An Architecture for XML/Web Service Firewall Module

Shahriar Mohammadi
mohammadi@kntu.ac.ir
Mojtaba Khaghani Milani
mojtaba.khaghani@gmail.com
Department of Information Technology,
Khajeh Nasir University, Tehran, Iran

Abstract

Organizations in e-commerce and business transactions are using Web Services more today. So the security of Web Services has become a challenging issue. Security mechanism like traditional firewalls even lots of web application firewalls can't protect service oriented architecture environment against attacks. Firewalls namely XML firewalls are responsible for providing security of this environment. These firewalls in the first step should be aware of content of SOAP message and XML and can protect Web Services against attacks. In the second step, they should provide a secure communication using encryption and digital signatures and control access to services. They should manage the key exchange and provide trusted communications. In the third step, they should support other Web Service specifications and standards. In this article, we introduce architecture and Design of an XML firewall

Keywords: Attack, Network Security, SOA, Web Services, Web Service Security, XML Firewall

1. Introduction

Web service as an implementation of the service oriented architecture (SOA) has brought new capabilities and new benefits. Web service has been designed to support interoperable machine to machine interaction over a network. A web service consists of three components: WSDL, SOAP and UDDI.

WSDL is a XML-based language which provides an interface to describe functionality of the web service. SOAP is the protocol describing the actual request and response which is transferred between web services. UDDI is the platform independent registry and a directory for businesses to locate, publish and find their web services. All of three components of web service are based on XML. XML is platform independent language. It means that an application is written with java can communicate with an application which is written with C++.

While providing advanced business functionality, web services introduce significant security considerations and challenges [1]. For example new kind of threats created, which we address some of these threats and attacks in separate section.

The Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium (W3C) have standardized several specifications related to security in Web services and XML [2]. As an example of these standards, we can mention XML-Encryption or WS-Trust. These standards provide confidentiality and Integrity of web service invocations.

In addition to these specifications, for protecting web services against threats and attacks; we need a new kind of firewalls. Traditional firewalls are not content-aware; furthermore, lots of web application firewalls are not XML-aware. We need a firewall which can scan deep into packets and not only prevents web service from general web application threats, but also parses XML/SOAP messages and prevents XML/web service specific attacks. These firewalls are referred to as XML firewalls.

The importance of XML firewall can be understood easily with a quick look at XML firewall products of business organizations. The popular industry name of XML firewalls is "XML Security Gateway"[1]. For example Cisco ACE XML Gateway, IBM Web Sphere Data Power XML Security Gateway XS40, Vordel Company XML Gateway, Layer 7 Company XML Networking Gateway, Intel SOA Expressway, Xtradyne WS-DBC XML/SOAP Security Gateway and Forum System SENTRY XML Gateway that previous name of this product was XWall XML firewall. Nearly all of company's products have hardware appliance support. In 2005 IBM acquired "DataPower" manufacturer of network devices. DataPower was recognized early as an innovator in XML processing which is a critical element of service oriented architecture (SOA). Except XS40 XML Gateway, IBM has four other SOA appliances. At the same year, Intel acquired Sarvega XML appliance and now it has SOA products division. SOA Expressway is Intel's XML Gateway product. Acquiring XML Companies with such a big companies like IBM and Intel and existence of other big companies like Cisco in this scope expresses the importance of xml firewalls.

The remainder of this article organized as follows: in section 2 an overview of related works is mentioned. Section 3 presents attacks on XML and web services. Section 4 is architecture and design of the XML firewall. Section 5 presents a case study to verify our work. In section 6 future works and enhancement is proposed. Finally, in the section 7 we give a conclusion of this article.

2.Related Works

Haiping Xu and his colleagues have done lot of works on XML firewall. In one of their works, they develop a state-based XML firewall at the application level. Their approach supports Role-Based Access control (RBAC) for users and detection of XML-Based attacks. In their XML firewall, access permissions to Web Service are only granted to users who are authenticated and authorized. They develop a detailed design by defining state-based information, user information, and various access control policies and detection rules. Finally, to demonstrate the effectiveness of their approach, they implement a prototype state-based XML firewall for efficient detection of XML-based attacks to a test system [3].

Another useful work is about Web Service attacks. It gives a survey of vulnerabilities in the context of Web Service. As a proof of the practical relevance of the threats, exemplary they performed implementation of attacks on widespread Web Service. Further, they discuss about general countermeasure for prevention and mitigation of such attacks [4].

Prof. Fernandez and his colleagues at the University of Florida Atlantic have valuable works on Application layer and XML firewalls patterns and designs. They present two patterns for Web Services: 1) a Security Assertion Coordination pattern that coordinates authentication and authorization using Role-Based Access Control and 2) A pattern for XML firewalls that filters XML messages or documents according to institution policies [5].

These works helped us for modeling and making architecture of XML firewall module. Some of these works emphasized on access control, but our work in this project is not concern in access control and emphasizes on protecting other attacks and practical considerations.

3. XML AND WEB SERVICE ATTACKS

The typical requirements for a secure system are integrity, confidentiality and availability. Any action targeting at violation of one of these properties is called an attack, the possibility for occurring an attack is called vulnerability. It is essential to understand different types of attacks on Web Services for developing a reliable and concrete XML based firewall. We categorized the attacks which we've studied in this paper as follow:

3.1XML Based Attacks

The first step has taken by a Web Service after receiving a SOAP message request is to read through, or parse the elements to extract parameters and methods to invoke. These processes allows for special kinds of attacks such as Coercive parsing, oversized payload attack.

3.1.1.Coercive parsing

Xml supports rich and complex document structures, including recursion, which can potentially cause infinite loop during processing input. Coercive parsing uses XML complexity for exhaustion CPU and memory web Server. While denial of service attacks usually requires a large numbers of massages, coercive parsing attack can be launched on a web service with a single 2KB malformed XML massage as shown below:

```
<x>
    <x>
        <x>
            …
```

3.1.2.Oversized Payload

XML parsing is directly affected by the size of the SOAP message. The total memory usage caused by processing one SOAP message is much higher than just the message size. This is due to the fact that most Web Service frameworks implement a tree-based XML processing model like the Document Order Model [4]. Using this model, an XML document – like a SOAP message – is completely read, parsed and transformed into an in-memory object representation, which occupies much more memory space than the original XML document. As a result, large amounts of CPU cycles and memory space are consumed when presented with large documents to process. A hacker can send a payload that is excessively large to deplete systems resources as shown in below.

```
<Envelope>
    <Body>
        <getArrayLength>
            <item>x</item>
            <item>x</item>
            <item>x</item>
            ...
        </getArrayLength>
    </Body>
</Envelope>
```

Oversized payload attack which results in CPU and memory exhaustion [4]

3.2Brute Force Attack

These attacks are characterized by ‘brute force’ which crack using combinations of letters, numbers and special characters in an attempt to determine which service requests and operations with what parameters

result in a security Breach. Examples of these attacks are WSDL scanning and parameter tempting.

3.2.1. WSDL Scanning

Since the WSDL document includes all of the operations that are available to the consumer, it is straightforward for a hacker to run through all of the operations with different message request patterns until a breach is identified. This "knocking on every door until one opens" approach is usually effective when poor Programming practices are employed or simply the result of operations that were excluded from published WSDL documents yet are still up and running for some reason [6].

3.2.2. Parameter Tempting

In a WSDL document, the parameters for the operations are defined. If the attacker tries to send requests with different parameter patterns, the Web Service has the possibility to crash. For example, by submitting special characters (or other unexpected content) the back-end implementation of the Web Service can be crashed, regardless of data validation.

3.3Injection Attack

In these attacks, an attacker inserts, removes or modifies information within a message to deceive the web server which can lead to undesired effects. Examples of these attacks are XML Injection, SQL Injection and SOAP action spoofing.

3.3.1.XML Injection

An XML Injection attack tries to modify the XML structure of SOAP by inserting content – e.g. operation parameters – containing XML tags. The modified XML may allow an attacker to alter the structure of xml document which result in change of behavior of web server as shown below.

```
<Envelope>
    <Body>
        <HelloWorld>
            <a> <b>1</b> </a>
            <b> 2 </b>
        </HelloWorld>
    </Body>
</Envelope>
```

Xml Inject of 1 result in change of value of a and b in web server [4].

3.3.2.SQL Injection

Structured Query Language (SQL) is used to manipulate database's records. SQL statements can be inserted into input of a SOAP message by an attacker. If the statements are not sanitized, the attacker may gain access to the databases. This attack uses SQL commands such as SELECT, CREATE, UPDATE, DELETE, DROP, 1=1, ALTER and INSERT.

3.3.3.SOAP Action Spoofing

The actual Web service operation addressed by a SOAP request is identified by the first child element of the SOAP body element. Additionally, the optional http header field "SOAP Action" can be used for operation

identification. Although this value only represents a hint to the actual operation, the SOAP Action field value is often used as the only qualifier for the requested operation. This operation identification enables attacker to invoke an operation different from the one specified inside the SOAP body. It is based on modification of the HTTP header as shown in below.

```
POST /Service.asmx HTTP/1.1
...
SOAPAction: "op2"
<Envelope>
    <Body>
        <op1>
            <s>Hello</s>
        </op1>
    </Body>
</Envelope>
```

The method call that was triggered by this message was op2 instead of op1 [4].

3.4DOS Attack

DOS attack attempts to slow down or completely shut down the web server so as to disrupt the service and deny the legitimate and authorized users to access it, an example of DOS attack is Replay Attacks.

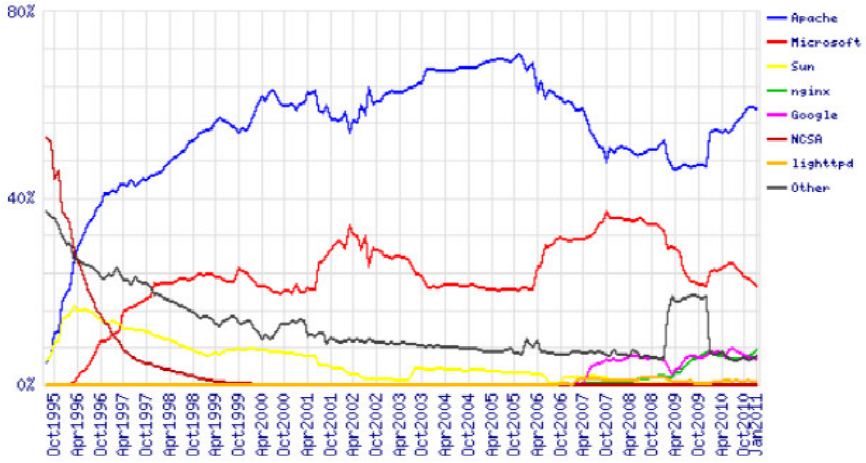
3.4.1.Reply Attack

An attacker attempts to resend SOAP requests to repeat sensitive transaction and overload the web service, similar to network ping of death [1].

4. ARCHITECTURE AND DESIGN

We have chosen Apache HTTPD for designing and implementing our XML firewall. Apache HTTPD is the most popular web server in the world. As Figure 1- Market Share for Top Servers Across All Domains shows, Netcraft survey on the response of 273,301,445 sites shows that 59.13% of web servers using Apache HTTPD [7].

The first characteristic of Apache is widespread use. The second characteristic is the modularity of Apache. Apache supports variety of features as compiled module which can extend the core functionality. These modules include some common languages as a module like Perl, PHP and Python. Another module supporting SSL is "mod_ssl", module implementing proxy/cache/gateway for Apache is "mod_proxy" and there are lots of other great modules. Apache module developers can use other Apache modules in writing his module with mechanisms provided in Apache. The third characteristic of Apache is performance. Apache provides Multi Processing Modules (MPMs) and provide multi-threading to reduce latency and increase throughput.



Developer	December 2010	Percent	January 2011	Percent	Change
Apache	151,516,152	59.35%	161,591,445	59.13%	-0.23
Microsoft	56,723,544	22.22%	57,392,351	21.00%	-1.22
nginx	16,910,205	6.62%	20,504,634	7.50%	0.88
Google	14,933,865	5.85%	15,112,532	5.53%	-0.32
lighttpd	1,308,935	0.51%	1,866,872	0.68%	0.17

Figure 1- Market Share for Top Servers Across All Domains

Apache can be used as a reverse proxy by enabling mod_proxy. In this situation Apache receives packets and forwards them to the backend servers. Our module can be used in proxy mode and filter out malicious packets.

Our Architecture and design is based on the Apache structure of module development. Architecture of our recommended XML firewall module illustrate in Fig 2.

Apache provides API for developers. Developers can use API functions as hook mechanism to run module functions in appropriate point of Apache execution cycle. Also module developers can use some data structures from Apache API. For example "request_rec" is request record data structure that provides any information about request includes request header parameters, hostname, request method, content length, pointer to previous request when redirecting and lots of other parameters [8].

We have two configuration components and a request processing component in the architecture. The first configuration component is per server configurations component. This component realizes configurations per server virtual host. The other configuration component is per request configurations component. Request processing components is the major component of our module. But how this architecture components works? In another words, we want to explain algorithm and design of our module.

When Apache starts execution, it executes per server and request configurations. In this phase, module defines registered hooks, initiates XML parser and parses WSDL file defined with Web Services. Also Apache executes other pre-configuration functions like obtaining rules from rule data base, parsing them and maintaining them in an appropriate data structure for using in processing transactions. Also Apache initiates module's data structures.

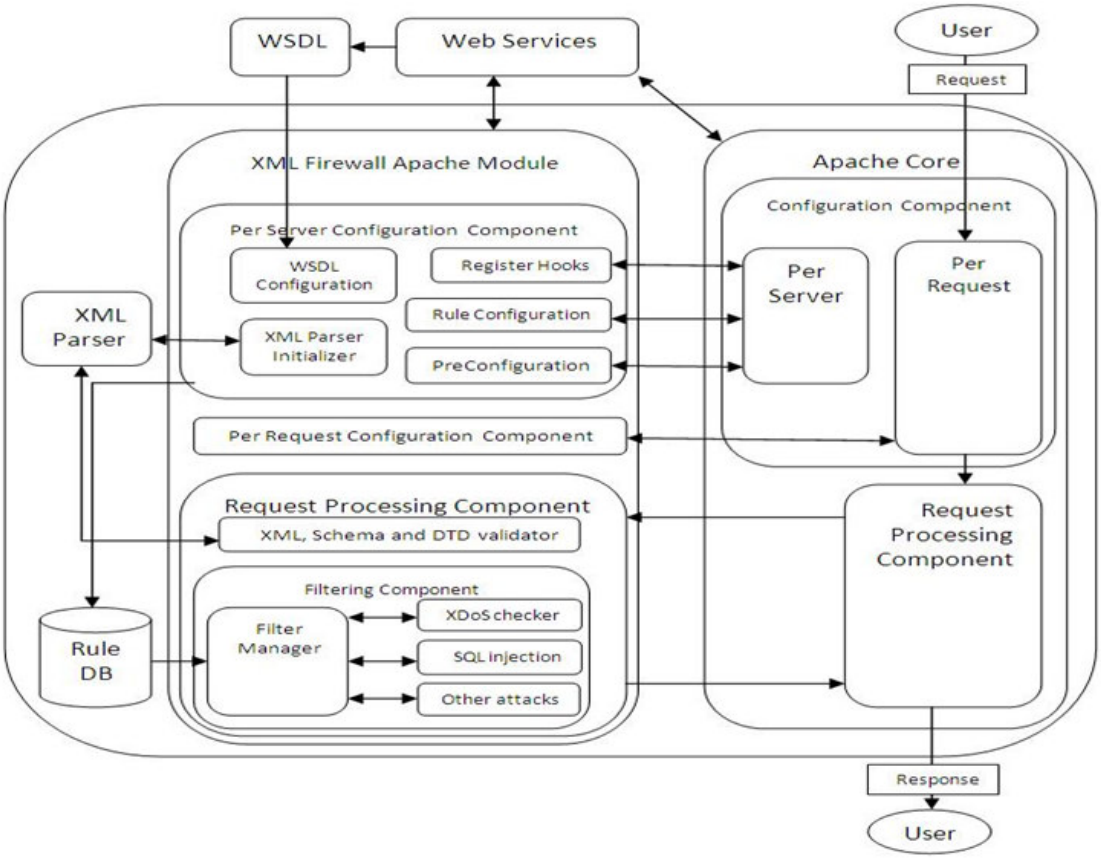


Figure 2- Architecture of recommended ML Firewall Apache Module

After the execution of these per server and request configurations, Apache starts listening. It waits for the requests from clients. When the request is received; request processing component will handle the job as XML firewall. This component uses following process. It scans, filters, validates, cleans, audits and logs the transaction. In this component, module scans requests for malicious content and attacks that some of them mentioned in previous section including XPath injection, SQL injection, XML injection, SOAP Action spoofing and parameter tampering. Additionally, the component validates requests by making sure they conform to the contract, verify property types and sizes etc. When a request is defined as problematic request, then the component audits and logs the request and then decides whether to filter it out or cleans or changes the content of the request and let it through [9].

5. CASE STUDY

This section explains a case study which tests and validates our design and implementation of XML firewall. We use two products of the Open Web Application Security Project (OWASP) for testing: WebScarab and WebGoat.

WebScarab operates as an intercepting proxy, allowing the operator to review and modify requests before they are received by the server, and review and modify responses which is returned from the server before they are received by the client. The requests and responses in this section are intercepted with WebScarab.

WebGoat application is designed to illustrate typical security flaws within web-applications. It has lessons

which are intended to teach a structured approach using for testing and exploiting such vulnerabilities within the context of an application security assessment. We used one of WebGoat's lessons for Web Service SQL injection attack.

For briefness, we have chosen just Web Service SQL injection attack for testing. This attack examined in the scenario in three steps.

This case study has three steps. In the first step we present normal request and normal response. In the second step, we present attack request and response without the presence of XML firewall. In the third step, we enable our XML firewall module. We examine attack request and see our XML firewall response. It should prevent attack and generate appropriate response.

5.1 First Step

In the normal situation, user with the ID number 101 requests to get his credit card number. And web service returns his credit card number in the response. Request and Response is based on the WSDL of web service. The SOAP Envelope, body and the parameter values are shown in the figures. Figure 3 shows normal request and figure 4 shows the response.

```
POST http://127.0.0.1:8080/WebGoat/services/WsSqlInjection HTTP/1.0
Accept: application/soap+xml, application/dime, multipart/related, text/*
Host: 127.0.0.1:80
Content-Type: text/xml; charset=utf-8
SOAPAction: "getCreditCard"
Content-length: 546
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

<?xml version='1.0' encoding='UTF-8'?>
<wsns0:Envelope
  xmlns:wsns1='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:wsns0='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
  <wsns0:Body
    wsns0:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
    <wsns1:getCreditCard
          xmlns:wsns1='http://lessons.webgoat.owasp.org'>
            <id   xsi:type='xsd:string'>
              101
            </id>
    </wsns1:getCreditCard>
  </wsns0:Body>
</wsns0:Envelope>
```

User ID

Figure 3 -Normal Request

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Sat, 12 Feb 2011 08:39:51 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <ns1:getCreditCardResponse
        xmlns:ns1="http://lessons.webgoat.owasp.org">
            <getCreditCardReturn xsi:type="xsd:string">
                2234200065411
            </getCreditCardReturn>
        </ns1:getCreditCardResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

User Credit Card Number

Figure 4- Normal Response

5.2 Second Step

In this step, request is a malicious request. It contains an exploit "101 or 1=1"instead of normal user ID. By putting this exploit in the SQL query in the web service; tautology of 1=1 will cause the query to return all of the entry of the table of user credit card numbers. The request and the response of successful attack are shown in the figure 5 and figure 6.

```
POST http://127.0.0.1:8080/WebGoat/services/WsSqlInjection HTTP/1.0
Accept: application/soap+xml, application/dime, multipart/related, text/*
Host: 127.0.0.1:80
Content-Type: text/xml; charset=utf-8
SOAPAction: "getCreditCard"
Content-length: 556
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=

<?xml version='1.0' encoding='UTF-8'?>
<wsns0:Envelope
  xmlns:wsns1='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:wsns0='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
  <wsns0:Body
    wsns0:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
    <wsns1:getCreditCard
          xmlns:wsns1='http://lessons.webgoat.owasp.org'>
              <id   xsi:type='xsd:string'>
                101 or 1=1
              </id>
    </wsns1:getCreditCard>
  </wsns0:Body>
</wsns0:Envelope>
```

SQL Injection Exploit

Figure 5 - Malicious Request

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Sat, 12 Feb 2011 08:40:38 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <ns1:getCreditCardResponse
        xmlns:ns1="http://lessons.webgoat.owasp.org">
            <getCreditCardReturn xsi:type="xsd:string">2234200065411</getCreditCardReturn>
            <getCreditCardReturn xsi:type="xsd:string">2435600002222</getCreditCardReturn>
            <getCreditCardReturn xsi:type="xsd:string">4352209902222</getCreditCardReturn>
            <getCreditCardReturn xsi:type="xsd:string">3334987033336</getCreditCardReturn>
            <getCreditCardReturn xsi:type="xsd:string">3333000033334</getCreditCardReturn>
            <getCreditCardReturn xsi:type="xsd:string">3341300333328</getCreditCardReturn>
            <getCreditCardReturn xsi:type="xsd:string">3388934533335</getCreditCardReturn>
            <getCreditCardReturn xsi:type="xsd:string">3384345353324</getCreditCardReturn>
        </ns1:getCreditCardResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

All Users Credit Card Number

Figure 6 - Successful Attack Response

5.3 Third Step

In this step, XML firewall module is enabled. Request is the same malicious request of second step. The response is shown in the figure 7 which is generated by the XML firewall module.

```
HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Sat, 12 Feb 2011 08:38:42 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>
                soapenv:Server.parameterException
            </faultcode>
            <faultstring>
                Invalid request with inappropriate values.
            </faultstring>
            <detail>
                <ns1:hostname xmlns:ns1="http://xml.apache.org/axis/">
                    Mohammadi
                </ns1:hostname>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

XML Firewall fault Response

Fig 1 : XML Firewall Response

6. FUTURE WORKS AND ENHANCEMENTS

Future works and enhancements include:

1. Implementing other attacks mentioned in section 4. It may need adding components to our architecture or rewriting some components depending on the attack.
2. Developing a user interface for graphical representation of reporting and showing attacks.
3. Adding access control enforcement section to our XML firewall architecture and implementing it.
4. Implementing web service and XML specifications. For example, encryption and digital signature support.
5. Using Apache utilities for performance enhancement.

7. Conclusion

In this article, we've discussed about the results of designing and implementation XML firewall. The results of the research were developed as an Apache module as first open source XML firewall. Our XML firewall module gets WSDL files and parses it with its XML Parser component. It is SOAP aware and can verify SOAP messages from WSDL file. Also it scans, filters, validates, cleans, audits and logs the transaction. The firewall scans content of messages and filters malicious contents and prevents attacks. For showing that our XML firewall works properly, we've tested the firewall with the case study. In the case study, we have chosen Web Service SQL Injection attack for testing. At first step of testing scenario, we've presented normal request from client and normal response from server. Then we've shown the malicious request and the successful injection attack response. Finally, after enabling our XML firewall, we've shown that the firewall prevented injection attack and generated appropriate SOAP Response. This module is in the first

phase; in the next section we explain future works which we are going to do to improve our Apache XML firewall module.

Acknowledgement

The authors would like to thank the Iran Education and Research Institute for ICT (ERICT) for its financial support.

References

[1] Don Patterson, "XML Firewall Architecture and Best Practices for Configuring and Auditing," 2007.

[2] Nils Agne Norbotten, "XML and Web Services Security Standards," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, vol. 11, 2009.

[3] Abhinay Reddyreddy Haiping Xu, "Securing Service-Oriented Systems Using State-Based XML Firewall," , 2008.

[4] Nils Gruschka, Ralph Herkenhoner Meiko Jensen, "A Survey of Attacks on WebServices," in *Springer-Verlag*, 2009.

[5] Eduardo B. Fernandez, "Two Patterns for Web Services security," in International Symposium on Web Services and Applications, Las Vegas, 2004.

[6] Walid Negm, "Anatomy o f a Web Services Attack," Forum System Inc., 2004.

[7] "Web Server Survey, http://news.netcraft.com/archives/category/web-server-survey/," Netcraft, Jan 2011.

[8] Nick Kew, *The Apache Modules Book*.: Prentice Hall, 2007.

[9] Khaghani, M. 2010. *A Solution for Securing SOAP-Based Web Services.* Thesis, (B.S) Sharif University of Technology, Tehran, Iran.