. Block diagrams illustrating the relationships between different Web services components generally run four or five layers deep. An attempt to dovetail the TCP/IP network stack with an equally complex Web services stack might then result in such an inelegant juxtaposition as in the chart below.

Web services create a stack within the TCP/IP Application Layer

Tool	Web Services Layer
WSFL	Service Flow
Static -> UDDI	Service Discovery
Direct -> UDDI	Service Publication
WSDL	Service Description
SOAP	XML-Based Messaging
HTTP, FTP, SMTP, MQ, RMI over IIOP	Transport

TCP/IP Layer	Example Protocols
Application	HTTP, FTP, SMTP
Transport	TCP, UDP
Network	IP, ICMP
Data Link	PPTP, L2TP

The TCP/IP Application Layer maps to Web services Transport.

Figure 6. Web services stack juxtaposed with the OSI model. (Myerson, 2002)

To resolve the problem of proper layer assignment for Web services, an authoritative reference on the concept of "layer" would be most helpful. Zimmerman (Zimmerman, 1980) presented one such classic statement very early in the specifications process for distributed data communications protocols. This work, which outlined what would be become known as the OSI reference model for networking, explained the rationale for which protocols should be placed together in layers and how the boundaries between adjacent layers should be defined. For example (quoting Zimmerman, with some rearrangement):

- Collect similar functions into the same layer.
- Create separate layers to handle functions that are manifestly different in the process performed or in the technology involved.
- Create a boundary where it may become useful at some point in time to have the corresponding interface standardized.

A cursory inspection of HTTP as compared to SOAP (or any other Web services specification for that matter), quickly reveals that HTTP has different syntax, functions, and processes than the XML-based Web services standards. HTTP can handle request-response sessions between clients and servers, transporting HTML and related content in the process. SOAP, by itself, is not capable of doing this. SOAP requires a transport binding (such as HTTP) in order carry its own type of message format. The processing of SOAP and other higher level Web services protocols requires specialized XML processes such as parsing, validation to schemas, canonicalization (white space normalization), and structural transformation. All of this is alien to the world of HTTP or other TCP/IP protocols. It would appear, on the surface, that SOAP and HTTP belong in different layers. Moreover, the interface between these layers, namely the SOAP transport binding, has already been specified. This would seem to confirm even more that Web services must need their own new network layer.

Tempering any such possible enthusiasm for the prospect of increasing the number of network layers in the standard reference models, however, we should also consider some of Zimmerman's other observations about when and when not to define a new layer:

- Do not create so many layers as to make difficult the system engineering task describing and integrating these layers.

- Create further subgrouping and organization of functions to form sublayers within a layer in cases where distinct communication services need it.

So if Zimmerman's guidelines support a case for defining a new network layer to handle Web services, they equally argue against such a case on the grounds that seven layers (in the OSI model) or four or five layers (in different versions of the TCP/IP model) are quite enough. Moreover, Zimmerman does suggest a reasonable alternative. Given the opportunity to define sublayers when technical differences require it, we would seem well justifed in defining, within the Application layer, an XML sublayer for purposes of orchestrating Web services network security.

The term XML sublayer preserves the economy of earlier reference models for data communications, while respecting one of the fundamental insights that lead to layered models in the first place. Protocols that differ substantially in form, syntax, semantics, or processing requirements from those other protocols with which they interact need to reside in their own grouping. So returning to the question of creating an operational framework for Web services network security, let us assume the availability of an XML sublayer and adjust our diagrams accordingly.

Network Layer	Protocols	Security Technique
Application Layer: XML Sublayer	SOAP, WSDL, UDDI, WS-Security, SAML	
Application Layer: TCP/IP Sublayer	HTTP, HTTPS	Content Filtering, Encryption
Transport	TCP, UDP	Port Filtering
Internetwork	IP, ICMP	Packet Filtering
Data Link	PPTP, L2TP	VPN Tunneling

Table 7. The XML sublayer

With the thorny issue of layer classification perhaps on its way to resolution, a compact and workable framework is now beginning to emerge. There remains only the question of finding an expression as handy and meaningful as "packet filtering", "port filtering", or "content filtering" in order to complete the picture. Here, however, is where the special nature of XML as compared to TCP/IP protocols beneath it in the network stack becomes apparent.

XML's structure, in comparison with lower layer TCP/IP protocols is quite unique. TCP/IP protocols are generally fixed in length. Or, in cases where these protocols can be extended, the type and number of available extensions are well defined in publicly available repositories such as the Internet Engineering Task Force's Request for Comment (RFC) system. For this reason, the processing of TCP/IP network protocols can often be reduced to reading and responding to predictably located sequences of bits. Not so with XML. There is no RFC to specify the length of a standard XML document. The very name of XML incorporates the idea of extensibility. XML documents can be as small as a handful of tags or as large as any file one might choose to imagine. Beyond this, end users are very welcome to invent their own XML expressions and vocabularies. Such open-endedness is thoroughly embedded in the design philosophy and formal specifications for XML. No other protocol in the TCP/IP stack works in quite so open a manner.

The problems of securing XML network traffic then are as ramified as the very structure of XML itself. A whole host of processing questions must be answered before any given XML payload can be considered secure. Has proper XML syntax been used (is the XML well-formed)? Does the XML match an expected schema or Document Type Definition (DTD), i.e. is it valid? Have any of the individual XML elements or sub-trees been encrypted or digitally signed? In this case, the XML needs to be parsed, and decryption or signature algorithms need to be applied specifically only to those elements of the XML sent in such an encoded manner. These processes are complicated by the question of whitespace. XML is oriented toward human readability. So tags and elements that differ only in the amount of whitespace they contain are considered logically equivalent. But encryption or digest algorithms require exactly matching binary inputs to produce correct results. So this imposes the additional XML processing requirement of canonicalization (white space normalization).

Beyond such structural issues for XML security inspection, there also remains the fact that XML syntax can be used to encapsulate content that may be critical from a security point of view. Security assertions themselves, for example, can be embedded in the XML payload. Such elements may include usernames, passwords, Kerberos tokens, and even references to X.509 digital certificates. Likewise, content that would not so obviously be associated with security functions might nevertheless have important security implications. Well-known malicious payloads such as buffer overflow attacks, SQL injections, or cross-site scripting attacks can readily by encapsulated in XML wrappers and transported across network boundaries (Shah, 2007). Firewalls and other devices that stop their packet inspections below the XML sublayer cannot detect such content. But devices and personnel that try to anticipate both XML structural and XML content-driven attacks would seem to have an endless (and thankless) list of duties to perform. Given the vast number of potential XML formats and payloads, how can a simple binary decision between "permit" and "deny" ever sensibility be resolved? What attributes of the XML would be decisive in determining such choices? How could such attributes and their allowable settings be communicated and configured?

A key idea behind the FIX model is to limit, reduce, cut down to size, or – in a word – fix the number of choices needed for XML security design. There may indeed be formal mathematical implications behind the FIX model in the sense that we are mapping between an infinite set of options (XML elements and vocabularies) to a limited (fixed) set of operations. This suggests allied problems in complexity theory related to the tractability of large scale optimizations. Leaving such formal questions for future consideration, on a heuristic level at least, the FIX model proposes the following general process for specifying a networked implementation of XML security. The basic idea is to make a list. Lists can be used to detail routines and procedures. In more object-oriented or event-driven scenarios, lists can at least be used to organize menus of parameters or configuration options. In either case, to form a list is to limit (fix) the configuration space of the possible settings, filters, and processing actions needed for XML packet inspection.

The example below, for example, suggests a sequential, procedural routine for checking into XML structural anomalies.

1) Preprocessing
2) Check for Well-formedness
3) Check for invalid characters
4) Parse document
5) Validate document against a schema or DTD

The rationale for this sequencing is as follows: Preprocessing needs to occur before XML processing does. A key test at the preprocessing stage is to verify that file size is reasonable. This is important, because overly large files can be used to create XML Denial of Service attacks against XML parsers. So some extrinsic test is needed to screen out bad files before parsers start processing them (because if the parser discovers the file is too large, the denial of service attack has already succeeded – the parser has been overloaded during the process of file size discovery) .Well-formedness is the next thing to test. The reason for this is, if the XML is not well-formed, other processes are not likely to be meaningful. Also, the standards for well-formed XML are generally available, so no special configuration is needed at this stage. As another early step, invalid character screening is also recommended. A variety of regular expressions can be tested in a linear pass through the file or data stream to find any suspicious character combinations that may be associated with injection attacks or other attack signatures. Only in the absence of concerns in these first few areas does it then make sense to parse the XML. In the process of parsing, validity of the XML against a schema or DTD may also be tested. (Validation before parsing is not an option, however, because XML parsing is required as a precursor for validation.) Whether on not the particular sequence above is universally accepted, the list of action steps so presented does at least illustrate that some elements of XML packet inspection may rely on a fairly standard set of procedural filters.

Beyond these initial steps, however, the furcated nature of XML takes over and linear sequences (such as simple process-oriented scripts or subroutines) fail to encompass the problem space. XML root elements can spawn many children and branches, and any of these sub-elements might contain security-significant payloads. This is where the many XML- and WS-Security specifications begin to play a role. All of these specifications describe specialized XML tag syntax that gives semantic meaning to the payload within the tag. So, for example, a line like that below means the payload is cipher text.

```
<xenc:CipherValue>a8G4FF6a…5nld3G5d</xenc:CipherValue>
```

The same sort of character string in different tags could be part of a digital signature, a password challenge, a piece of binary code, or just nonsense (as might be the case for an attempted denial of service or buffer overflow attack). The problem of testing the security properties of content such as this involves applying any of a wide range of specifications to an even wider set of potential data elements. To summarize the available options with respect to XML content inspection, we might say that depending on what sort of payload is anticipated by the application, one or more XML filters needs to be applied. A filter, in brief, is a specifiable set of rules or mechanisms. The phrase Filter to Inspect XML is thus shorthand for the larger process of selecting filters for use in the inspection of XML data traffic. Whether the system goal is intrusion prevention (firewall, gateway, or proxy) or intrusion detection (IDS), the concept of XML filter selection still pertains. For this reason, a FIX subsystem is needed on any sort of network security device designed to attend to XML.

The vast range and complexity of potential XML security actions may itself hinder XML security. Bishop (Bishop, 2003) notes that simplicity is a core design principle of information security: “Simplicity makes designs and mechanisms easy to understand. More importantly, less can go wrong with simple designs. … Simplicity also reduces the potential for inconsistencies within a policy or set of policies.” To the extent that simple, unambiguous procedures for XML security inspection are possible, the FIX model can embody such procedures. More significantly, in so far as XML by its nature transcends procedural limitations, the FIX model may prove useful for suggesting the development of a set of optional filters to be applied on a case by case basis. This combining of standard procedures with a specifiable set of options reduces to the problem of XML network security to something roughly comparable to security designs and activities at other network levels. The resulting operational framework would then be this:

Network Layer	Protocols	Security Technique
Application Layer: XML Sublayer	SOAP, WSDL, UDDI, WS-Security, SAML	Filtering to Inspect XML (FIX)
Application Layer: TCP/IP Sublayer	HTTP, HTTPS	Content Filtering, Encryption
Transport	TCP, UDP	Port Filtering
Internetwork	IP, ICMP	Packet Filtering
Data Link	PPTP, L2TP	VPN Tunneling

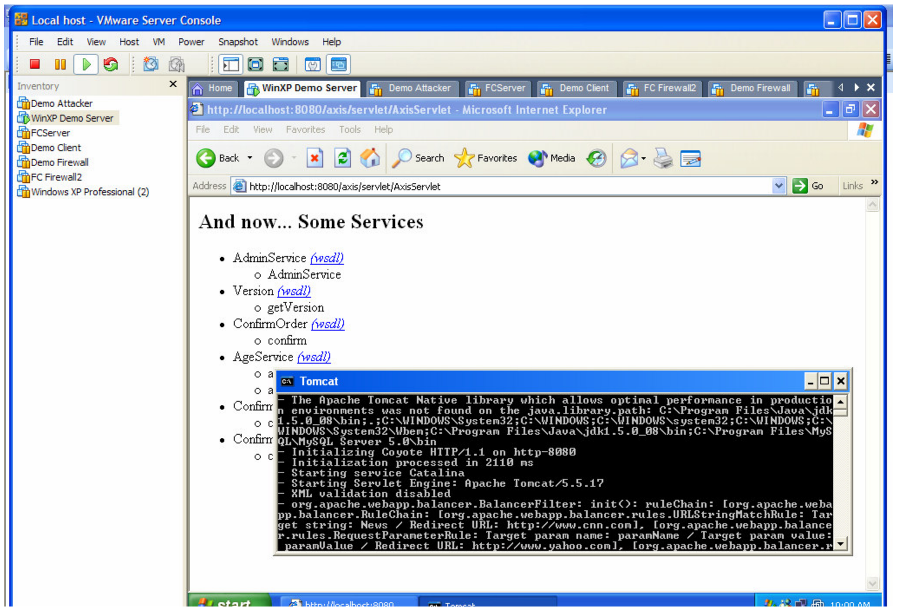
Table 8. An operational framework for XML network security

TESTING AND RESULTS

The FIX model assumes that different filters will be needed in different scenarios for the security inspection of XML-based applications. Further, a key goal of the model is to improve the operational simplicity of deploying such filters. To illustrate the practicality of the proposed approach, a demonstration system has been developed featuring some of the key ideas of the FIX model. First, an assortment of XML inspection filters was developed. Then, these filters were added to a simple TCP/IP proxy server. Testing was then conducted to determine the XML-filtering capabilities of the proxy server where various filters or combinations of filters were selected.

Test Design

The FIX model is abstract and can be implemented on a variety of platforms and languages. The specific tools selected for the prototype in this study included a mixture of commercial and open source solutions. First, to speed development and testing, a virtual network environment was created on VMWare Server. Several virtual OS images were installed on this platform, including Windows XP Professional and Fedora Core 4 Linux. (See Figure 7.)


Local host - VMware Server Console
File Edit View Host VM Power Snapshot Windows Help
Inventory
Demo Attacker
WinXP Demo Server
FCServer
Demo Client
Demo Firewall
FC Firewall2
Windows XP Professional (2)
Home
WinXP Demo Server
Demo Attacker
FCServer
Demo Client
FC Firewall2
Demo Firewall
http://localhost:8080/axis/servlet/AxisServlet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/axis/servlet/AxisServlet Go Links
Tomcat
start

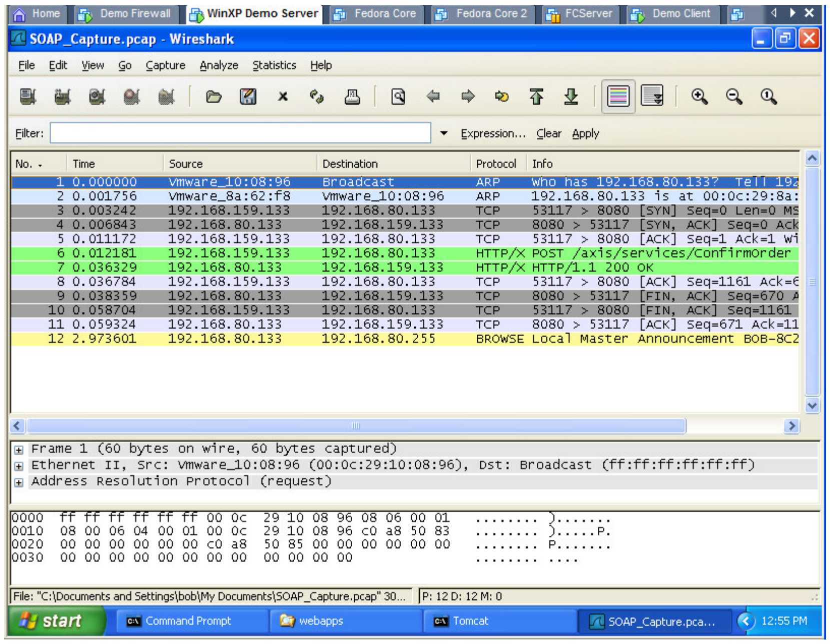
And now... Some Services

- AdminService (wsdl)
  - AdminService
- Version (wsdl)
  - getVersion
- ConfirmOrder (wsdl)
  - confirm
- AgeService (wsdl)
  - a
  - a
- Confirm
  - c
- Confirm
  - c

```
- The Apache Tomcat Native library which allows optimal performance in productio
n environments was not found on the java.library.path: C:\Program Files\Java\jdk
1.5.0_08\bin;.;C:\WINDOWS\System32;C:\WINDOWS\system32;C:\WINDOWS;C:\
WINDOWS\System32\Wbem;C:\Program Files\Java\jdk1.5.0_08\bin;C:\Program Files\MyS
QL\MySQL Server 5.0\bin
- Initializing Coyote HTTP/1.1 on http-8080
- Initialization processed in 2110 ms
- Starting service Catalina
- Starting Servlet Engine: Apache Tomcat/5.5.17
- XML validation disabled
- org.apache.webapp.balancer.BalancerFilter: init(): ruleChain: [org.apache.weba
pp.balancer.RuleChain: [org.apache.webapp.balancer.rules.URLStringMatchRule: Tar
get string: News / Redirect URL: http://www.cnn.com], [org.apache.webapp.balance
r.rules.RequestParameterRule: Target param name: paramName / Target param value:
 paramValue / Redirect URL: http://www.yahoo.com], [org.apache.webapp.balancer.r
```

Figure 7. Test network on VMWare

Although running via virtualization, each of these images contains a fully featured and independent operating system. The OS images communicated with each other by TCP/IP networking and virtual switches. Packet capturing software (such as Wireshark) shows this network communication to be essentially identical to that seen between different physical machines, as shown in Figure 8.


Home
Demo Firewall
WinXP Demo Server
Fedora Core
Fedora Core 2
FCServer
Demo Client
SOAP_Capture.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No.  Time      Source            Destination       Protocol Info
  1 0.000000   Vmware_10:08:96   Broadcast         ARP      Who has 192.168.80.133?  Tell 192
  2 0.001756   Vmware_8a:62:f8   Vmware_10:08:96   ARP      192.168.80.133 is at 00:0c:29:8a:
  3 0.003242   192.168.159.133   192.168.80.133    TCP      53117 > 8080 [SYN] Seq=0 Len=0 MS
  4 0.006843   192.168.80.133    192.168.159.133   TCP      8080 > 53117 [SYN, ACK] Seq=0 Ack
  5 0.011172   192.168.159.133   192.168.80.133    TCP      53117 > 8080 [ACK] Seq=1 Ack=1 Wi
  6 0.012181   192.168.159.133   192.168.80.133    HTTP/X   POST /axis/services/ConfirmOrder
  7 0.036329   192.168.80.133    192.168.159.133   HTTP/X   HTTP/1.1 200 OK
  8 0.036784   192.168.159.133   192.168.80.133    TCP      53117 > 8080 [ACK] Seq=1161 Ack=6
  9 0.038359   192.168.80.133    192.168.159.133   TCP      8080 > 53117 [FIN, ACK] Seq=670 A
 10 0.058704   192.168.159.133   192.168.80.133    TCP      53117 > 8080 [FIN, ACK] Seq=1161
 11 0.059324   192.168.80.133    192.168.159.133   TCP      8080 > 53117 [ACK] Seq=671 Ack=11
 12 2.973601   192.168.80.133    192.168.80.255    BROWSE   Local Master Announcement BOB-8C2

⊞ Frame 1 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: Vmware_10:08:96 (00:0c:29:10:08:96), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 00 0c  29 10 08 96 08 06 00 01   ........ ).......
0010  08 00 06 04 00 01 00 0c  29 10 08 96 c0 a8 50 83   ........ ).....P.
0020  00 00 00 00 00 00 c0 a8  50 85 00 00 00 00 00 00   ........ P.......
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....

File: "C:\Documents and Settings\bob\My Documents\SOAP_Capture.pcap" 30...   P: 12 D: 12 M: 0
```

Figure 8. SOAP traffic captured between virtual OS images

To test XML-based Web services, an Apache Tomcat server was added to one of the images. Axis 1.4 was installed on Tomcat, and a variety of test services were deployed using the Java utilities incorporated in Axis. Java clients to consume the test services were also written and installed on a different client OS image. Having shown simple client/server functionality with the test services, the virtual network was then altered to include a firewall image between the client and the server. IP address schemes and the Linux IPTables utility were used to force all client/server traffic through the firewall image. This firewall image was then used to host the primary demonstration system, called FIX Point Proxy.

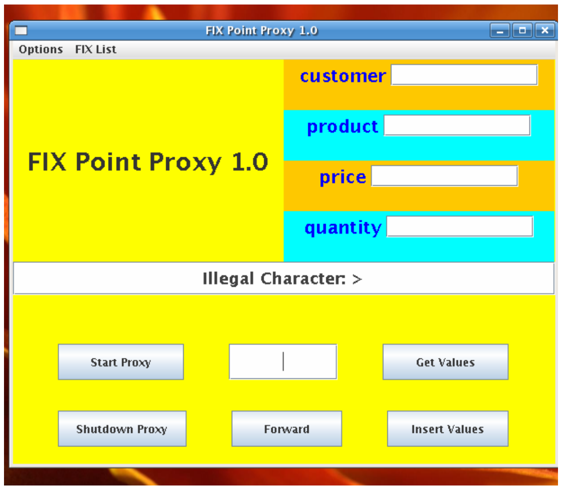


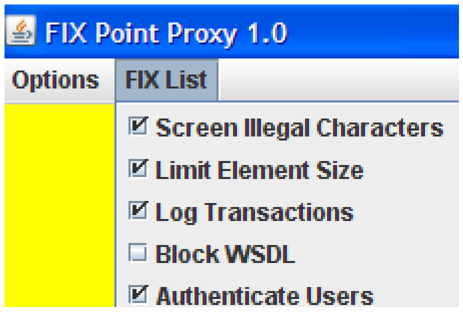
Figure 9. Screen shot for FIX Point Proxy

The most significant feature of FIX Point Proxy is its specific focus on filters as a design abstraction for the organization of XML security inspection processes. Filters are implemented as configurable options that can be invoked or disabled by network administrative personal not deeply schooled in the theory of XML protocols. Filters thus meet the functional requirement of a separation of duties model. On the one hand, a software or security architect could specify which filters should be applied. On the other hand, a network administrator could apply the filters and verify the results of this action.

Filter	Function	Rationale
Transparent Mode	Passes HTTP traffic. No XML filters enabled	Baseline testing of TCP/IP functionality below XML sublayer
Block WSDL	Blocks viewing of WSDL files	Prevents reconnaissance of Web services internals
Screen Illegal Characters	Regex to limit element content to alphanumeric characters only	Prevents XML Injection strategy of passing XML as text values
Limit Element Size	Limits maximum element size to a selected value	Useful against Denial of Service and buffer overflow
Authenticate User	Parsers XML for authentication string	Models authentication within the XML sublayer
Log Transactions	Records transactions to file	Models auditing system. Also suggests approach to XML IDS.

Table 9. Selected filters from prototype FIX system

To facilitate communications between designers and deployers, a list of filters for the FIX Point Proxy has been organized in the form of drop down menu. Such a menu illustrates one approach to the implementation of a FIX list. This graphical user interface approach was selected for this prototype, because a key specification for the project was to facilitate demonstration and training. High speed performance was not a functional requirement for this prototype. Indeed, performance was slowed down in places by the introduction of meaningless loops in order to improve the timing of visual effects in live demonstrations. Such visual elements are only superficially related to the underlying FIX model. In a production environment, high speed performance and light-weight controls would be more important requirements, and extraneous presentation features could be discarded.



☑ Screen Illegal Characters
☑ Limit Element Size
☑ Log Transactions
☐ Block WSDL
☑ Authenticate Users

Figure 10. Implementation of a FIX List via drop-down menu

Test Results

For testing purposes, a variety of Web services were deployed. First, as a baseline, a transparent server option was added to the FIX Point Proxy. In transparent mode, all network traffic was passed with no interference from the proxy. Under transparent mode, the client OS was shown to have complete HTTP access to the Apache web server on the server OS. Such access included the WSDL files associated with the Web services deployed on Axis on that server. Client classes corresponding to each Web service were deployed on the client image. In transparent mode, each Web client application was used to call its corresponding Web service, using a variety of test parameters. Each Web service was designed so the server would need to perform some simple operation (such as multiplying two input values) and then return the results to the client. Testing in this manner showed that as a baseline the FIX Point Proxy could be configured to pass all traffic transparently and to allow any Web service through.

Use Case #1: A Test Web Service

Client requests confirmation of a purchase order. Client inputs a username and product ID as strings. Client inputs a price and a quantity as integers. Server accepts the inputs and multiplies price by quantity. Server returns a single string to the client, echoing back all the inputs along with a calculated total price.

WSDL for Use Case #1

```
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/services/ConfirmOrder">
      <!--
WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)
-->
      <wsdl:message name="confirmRequest">
            <wsdl:part name="buyerID" type="xsd:string"/>
            <wsdl:part name="productID" type="xsd:string"/>
            <wsdl:part name="price" type="xsd:int"/>
            <wsdl:part name="quantity" type="xsd:int"/>
      </wsdl:message>
      <wsdl:message name="confirmResponse">
            <wsdl:part name="confirmReturn" type="xsd:string"/>
      </wsdl:message>
      <wsdl:portType name="ConfirmOrder">
            <wsdl:operation name="confirm" parameterOrder="buyerID
productID price quantity">
                  <wsdl:input message="impl:confirmRequest"
name="confirmRequest"/>
                  <wsdl:output message="impl:confirmResponse"
name="confirmResponse"/>
            </wsdl:operation>
      </wsdl:portType>
      <wsdl:binding name="ConfirmOrderSoapBinding"
type="impl:ConfirmOrder">
            <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="confirm">
                  <wsdlsoap:operation soapAction=""/>
                  <wsdl:input name="confirmRequest">
                        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://demo" use="encoded"/>
                  </wsdl:input>
                  <wsdl:output name="confirmResponse">
                        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/ConfirmOrder"
use="encoded"/>
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="ConfirmOrderService">
            <wsdl:port binding="impl:ConfirmOrderSoapBinding"
name="ConfirmOrder">
                  <wsdlsoap:address
location="http://localhost:8080/axis/services/ConfirmOrder"/>
            </wsdl:port>
      </wsdl:service>
</wsdl:definitions>
```

Sample Input for Use Case #1:

```
-lhttp://firewall:3000/axis/services/ConfirmOrder
Joey  XYZ123  135 4
```

Sample Output for Use Case #1:

```
Customer# Joey
Product# XYZ123
4 units @ $135
Total is $540
```

To illustrate the vulnerability of Web services traffic to illicit “Man in the middle” activities, the open source Wireshark packet capturing tool was used to capture SOAP traffic between client and server. This capture was easily analyzed, showing all request and response parameters between client and server. Using this information, a Document Object Model (DOM) parser was configured on the FIX Point Proxy, using expected SOAP request and response parameters to capture and manipulate values within the SOAP envelope. In this manner, it was shown how freely available technologies could be used to intercept and manipulate Web services traffic in a manner not visible to the client or the server. Defenses against such traffic manipulation were also demonstrated, using message digests to verify the integrity of message traffic for the Web services.

Use Case #2: Man in the Middle

Client requests confirmation of a purchase order. Client inputs a username and product ID as strings. Client inputs a price and a quantity as integers. Proxy between client and server intercepts the message. Proxy parses SOAP request and captures element values. Proxy changes one or more element values and forwards modified inputs to the server. Server accepts the inputs and multiplies price by quantity. Server returns a single string to the client, echoing back all the inputs along with a calculated total price.

WSDL for Use Case #2

Same as for Use Case #1

Sample Input for Use Case #2:

```
-lhttp://firewall:3000/axis/services/ConfirmOrder
Joey  XYZ123  135 4
```

Sample Output for Use Case #2:

```
Customer# Joey
Product# XYZ123
40 units @ $135
Total is $5400
```

To illustrate the difference between XML parsers, a Simple API for XML (SAX) parser was also added to the proxy. To show how an XML Injection attack can succeed against a SAX parser, crafted XML packets were used to overwrite authentication values detected by the SAX parser.

```
Crafted parameters for XML Injection

<arg 0> Test data </arg 0>
<arg 1> <arg 0> Crafted data </arg 0> </arg 1>
```

Figure 11. Example of an XML injection attack

Using an event-driven process and lacking a model of the data in object memory, the SAX parser overwrites the value “Test data” with “Crafted data” for the string parameter <arg 0/>. Demonstrations such as this were used to show the need for data validation at any of the possible FIX Points in this topology. For example, when the FIX Point Proxy was configured to exclude non-alphanumeric values from string fields, the injection attack above was prevented. Similarly, other security interventions for Web services were modeled on the test network. Such interventions included blocking WSDL discovery, checking for maximum element size, and transaction logging. Far from exhausting the potential techniques for XML application security, these demonstrations at most scratch the surface. Yet, given the growing deployment of XML-based Web services and given the need to improve training in security in this area, the test design used here contributes as a light-weight, accessible, and clear demonstration of how XML services security can fit into larger network security designs.

Use Case #3: An Injection Exploit

Client requests confirmation of a purchase order. Client inputs a username and product ID as strings. Client inputs a price and a quantity as integers. Client also inputs crafted data (see Figure 11) into an optional comment field as a string. Proxy parses the comment field as XML and overwrites username, circumventing authorization.

WSDL for Use Case #3

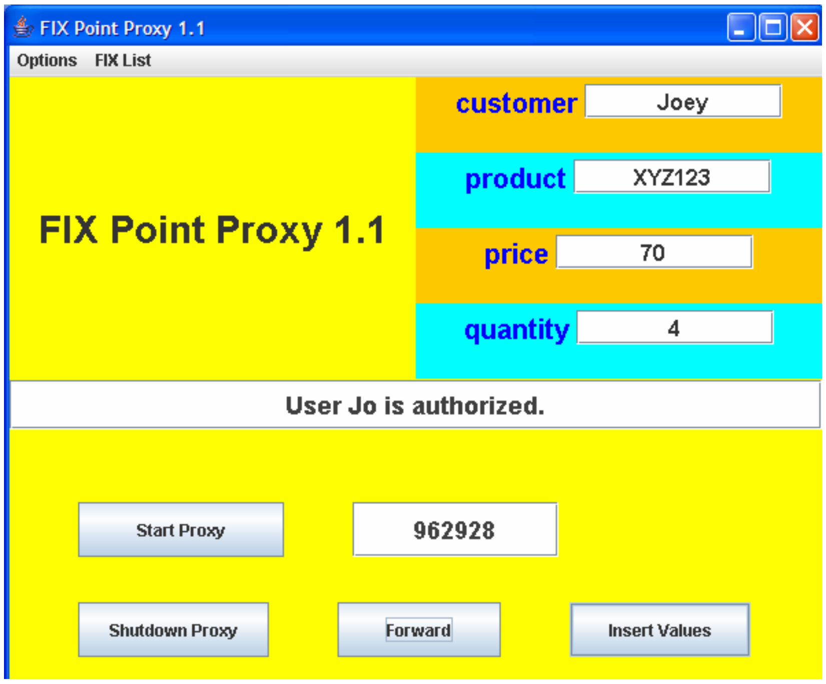
```
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/services/ConfirmOrder2">
      <!--
WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)
-->
      <wsdl:message name="confirmResponse">
            <wsdl:part name="confirmReturn" type="xsd:string"/>
      </wsdl:message>
      <wsdl:message name="confirmRequest">
            <wsdl:part name="buyerID" type="xsd:string"/>
            <wsdl:part name="productID" type="xsd:string"/>
            <wsdl:part name="price" type="xsd:int"/>
            <wsdl:part name="quantity" type="xsd:int"/>
            <wsdl:part name="comment" type="xsd:string"/>
      </wsdl:message>
      <wsdl:portType name="ConfirmOrder2">
            <wsdl:operation name="confirm" parameterOrder="buyerID
productID price quantity comment">
                  <wsdl:input message="impl:confirmRequest"
name="confirmRequest"/>
                  <wsdl:output message="impl:confirmResponse"
name="confirmResponse"/>
            </wsdl:operation>
      </wsdl:portType>
      <wsdl:binding name="ConfirmOrder2SoapBinding"
type="impl:ConfirmOrder2">
            <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
            <wsdl:operation name="confirm">
                  <wsdlsoap:operation soapAction=""/>
                  <wsdl:input name="confirmRequest">
                        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://demo" use="encoded"/>
                  </wsdl:input>
                  <wsdl:output name="confirmResponse">
                        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/ConfirmOrder2"
use="encoded"/>
```

```
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="ConfirmOrder2Service">
            <wsdl:port binding="impl:ConfirmOrder2SoapBinding"
name="ConfirmOrder2">
                  <wsdlsoap:address
location="http://localhost:8080/axis/services/ConfirmOrder2"/>
            </wsdl:port>
      </wsdl:service>
</wsdl:definitions>
```

Sample Input for Use Case #3:

```
-lhttp://firewall:3000/axis/services/ConfirmOrder2
Joey  XYZ123 70 4 <arg1>Jo</arg1>
```

Proxy Screen for Use Case #3:



customer Joey
product XYZ123
price 70
quantity 4

User Jo is authorized.

962928

Future Work

FIX Point Proxy prototypes an approach to XML security that, like XML itself, is readily extensible. The configuration options shown here could be expanded to include additional functionality both within and beneath the XML sublayer. Given that the FIX model standardizes the human interface to XML security through the abstraction of the filter, a roadmap to future development of this type of system includes the elaboration of the proposed filter set to encompass more and better XML security techniques. A number of other future research directions are also suggested by this work. One open question is the optimization of XML filter selection. In automated environments, is there an algorithmic solution to the problem of efficient filter invocation to screen out the most probable XML-based vulnerabilities? Another area to be explored is the integration of the FIX model with higher level service composition approaches such as Business Process Execution Language (BPEL) or semantic Web services. Also, looking beneath the XML sublayer to existing TCP/IP-based security systems, another question for future consideration would how to add XML intrusion detection to current IDS products such as Snort. This study suggests the FIX model, with its filters, might fit naturally with the rule-based approaches common to current TCP/IP firewalls, access control lists, content proxies, and intrusion detection systems. Finally, the FIX model provides a standardizable reference model and nomenclature for the side-by-side comparison of the many XML firewalls, proxies, gateways, and intrusion protection systems that have come onto the market in recent years. The FIX list model lends itself to feature list comparisons of product functionality for XML security, and the integration of the FIX model with the existing OSI and TCP/IP reference models should help network architectures get a "fix" on where XML security needs to be applied in their network topologies.

CONCLUSION

The field of SOA (including Web services and XML) network security is maturing, yet not mature. Specific procedures for securing XML network applications are not yet widely known. Although design patterns and representative approaches are beginning to emerge in this area, direct answers to questions of how to secure Web services implementations remain difficult to obtain. The FIX model proposed above provides several advantages over previous approaches to SOA security design. Firstly, it makes specific reference to network topology (through the notion of "FIX point"), which aids in the separation of duties between application development and application deployment. Secondly, through the concept of the "FIX list", the model points to the enumeration of security activities for SOA networks and makes the analysis of such activities more tractable. Finally, the compact acronym "FIX" suggests that XML application layers may require specific network security interventions appropriate to the distinctive nature of XML. As systems and products such XML firewalls, proxies, and gateways begin to enter more often into enterprise design, the FIX model offers a simplified point of reference for interdisciplinary discussion of such approaches. Clear and simple security designs including separation of duties require clear and simple conceptual frameworks to facilitate such separation. The FIX model is offered as a further step toward the understanding of the special requirements of XML network security.

APPENDIX 1: FIX Point Proxy Uses Manual

Overview

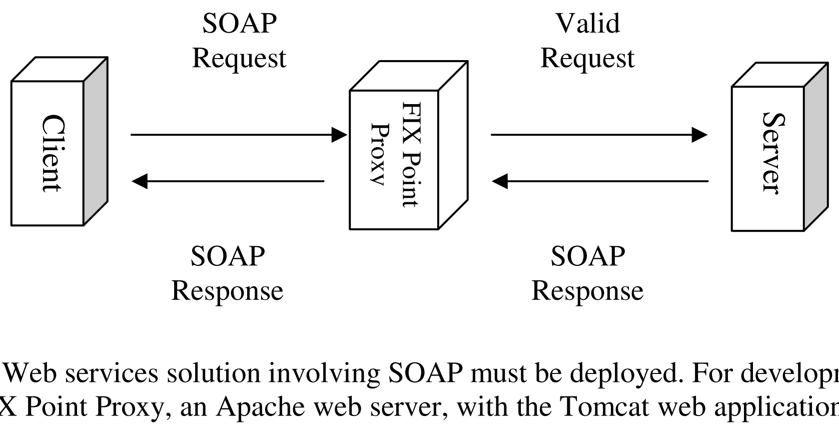
FIX Point Proxy is a light-weight, graphically oriented demonstration system for Web services security. FIX Point Proxy implements a framework for Web services security called *Filtering to Inspect XML (FIX).* The FIX approach to Web services security visualizes distributed Web services applications passing through one or more security *filters*. Locations on a network pathway where such filters are installed are called *FIX points*. The list of configuration options for the available filters at each FIX point is known as a *FIX list*. So FIX Point Proxy is to be installed at one or more network locations between a Web services client and a distributed Web service. At each location, one or more security filters may implemented from the FIX list implemented by the FIX Point Proxy.

Architecture

FIX Point Proxy is based on a simple HTTP proxy server. On top of this HTTP proxy, a number of XML security functions have been added. Use of the FIX Point Proxy assumes the installation of a client-server pair for Web services production and consumption. A variety of configurations are possible to set up such a demonstration. These include:

- client-proxy-server all on one computer, all on the same OS
- client-proxy-server on separate computers, connected by a TCP/IP network
- client-proxy-server each on separate virtual OS images, with one or more virtual images on the same computer (through VMWare or Xen, for example)

There are other ways to mix and match these services. Also, additional servers (such as a SQL database server) can be added to the network. Below is a diagram illustrating a sample network installation for the FIX Point Proxy:



On the server, a Web services solution involving SOAP must be deployed. For development and testing the of FIX Point Proxy, an Apache web server, with the Tomcat web application container, was used for the server. The Axis library (version 1-4) was installed along with Apache/Tomcat server. However, SOAP is vendor neutral, and any server supporting standards-based XML Web services may be used. For example, .NET clients and servers could be used to pass services traffic through the FIX Point Proxy if desired.

Dependencies

FIX Point Proxy itself is written in Java (jre 1.5.0_07). The following libraries (downloadable as .JAR files) must be in the build path for the FIX Point Proxy:

From Axis: (http://ws.apache.org/axis/)

- axis.jar
- axis-ant.jar
- commons-discovery-0.2.jar
- commons-logging—0.1.4.jar
- jaxrpc.jar
- log4j-1.2.8.jar
- saaj.jar
- wsdl4j-1.5.1.jar

Other libraries needed include:

- activation.jar (http://java.sun.com/products/javabeans/jaf/downloads/index.html)
- mail.jar (http://java.sun.com/products/javamail/)

Not all of these libraries are necessarily used directly in the current version of the FIX Point Proxy. However, Axis may throw error messages if any of them are absent. So the recommended best practice is to download all these libraries and include them in the classpath. All of the libraries referenced above are freely downloadable without charge.

File Structure

FIX Point Proxy itself includes the following source files:

- AuthorizationQuery.java
- DomParser.java
- FIXProxy.java
- MessageHandler.java
- PolicySettings.java
- ProxyGUI.java
- SaxParser.java
- XMLHandler.java

All of these files currently ship in a package named "proxy". However, the files can repackaged in any convenient manner, as long as the complied class files are all visible to each other. Here is a brief summary what each of these files contributes to the complete solution:

- AuthorizationQuery.java

SQL query for use with SQL authorization option

- DomParser.java

Used for granular content capture and editing of XML Web services content

- FIXProxy.java

Contains main() method for the application. Forms TCP/IP socket connection with client and server. Forwards messages, depending on the outcome of the security filters applied to the Web service.

- MessageHandler.java

Provides various utilities to extract SOAP messages from the HTTP header. Also modifies HTTP header as needed prior to forwarding Web service.

- PolicySettings.java

Container for properties to control XML security filters. Most of these properties are boolean variables with PolicySettings holding the "on" or "off" state of each filter.

- ProxyGUI.java

Provides graphical user controls for the FIX Point Proxy. Passes values to filters. Captures and displays Web services content. Allows user to modify Web services content through graphical input methods.

- SaxParser.java

Used to implement several filters. Takes its settings from PolicySettings.

- XMLHandler.java

Directs XML traffic to one or both parsers. Calls for XSLT transformation to output results of any content editing performed by the DOM parser.

Installation

To install FIX Point Proxy, simply copy all the files above to a directory on any computer with a Java runtime current enough to handle the libraries listed above. Test installations on Windows XP Pro and Fedora Core Linux 4.0 were both straightforward.

Starting the FIX Point Proxy

FIX Point Proxy can be started from a command line simply by calling its main() method:

```
java proxy.FixProxy
```

To provide any proxy functionality, FIX Point Proxy needs three parameters to be configured. These are:

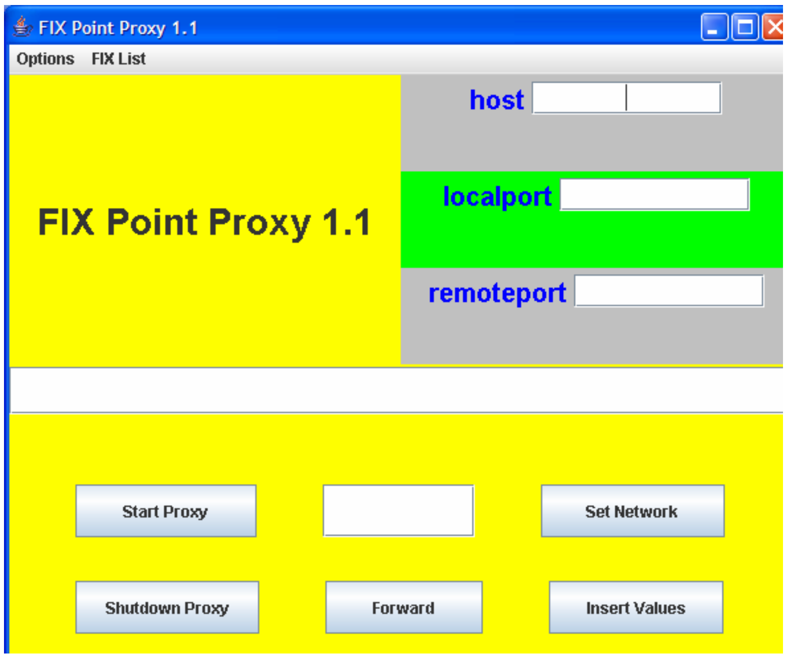
- host
- remoteport
- localport

Host is the IP address of the Web services server.

Remoteport is the TCP port number of the Web services server.

Localport is the TCP port number the FIX Point Proxy will use to listen for requests to the host server.

Each of these settings can be configured through the GUI:



After entering the parameters, select “Set Network”. This will point the FIX Proxy toward the server. Example settings for these parameters are as follows:

Host: 192.168.0.10

Localport: 3000

Remoteport: 8080

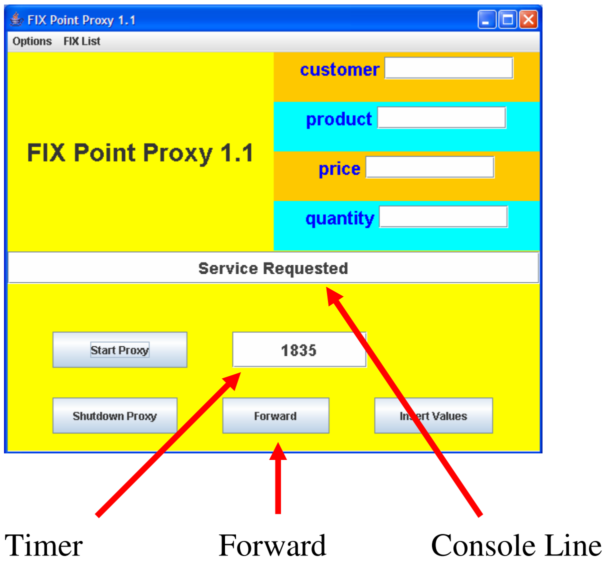
This assumes a server with IP address 192.168.0.10 listening on port 8080. The FIX Point Proxy listens on its own port 3000 and forwards traffic to 192.168.0.10 on 8080.

After entering the three parameters and selecting “Set Network”, the FIX Point Proxy can be started by selecting “Start Proxy”.

<u>Using the FIX Point Proxy</u>.

For full effect, FIX Point Proxy requires connections to a client and server passing SOAP messages. However, some of the functionality of FIX Point Proxy can be tested on HTTP traffic without SOAP.

As an initial such test, choose “Transparent Proxy” from the “Options” menu. In this mode, the FIX Point Proxy will pass HTTP requests to the server. A timer on the proxy will run and the proxy’s console line will indicate a service request has been made.



To permit the HTTP request to go through the proxy, the “Forward” button can be pressed. Another filter that can be applied without a Web services client is WSDL filtering. WSDL filtering is available on the FIX List menu. If “Block WSDL” is selected on the FIX List menu, HTTP requests for files with a wsdl extension will be blocked. This is useful for preventing network reconnaissance against Web services.

<u>Using the FIX Point Proxy with a Web Service</u>.

In the current version, FIX Point Proxy is tightly coupled to Web services with a particular set of parameters. The WSDL file below illustrates the type of Web service that can be supported at this time:

--------------------- begin WSDL ---------------------

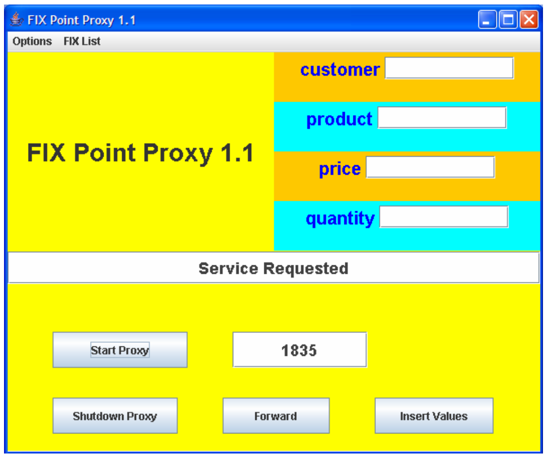
```
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/ConfirmOrder">
–
            <!--
WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)
-->
–
            <wsdl:message name="confirmRequest">
<wsdl:part name="buyerID" type="xsd:string"/>
<wsdl:part name="productID" type="xsd:string"/>
<wsdl:part name="price" type="xsd:int"/>
<wsdl:part name="quantity" type="xsd:int"/>
</wsdl:message>
–
            <wsdl:message name="confirmResponse">
<wsdl:part name="confirmReturn" type="xsd:string"/>
</wsdl:message>
–
            <wsdl:portType name="ConfirmOrder">
–
            <wsdl:operation name="confirm" parameterOrder="buyerID productID price quantity">
<wsdl:input message="impl:confirmRequest" name="confirmRequest"/>
<wsdl:output message="impl:confirmResponse" name="confirmResponse"/>
</wsdl:operation>
</wsdl:portType>
–
            <wsdl:binding name="ConfirmOrderSoapBinding" type="impl:ConfirmOrder">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
–
            <wsdl:operation name="confirm">
<wsdlsoap:operation soapAction=""/>
–
            <wsdl:input name="confirmRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://demo"
use="encoded"/>
</wsdl:input>
–
            <wsdl:output name="confirmResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/ConfirmOrder" use="encoded"/>
</wsdl:output>
```

</wsdl:operation>
</wsdl:binding>
–
            <wsdl:service name="ConfirmOrderService">
–
            <wsdl:port binding="impl:ConfirmOrderSoapBinding" name="ConfirmOrder">
<wsdlsoap:address location="http://localhost:8080/axis/services/ConfirmOrder"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>

---------------------- end WSDL -------------------------------

The field labels on the graphical user interface for the FIX Point Proxy are based on the WSDL file above.

Note: There is no necessary connection between the WSDL part names, the GUI labels, and the actual data. The XML filters in the FIX Point Proxy will look for two strings and two integers. These values can be manipulated to create a variety of attack and defense scenarios.



<u>Deploying a Web service</u>

There are a variety of ways to deploy a Web service in a suitable web application context. The approach below illustrates the method used to test the FIX Point Proxy.

First, develop a simple Web service. For example:

```
package demo;

public class ConfirmOrder {

      String buyerID = "none";
      String productID = "none";
```

```
	int price = -1;
	int quantity = -1;
	int total = -1;

	public ConfirmOrder(){

	}

 public String confirm( String buyerID, String productID, int price,
int quantity ) {
       total = price * quantity;
  return "Order confirmed for: " + "\nCustomer# " + buyerID +
"\nProduct# "+ productID + "\n"+ quantity + " units @ "
+ "$" + price  +"\nTotal is $" + total;}
}
```

This service can be deployed using a command line tool available with Axis. Good documentation on how to use this tool is available at this URL:

Java Reference Guide > Developing Web Services Using Axis and Tomcat.
http://www.informit.com/guides/content.asp?g=java&seqNum=165&rl=1

On deployment, the service will be mounted in Axis and a WSDL file for the service will be available at a URL such as:

http://localhost:8080/axis/services/ConfirmOrder?wsdl

Deploying a Web services client

Based on the service above, a client for the service can be developed. Below is sample code for a client to consume the given service.

```
package demo;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;
import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;

public class ConfirmOrderClient {
	public static void main(String[] args) {
		try {
			Options options = new Options(args);

			String endpointURL = options.getURL();

			String buyerID;
			String productID;
			Integer price;
			Integer quantity;
```

```
                  args = options.getRemainingArgs();
                  if ((args == null) || (args.length < 4)) {
                        buyerID = "NoBuyer";
                        productID = "NoProduct";
                        price = new Integer(0);
                        quantity = new Integer(0);
                  } else {
                        buyerID = args[0];
                        productID = args[1];
                        quantity = new Integer(args[2]);
                        price = new Integer(args[3]);

                  }

                  Service service = new Service();
                  Call call = (Call) service.createCall();

                  call.setTargetEndpointAddress(new
java.net.URL(endpointURL));
                  call.setOperationName(new QName("http://demo",
"confirm"));
                  call.addParameter("arg1", XMLType.XSD_STRING,
ParameterMode.IN);
                  call.addParameter("arg2", XMLType.XSD_STRING,
ParameterMode.IN);
                  call.addParameter("arg3", XMLType.XSD_INT,
ParameterMode.IN);
                  call.addParameter("arg4", XMLType.XSD_INT,
ParameterMode.IN);

      call.setReturnType(org.apache.axis.encoding.XMLType.XSD_STRING);

                  String ret = (String) call.invoke(new Object[]
{ buyerID,
                              productID, quantity, price });

                  System.out.println("ORDER CONFIRMATION \n" + ret);
            } catch (Exception e) {
                  System.err.println(e.toString());
            }
      }
}
```

Note the presence of import statements referencing both the Axis and Javax.xml libraries. The easiest way to get these libraries into the client executable is to build the client in an IDE such as Eclipse. Later, after compilation, the client class files can be exported and installed on a host OS with little difficulty.

The client can simply be copied to a host. From there, it can be run from a command line. (Windows and Linux have both been used in this way as client hosts.) This client requires four parameters. There will be another parameter also, which needs to go before these inputs, to point the client toward its service. For example, the client could be launched as follows:

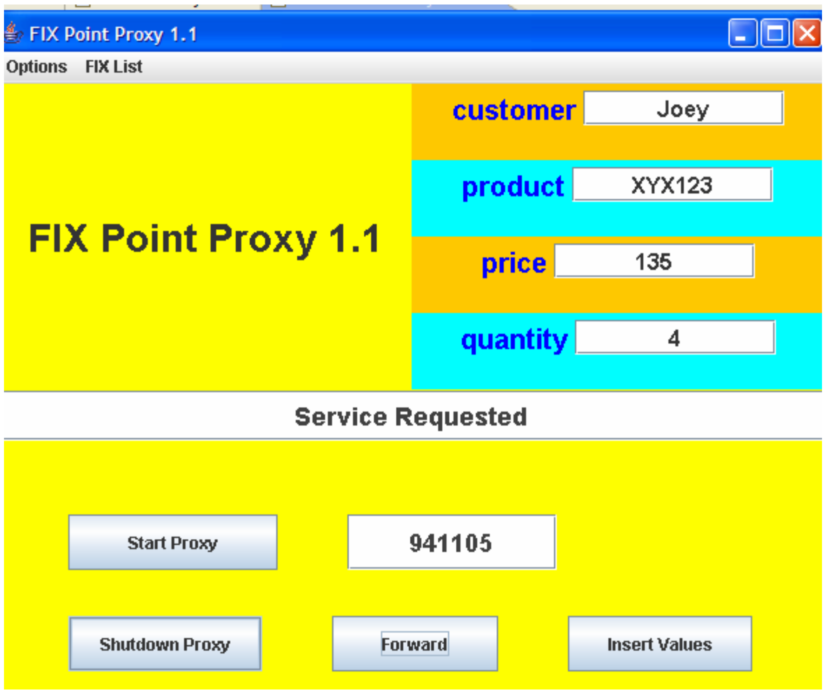
```
java ConfirmOrderClient -lhttp://localhost:3000/axis/services/ConfirmOrder
    Joey  xyx123  100 35
```

Note that the client is pointed to port 3000. The normal port for Axis on Tomcat is 8080. The FIX Point Proxy will block this port. Instead, the client needs to reference whatever port number has been selected as localport on the FIX Point Proxy. Also, the client needs to point to the IP number of the FIX Point Proxy, not the Web services server.

In the absence of any special XML filters being selected, the FIX Point Proxy will acknowledge a service request with the console output: "Service Requested". To pass the service without any filtering, press "Forward". The SOAP request from client to server will then be permitted. In the current version of FIX Point Proxy, all filtering is done on the outbound trip from client to server. So any SOAP response will come back through the FIX Point Proxy without any special processing or notification. While a service request is pending, the timer on the FIX Point Proxy will count down. This is mostly for visual effect. The numbers in the timer have no special meaning. However, after a few trail runs, users should begin to get an idea of how far the timer can count down before the service times out.

Intercepting and Modifying Data

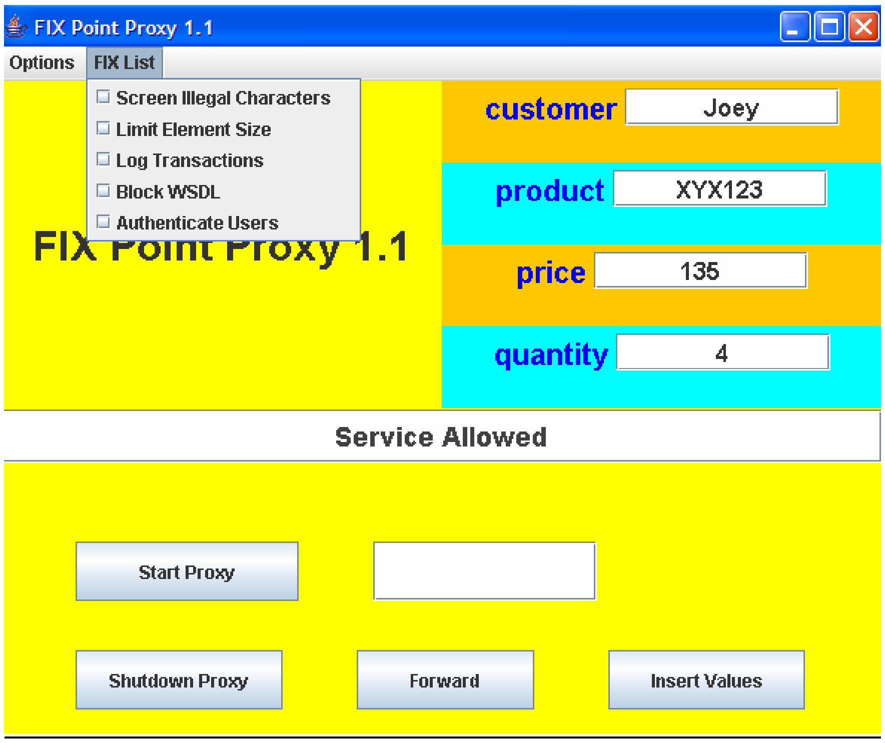
By default, FIX Point Proxy intercepts SOAP requests and presents the values of each SOAP part to the GUI. This is accomplished through the class DomParser. A Document Object Model (DOM) parser analyzes an XML tree and holds all its values in object memory. This allows element values to be modified by the operator of the FIX Point Proxy. For example, given the SOAP request below, the customer "Joey" could be replaced by the customer "Jane". After making any such changes in the input textboxes, press "Insert Values". The new values will be inserted into the DOM. With new values for the SOAP request established in object memory, on "Forward" and XSLT transformation will occur, putting the changed values back into the original SOAP wrapper. Finally, other utility classes in the FIX Point Proxy will attach an HTTP header to the edited SOAP request and send the new data to the Web services server.



customer: Joey
product: XYX123
price: 135
quantity: 4
941105

Applying Filters

A variety of XML security filters are available on the FIX List menu. Below is a summary of what each of them do.



customer Joey
product XYX123
price 135
quantity 4

Service Allowed

- **Screen Illegal Characters**

This filter is used to prevent XML Injection attacks. Element values are limited to the following character ranges: {A .. Z, a .. z, 0 .. 9}. Punctuation such as </ > (used to embed one XML element within another element) is not allowed.

- **Limit Element Size**

This filter is used as one measure against Denial of Service or Buffer Overflow attacks. When this filters is selected, the FIX Point Proxy supplies a console warning for overly long element values. The limit is element size is set in PolicySettings.

- **Log Transactions**

Writes a summary of each SOAP request to a log file named logfile.txt

- **Block WSDL**

Warns user on each HTTP request for a URL with a wsdl extensions. Used to prevent WSDL discovery.

- **Authenticate Users**

Checks the contents of the “customer” field against a list of authorized users. There are two current options for checking user authorization.

> ➢ A default test case is written into the class SaxParser. This test case uses methods such as the examples below to populate ArrayLists of authorized and unauthorized users:

```
authorized.add("Jim");
unauthorized.add("Joey");
```

> ➢ A MySQL server can be used to maintain a table of user authorizations. This option is invoked from the Options menu by selecting “Authentication Database”. When using this option, a MySQL server must be installed and the proper JDBC libraries must be obtained. Within the class AuthorizationQuery proper parameters for the JDBC connection must be entered. An example configuration of this connection is:

```
public static void connect() {

            try {
                  String userName = "root";
                  String password = "password";
                  String url = "jdbc:mysql://127.0.0.1/test";
                  Class.forName("com.mysql.jdbc.Driver").newInstance();
                  conn = DriverManager.getConnection(url, userName,
                  password);
                  System.out.println("Database connection established");
            } catch (Exception e) {
                  System.err.println("Cannot connect to database
                  server");
            } finally {
                  if (conn != null) {
                        try {
                              authorizationQuery();
                              conn.close();
                              System.out.println("Database connection
                              terminated");
                        } catch (Exception e) {
                        /* ignore close errors */
                        }
                  }
            }
      }
```

AuthorizationQuery assumes the existence of a MySQL table to hold settings for user authentication. The sample queries below suggest the structure of such a table:

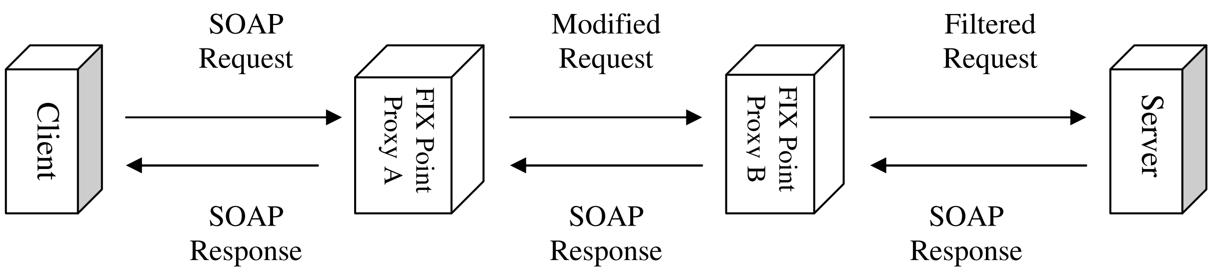
```
public static void authorizationQuery() throws Exception {
            //connect();
            Statement s = conn.createStatement();
            s.executeQuery("SELECT * FROM users WHERE status =
'allow'");
            ResultSet rs = s.getResultSet();
            while (rs.next()) {
                  String result = rs.getString(1);
                  authorized.add(result);
            }

           s.executeQuery("SELECT * FROM users WHERE status = 'deny'");
            rs = s.getResultSet();
            while (rs.next()) {
                  String result = rs.getString(1);
                  unauthorized.add(result);
            }
            rs.close();
            s.close();
            Iterator iter = authorized.iterator();
            while (iter.hasNext()) {
                  System.out.println(iter.next() + ": authorized");
            }

            iter = unauthorized.iterator();
            while (iter.hasNext()) {
                  System.out.println(iter.next() + ": unauthorized");
            }
      }
```

Chaining FIX Point Proxies

FIX Point Proxy can be deployed in a variety of network locations. This allows for complex message handling behavior to be modeled. For example, two FIX Point Proxies could be inserted in the middle of SOAP client/server pair.



In the scenario above, FIX Point Proxy A could be used to modify a normal SOAP request by inserting malicious content. FIX Point Proxy B could be used to protect the server by filtering for such content. This is but one example of the many demonstrations that can be constructed using FIX Point Proxy.

REFERENCES

Bebawy, R., Sabry, H., El-Kassas, S., Hanna, Y., & Youssef, Y. (2005). Nedgty: Web services firewall. Paper presented in Proceedings of the 2005 IEEE International Conference on Web Services(ICWS'05), 00, 597-601.

Beznosov, K., Flinn, D. J., Kawamoto, S., & Hartman, B. (2005). Introduction to web services and their security. Information Security Technical Report, 10, 2-14.

Bishop, M. (2003). Computer Security. Boston: Addison-Wesley.

Contos, B. (2006). The intersection of Sarbanes-Oxley and insider threats. Retrieved May 22, 2007 from: http://www.computerworld.com/networkingtopics/networking/story/0,10801,109527,00.html

CERT Coordination of the Carnegie Mellow Software Engineering. (2006). Retrieved May 20, 2006, from : http://www.cert.org

Cremonini, M., Damiani, E., De Capitani di Vimercati, S., & Samarati, P. (2003). An XML-based approach to combine firewalls and web services security specifications. ACM Workshop on XML Security, 69-78.

Clark, D. & Wilson, D. (1987). A comparison of commercial and military security policies. In 1987 IEEE Symposium on Security and Privacy, 184-194.

Damiani, E., De Capitani di Vimercati, S., & Samarati, P. (2002). Towards securing XML web services. ACM Workshop on XML Security, 90-96.

Department of Homeland Security. (2006). National Vulnerability Database of the National Cybersecurity Division. Retrieved May 20, 2006, from : http://nvd.nist.gov

Epstein, J., Matsumoto, S., & McGraw, G. (2006). Software security and SOA: Danger, Will Robinson! IEEE Security and Privacy, 4(1), 80-83.

Farlex, Inc. (2007). fix - definition of fix by the Free Online Dictionary, Thesaurus and Encyclpedia . Retrieved March 26, 2007, from : http://www.thefreedictionary.com/fix

Fernandez, E. B., & Delessy, N. (2006). Using patterns to understand and compare web services security products and standards. Paper presented in Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), 157-157.

Heasley, J. (2004). Securing XML data. Paper presented in InfoSecCD Conference'04, 112-114.

Holgersson, J., & Söderström, E. (2005). Web service security - vulnerabilities and threats with the context of WS-Security. Paper presented in The 4th Conference on Standardization and Innovation in Information Technology, 138-146.

International Data Group. (2006). CIO and Computerworld research: The forecast for SOA. Retrieved May 11, 2006, from : http://www2.cio.com/research/surveyreport.cfm?id=106

Kobielus, J. (2006). SOA platform vendors will own ESB market. Retrieved May 11, 2006, from : http://www.networkworld.com/columnists/2006/021306kobielus.html

Koyama, K., Iguchi, K. J., Hosono, S., & Fujita, S. (2005). Web services proxy: An extensible platform for intermediaries of XML networks. Paper presented in Proceedings of the IEEE Conference on Web Services (ICWS'05), 00, 246-253.

Myerson, J. (2002). Advancing the Web services stack. Retrieved May 22, 2007 from: http://www-128.ibm.com/developerworks/webservices/library/ws-wsa/

Nakamura, Y., Tatsubori, M., Imamura, T., & Ono, K. (2005). Model-driven security based on a web services security architecture. Paper presented in Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05), 1, 7-15.

SANS Institute. (2006). Retrieved May 20, 2006, from : http://www.sans.org

Shah, S. (2007). Hacking web services. Boston: Charles River Media.

Skalka, C., & Wang, X. (2004). Trust by verify: Authorization for web services. Paper presented in ACM Workshop on Secure Web Services, 47-55.

Stamos, A., & Stender, S. (2005). Web services security. Retrieved May 11, 2006, from : http://www.owasp.org/docroot/owasp/misc/OWASP_DC_2005_Presentations/Track_1-Day1/AppSec2005DC-Alex_Stamos-Attacking_Web_Services.ppt#275,11,Code Breaks Free...

Steel, C., Nagappan, R., & Lai, R. (2006). Core security patterns: Best practices and strategies for J2EE, web services, and identity management. Upper Saddle River, NJ: Pearson.

Tari, Z., Bertok, P., & Simic, D. (2006). A dynamic label checking apporach for information flow control in web services. International Journal of Web Services Research, 3(1), 1-28.

W3C (2001). Web Services Description Language (WSDL) 1.1. Retrieved May 22, 2007 from: http://www.w3.org/TR/wsdl#_soap-e

Yuan, E., & Tong, J. (2005). Attributed based access control (ABAC) for web services. Paper presented in Proceedings of the 2005 IEEE International Conference on Web Services(ICWS'05), 00, 561-569.

Zimmerman, H. (1980). OSI reference model – the ISO model of architecture for open systems connection. IEEE Transactions on Communication 28(4), 425-432.

ABOUT THE AUTHOR

Robert Bunge is a graduate student at the University of Washington Tacoma in Tacoma, Washington. Also, he is a faculty member at DeVry University in Federal Way, Washington. A former technical administrator, he teaches network communication management for DeVry. He also holds masters degrees in History and Education from Stanford University and a bachelors degree in History from the University of Washington.

Sam Chung is an Assistant Professor of CSS at the University of Washington Tacoma in Tacoma, Washington. He is directing his Intelligent Service-Oriented Computing (ISOC) research group and the Applied Distributed Computing laboratory at the UW Tacoma. He received his PhD in CS from University of South Florida in Tampa, Florida. His current research focuses on SoSR (Service-Oriented Software Reengineering) and intelligent service broker.

Barbara Endicott-Popovsky is Director of the Center for Information Assurance and Cybersecurity, with a joint faculty appointment in the Information School and the University of Washington Institute of Technology, Tacoma, as well as an affiliate appointment in the School of Architecture/Dept. of Urban Planning MS in Critical Infrastructure Protection Program. Her research involves developing approaches that support emerging trends in network forensic investigations, defining information system security strategies to include requirements for meeting courtroom admissibility standards, etc. She has a BA in Liberal Arts from the University of Pittsburgh, 5th year in Accounting and an MBA from the University of Washington, an MS in Information Systems Engineering from Seattle Pacific University and is a PhD in Computer Science from the University of Idaho.

Don McLane is a lecturer and the manager of the Information Assurance laboratory at the University of Washington Tacoma in Tacoma, Washington.