An Operational Framework for Service Oriented Architecture Network Security

Robert Bunge
Network Systems Administration
DeVry Univ. Federal Way
Federal Way, WA, USA
rbunge@myuw.net

Sam Chung
CSS
Institute of Technology
Univ. of WA Tacoma
Tacoma, WA, USA
chungsa@u.washington.edu

Barbara Endicott-Popovsky
Center for Information Assurance & Cybersecurity
Univ. of WA
Seattle, WA, USA
endicotta@u.washington.edu

Don McLane
CSS
Institute of Technology
Univ. of WA Tacoma
Tacoma, WA, USA
dmclane@u.washington.edu

Abstract

This study proposes a new operational framework of a network administrator 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.

1. 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.

2. Previous Efforts

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

1530-1605/08 $25.00 © 2008 IEEE

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, which is shown in Table 1.

Table 1. Summary of Recent Researches for SOA Security

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

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.

SOA Security Techniques: Skalka and Wang (Skalka & Wang, 2004) propose a "trust-but-verify" approach, in which authorization is separated into different online and offline phases. 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). 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. 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. In short, it appears that deployment of XML-aware network edge systems would facilitate implementation of an ABAC model for SOA security.

Table 2. Operational Impact Of Different Security Approaches

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

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. 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. 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). 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. 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. 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.

3. Filtering to Inspect XML (FIX) Model

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 for network administrators.

3.1. What is the FIX model?

The proposed framework begins with an acronym: FIX. FIX stands for Filtering to Inspect XML. The FIX model proposes a related nomenclature to facilitate analysis, modeling, planning, discussion, and deployment of XML-based security systems. This nomenclature includes: FIX, FIX point, and FIX list. The acronym "FIX" may be used as a verb, including "to FIX", meaning to "filter and inspect XML". The term FIX point is 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.

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

3.2. Why is the FIX model needed?

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.

Table 3. Network Layers, Corresponding Protocols, And Security Mechanisms

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

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 network administrators. 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.

Table 4. Problem Of Locating Web Services In A Network Reference Model

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

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) is 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? This problem is complicated further by the ever-growing complexity of protocols and specifications within Web services.

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.

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, which we call Application Layer: TCP/IP sublayer.. This would seem to confirm even more that Web services must need their own new network layer.

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 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.

Table 5. Application Layer: XML and TCP/IP Sublayers

Network Layer	Protocols	Security Technique
Application Layer: XML Sublayer	SOAP, WSDL, WS-Security, UDDI, 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

3.3. FIX List

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. There is no RFC to specify the length of a standard XML document. 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? 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.

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:

Table 6. An Operational Framework For XML Network Security

Network Layer	Protocols	Security Technique
Application Layer: XML Sublayer	SOAP, WSDL, WS-Security, UDDI, 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

4. Testing

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.

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.

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 1.

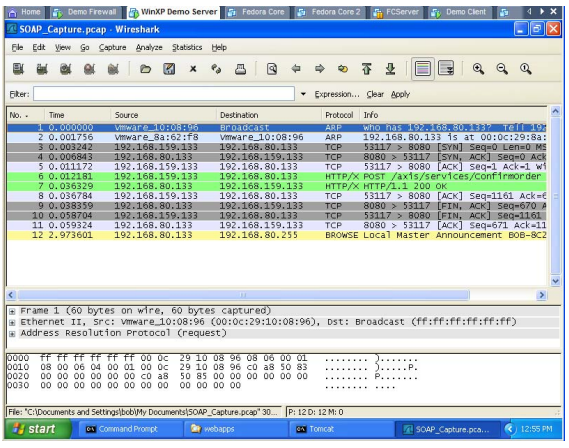

Figure 1. SOAP traffic captured between virtual OS images

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, which is shown in Figure 2.

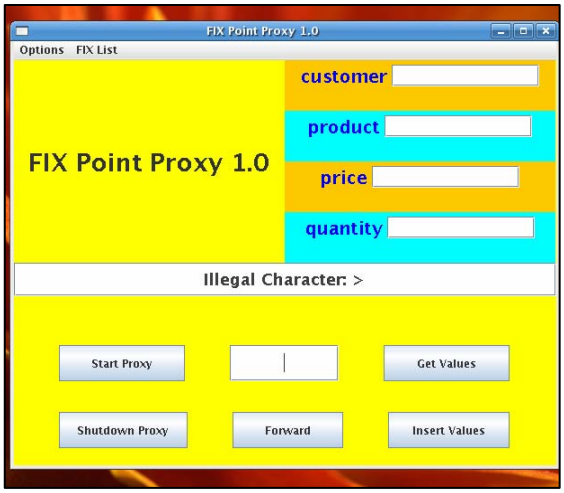

Figure 2. The Interface of FIX Point Proxy

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, which is described in Table 7 in detail. Filters are implemented as configurable options that can be invoked or disabled by network administrative personnel 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.

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, which is shown in Figure 3. 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.

Table 7. Selected Filters from a Prototype FIX System

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.

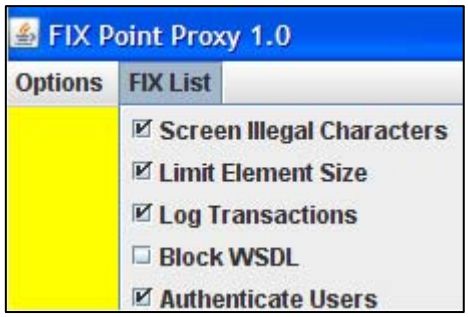

Figure 3. Implementation of a FIX List via Drop-down Menu

5. Experimental 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.

5.1. 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. The WSDL file of the invoked Web service 'ConfirmOrder', its invocation with parameters such as 'buyerID', 'productID', 'price', and 'quantity'. The total price for the ordered product of the customer is displayed with the entered input data.

WSDL for Use Case #1

```
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/services/Confir
mOrder">
  <!--
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/ConfirmOrde
r" 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

5.2. Use Case #2: Man in the Middle

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.

A 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. For example, server accepts the inputs and multiplies price by quantity. The quantity was modified from 4 to 40 in this case. 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

6. Conclusions and Future Research

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.

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.

References

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

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

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.

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.

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

Nakamura, Y., Tatsubori, M., Imamura, T., & Ono, K. (2005). Model-driven security based on 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

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

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 approach for information flow control in web services. International Journal of Web Services Research, 3(1), 1-28.

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.